WHAT FILE-LEVEL HITS HIDE IN BUG LOCALIZATION: PATHS, FUNCTIONS, AND REPAIR

Anonymous authors
Paper under double-blind review

ABSTRACT

Bug localization narrows an issue report to code that needs inspection or repair. File recall measures whether the right file is found, but does not distinguish useful path cues from code evidence or establish that the edit site has been located. Controlled experiments with CodeGRIP reveal both gaps. Changing displayed paths while holding reports, code previews and candidate pools fixed, we find that readable names from unrelated repositories can be more disruptive than opaque identifiers even when code remains visible. On issue-held-out SWE-bench Lite, this PathAlias intervention lowers Recall@1 by 21.8 percentage points for a code-in-prompt Qwen2.5-14B recipe without path augmentation and 4.9 points with it. Separate matched code-preview controls reveal sharply different opaque-path performance behind similar original-path scores. On GREPO, following file predictions through function selection, about one quarter of file-hit records across four rerankers also yield a changed-function hit under their own scorers. Under either tested shared scorer, neither code-visible reranker has a reliable advantage over path-only on common file wins. In separate Lite repair populations, correct supplied functions are associated with higher resolution among non-empty patches, whereas file-given comparisons remain inconclusive. File recall therefore needs to be read alongside path interventions and function outcomes, with repair interpreted under the location supplied to the fixer.

1 INTRODUCTION

Bug localization turns an issue report into code to inspect or change. For file-level systems, progress is commonly measured by how highly a file changed by the reference fix is ranked (Jimenez et al., 2024; Xia et al., 2025; Zhang et al., 2024; Chen et al., 2025; Reddy et al., 2026). This is a useful measure of search success, but the same credit can conceal differences in the evidence used and in how much of the remaining localization problem has been solved.

A file can be retrieved through its name, its code, or both. Names are often informative: developers name files after what they do, and some pipelines choose files from the repository tree before opening any (Xia et al., 2025). For a report that login fails after a token refresh, `auth/login.py` is a sensible guess from its path alone. Yet a reranker that scores each candidate from the issue, its path and a code preview can rank the right file first through either route or both. The preview makes code-based ranking possible, not necessary: the ranking loss rewards whichever visible cue puts a changed file first, and the path can become a shortcut when names cease to identify the relevant code (Geirhos et al., 2020). Recall@1 gives the same credit regardless of this route and therefore cannot distinguish them.

This ambiguity matters on both sides of a file prediction. On the input side, a model that can use code when names become misleading may behave differently from one that depends on the names, even when their original-path scores are similar. On the output side, finding a file leaves the choice of a function unresolved. A complete localizer must still select a location within its retrieved files, and a repair model given an entire file faces a different task from one given a specific function. File recall measures success at the file boundary, not these additional distinctions.

Input-ablation baselines have exposed analogous gaps between benchmark performance and the information needed to obtain it, as in hypothesis-only natural language inference (Poliak et al., 2018;

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.

Gururangan et al., 2018). In code localization, textual-feature controls, keyword-agnostic queries and repository-context perturbations examine related concerns (Gupta et al., 2025; Xu et al., 2026; Li et al., 2026). Yet removing a name does not show whether a reranker will follow a readable wrong name while code remains visible, and perturbing an entire agent also changes what it retrieves. Likewise, separate aggregate file and function scores do not show how often the same file hits survive the subsequent function-selection procedure.

We examine these questions in CodeGRIP, whose rerankers score candidate files from an issue report, a displayed path and, in code-visible variants, a source preview. The main path experiments use issue-held-out SWE-bench Lite cases and a candidate pool retrieved once from the original repository. PathAlias supplies readable unrelated names and PathSwap opaque identifiers, with reports, candidate identities and code previews fixed. Single-pair exchanges, historical names and matched code-input controls complement these interventions. On GREPO, we follow file predictions through function selection, comparing complete procedures and shared-scorer results. Separate Lite repair studies distinguish file-given from function-given inputs.

The path experiments show that readable wrong names can be more disruptive than opaque identifiers on Lite even when code remains visible. In a separate matched 14B training comparison, code-visible and path-only models achieve similar original-path scores but sharply different performance under opaque paths. A three-seed 7B factorial further reveals a trade-off between original-path accuracy and robustness to path changes. File recall alone leaves these differences unresolved.

Our contributions.

1. **Controlled tests of the evidence behind file hits.** Path interventions and matched code-input controls expose behavior that similar original-path scores can conceal (§2–4).
2. **File-to-function follow-through.** We trace each system's file hits through its function-selection procedure and compare file rerankers under shared function scorers. Roughly one quarter of own-scorer file-hit records also yield a changed-function hit (§5).
3. **Repair under the location actually supplied.** Separate file-given and function-given studies distinguish the unresolved file-given comparison from positive associations with correct supplied functions (§6).

2 Testing Paths on a Fixed Candidate Pool

Scoring interface. CodeGRIP provides the localization testbed used to train and evaluate our rerankers. Its full workflow proposes files from an issue report, expands the candidates with retrieval and repository relations, and reranks them (Appendix J). The main SWE-bench path experiments start after retrieval: a BM25 top-100 file pool is built once from the original paths and kept fixed. A LoRA-adapted Qwen2.5 model scores one report–file pair at a time from the report, the displayed path and, in code-visible variants, a 30-line source preview. Its score is $s_\theta(q, f) = z_{\text{Yes}} - z_{\text{No}}$. Changing only the displayed path therefore tests the ranking of the same files with the same code available.

Training the reranker. For a positive file f_0 and negatives $f_1, \ldots, f_m$, training uses the group-softmax loss

$$\ell(q; f_0, \ldots, f_m) = -\log \frac{\exp s_\theta(q, f_0)}{\sum_{j=0}^{m} \exp s_\theta(q, f_j)}. \tag{1}$$

Each report–file pair is encoded separately. The denominator couples their scores during training, while inference ranks independently scored candidates. The documented SWE-bench adaptation recipe uses 32 negatives, 75% drawn from BM25-hard candidates and 25% at random. Its path augmentation changes the displayed path while retaining the code preview and original target. The historical GREPO recipe uses a different negative mixture; Appendix J gives both configurations.

Four path interventions. All four change only displayed paths (construction and preregistration record in Appendix B).

PathAlias replaces repository-specific path components with readable alternatives from unrelated repositories, matching extension, depth, separators, casing, and character- and tokenizer-length bins. One-to-one mappings within each repository transform every candidate; common components such as `tests` remain. Ten mappings were fixed before scoring.

PathSwap replaces each component with a stable opaque identifier, keeping depth, extension and co-location (`cirq/value/linear_dict.py` becomes `d_f4e8/d_a406/m_008e.py`). It is a reference for removing readable path content, not a deployment scenario.

Path exchange swaps the displayed paths of a correctly ranked gold file and one distractor (the highest-ranked distractor, a same-directory file or a random file), keeping both files' code and the rest of the pool fixed.

Historical renames show the gold file under its earliest path in repository history, for the 862 GREPO issues whose gold file was renamed.

Rerankers, populations and statistics. The primary reranker is a Qwen2.5-14B code-in-prompt model trained without path augmentation; a matched recipe in which half of the training examples show shuffled paths, with code fixed, is the comparison recipe (Codeaware in the appendices). The matched code/no-code 14B pair and the three-seed 7B factorial of §4 are separate checkpoints. SWE-bench Lite is the primary path population: none of its 300 issues is among the 1,870 used for adaptation, though its 12 repositories are. In Verified, 384 of 500 issues occur in the adaptation set. The preregistration had named Verified primary. After scoring, an outcome-blind identifier audit found this overlap, so Lite became the primary evaluation and Verified a sensitivity analysis. No cell, seed or result changed. Function analyses use GREPO's 1,704 Python test issues. File accuracy is fractional Recall@1 (a correct first file counts $1/|G|$ for gold set G); analyses conditioned on a file hit use the binary event that the first file is gold. Intervals are paired bootstrap 95% CIs (Appendix A); a contrast whose interval includes zero is *unresolved*, which is not evidence of equivalence.

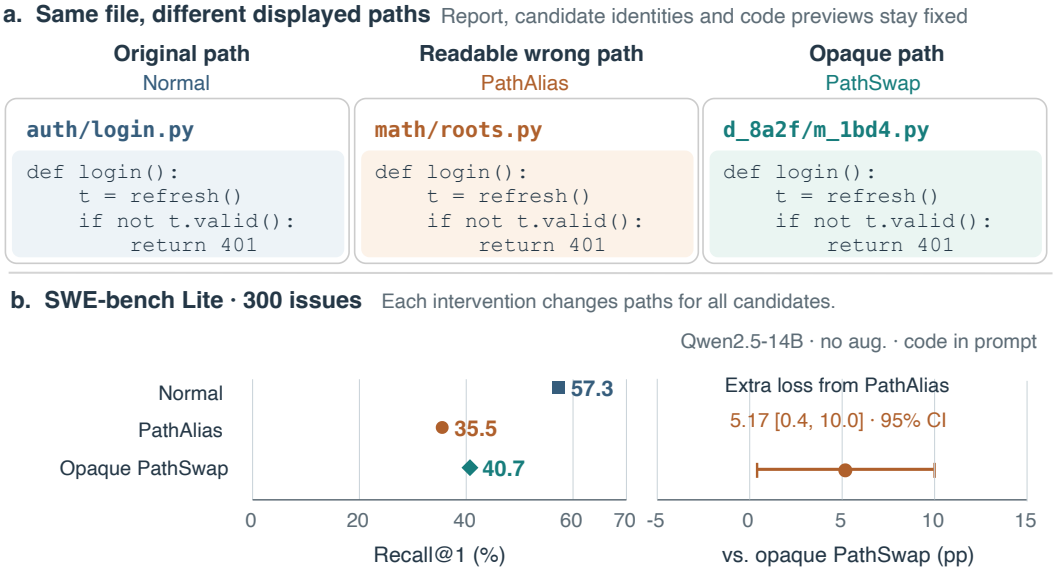


Figure 1: **Readable wrong names can be more disruptive than opaque names while code remains visible.** (a) Three path displays for one illustrative file; reports, candidate identities and code previews stay fixed. (b) The no-augmentation, code-in-prompt Qwen2.5-14B reranker on 300 issue-held-out Lite cases with fixed BM25 top-100 pools. Absolute Recall@1 values are point estimates; PathAlias averages ten mappings. The separate direct contrast uses a crossed instance-by-mapping 95% CI. Full mapping-level and Verified results are in Figure 3.

3 READABLE WRONG PATHS REDIRECT CODE-VISIBLE RERANKERS

3.1 READABLE WRONG PATHS ARE MORE DISRUPTIVE THAN OPAQUE IDENTIFIERS ON LITE

PathAlias and PathSwap remove the original names in different ways. Comparing them tests whether a plausible unrelated name is more disruptive than an opaque identifier while the same code remains visible (Figure 1).

On Lite, PathAlias lowers the primary reranker's Recall@1 from 57.3% to 35.5%, a paired change of -21.8 points (crossed instance-by-mapping CI [-27.1, -16.8]); each of the ten mappings costs 19.7–24.7 points. Opaque identifiers lower it to 40.7%, so a readable wrong path costs 5.17 points more than an opaque one (CI [0.4, 10.0]). Readable unrelated names are therefore more disruptive than opaque identifiers in this Lite comparison, despite an unchanged code preview. On the full Verified set, PathAlias and PathSwap cost 17.1 and 14.2 points, respectively, and their difference is unresolved (+2.86, CI [-0.2, 5.9]), so the ordering is established on Lite only; PathAlias still costs 14.2 points on the 116 Verified issues outside the adaptation set. Aliases sometimes share generic words such as `port` with the issue, which can bias the estimate in either direction; a construction excluding all issue vocabulary failed its preregistered feasibility limit. Masking the path text instead costs 21.7 points, not detectably different from PathAlias (Table 13).

3.2 EXCHANGING TWO FILES' PATHS MOVES THE TOP PREDICTION

PathAlias changes every path at once. The exchange asks which cue prevails when a file's name and its code point to different files.

Table 1: Single-pair path exchange. The gold file and a distractor exchange displayed paths while code and the rest of the pool stay fixed. Capture is the percentage of a recipe's original correct-file predictions that move to the distractor, with 95% bootstrap CI; G is that conditional number of original wins. Verified uses the 116 adaptation-disjoint issues. Full tier sizes, reference rates and path-margin shares are in Table 12.

Distractor	Recipe	Lite capture [CI]; G	Verified capture [CI]; G
Top-ranked	No augmentation	77.9 [71.5, 83.7]; 172	84.1 [72.7, 93.2]; 44
Top-ranked	Codeaware	18.6 [13.1, 24.6]; 183	4.9 [0.0, 12.2]; 41
Same directory	No augmentation	80.2 [74.1, 86.4]; 162	79.5 [66.7, 92.3]; 39
Same directory	Codeaware	11.7 [7.0, 17.0]; 171	11.1 [2.8, 22.2]; 36
Random	No augmentation	63.4 [55.8, 70.3]; 172	77.3 [63.6, 88.6]; 44
Random	Codeaware	1.1 [0.0, 2.7]; 183	0.0 [0.0, 0.0]; 41

Among Lite issues that the no-augmentation reranker ranks correctly, swapping the displayed paths of the gold file and one distractor moves the top prediction to the distractor in 77.9% of cases for the highest-ranked distractor, 80.2% for a same-directory file and 63.4% for a random file (CI [55.8, 70.3]; 172 issues), although a random distractor held the top rank on original paths in only 0.4% of the full tier (Table 12). On the 116 adaptation-disjoint Verified issues, those no-augmentation capture rates are 77–84%. The path-augmented recipe follows exchanged paths far less often (18.6%, 11.7% and 1.1% on Lite), a descriptive difference, since no test between recipes was preregistered. Because the test constructs a conflict, it shows which cue prevails when name and code disagree, not how often ordinary predictions rely on names.

3.3 REAL RENAMES AND OTHER RERANKERS

Synthetic paths could exaggerate what happens in practice, so we also use renames recorded in repository history. On the 862 GREPO issues whose gold file was renamed, showing it under its earliest path lowers Recall@1 by 3.47 points (CI [-5.12, -1.82]) for the no-augmentation 14B reranker and by 2.22 points ([-3.64, -0.84]) for the path-augmented one; their difference is unresolved, and recall at cutoffs 5 and 10 rises. An outdated name can thus cost top-1 accuracy with the code unchanged, at a far smaller scale than synthetic replacement; the calibration does not estimate how often renames cause failures in practice.

The sensitivity is not confined to one model family or ranker type. Path-only rerankers built on Qwen2.5-7B, Llama-3.1-8B and Qwen3-8B lose 73–75% of their Recall@1 when filenames are shuffled and 95–96% under PathSwap (Table 30), and filename shuffling also lowers accuracy for a bi-encoder, a listwise generator and frontier models prompted with code (Appendix G). The exception is SweRank's native function-body pipeline, which shows no significant PathSwap change; our reproduction reaches 20.8% file accuracy against a published 78.1%, so that case remains unresolved (Appendix I).

4 Original-Path Recall@1 Can Conceal Code Use

The matched training controls show how similar original-path scores can conceal different behavior when readable names are removed.

Table 2: Code-preview controls at training and inference time. Values are fractional Recall@1 (%); changes are in percentage points. Panel A compares separately trained Qwen2.5-14B LoRAs (seed 42, Lite $n = 300$); Panel B changes only the preview of one fixed 7B Codeaware checkpoint. Both use BM25 top-100 pools. Full counts and paired tests remain in Tables 24 and 27.

A. Matched training inputs: code present minus code absent			
Condition	With code	Without code	Δ pp
Normal	57.00	55.67	+1.33
PathSwap	39.67	0.33	+39.33
B. Fixed checkpoint: redacted minus original preview			
Split / condition	Original	Redacted	Δ pp
Lite / Normal	49.33	48.33	−1.00
Verified / Normal	43.32	41.80	−1.52
Lite / PathSwap	40.67	26.67	−14.00
Verified / PathSwap	32.23	24.53	−7.70

At the matched 768-token limit, separately trained Qwen2.5-14B rerankers with and without code previews reach 57.0% and 55.7% Recall@1 on original Lite paths. Their preregistered primary difference is unresolved (+1.33 points; 95% CI [-3.00, 5.67]), whereas their PathSwap scores are 39.7% and 0.3% (Table 2A). These checkpoints are separate from Figure 1's path-intervention runs; Appendix B records the operating points. Similar original-path recall can therefore conceal a substantial difference in performance when readable names are removed. This supports the value of code in the perturbed setting, without establishing how much either model's original-path ranking depends on code.

Table 3: Complete code-preview/path-augmentation factorial. Qwen2.5-7B is trained on SWE-only data with seeds 42/43/44, with 30-line previews in code-present cells, then evaluated on Lite ($n = 300$). Fractional Recall@1 is reported as mean ± seed standard deviation. These are the corrected four-cell checkpoints; full effects and separate seed-42 tests are in Tables 25 and 26.

Path augmentation	Normal Recall@1 (%)		PathSwap Recall@1 (%)	
	No code	Code preview	No code	Code preview
0%	53.00 ± 0.88	54.33 ± 1.20	1.44 ± 0.51	30.22 ± 3.24
50%	50.00 ± 1.00	48.56 ± 2.59	3.11 ± 1.58	36.78 ± 4.86

A three-seed 7B factorial trained on SWE-bench data crosses code previews with path augmentation (Table 3). At 0% and 50% path augmentation, respectively, code previews change original-path Recall@1 by +1.3 and -1.4 points, but PathSwap Recall@1 by +28.8 and +33.7. With code previews, augmentation raises PathSwap Recall@1 by 6.6 points and lowers original-path Recall@1 by 5.8, so original-path accuracy and opaque-path robustness favor different recipes; the interaction (+4.9 points) is unresolved with three seeds. When a fixed 7B path-augmented reranker sees previews with 80% of body lines replaced by placeholders (imports, signatures and the first non-blank line kept), accuracy moves by -1.0 and -1.5 points on the original paths in Lite and Verified (neither significant)

Table 4: Selective code retrieval on original GREPO paths. The full-test results cover 1,704 issues and report mean ± seed standard deviation (three path-only seeds, two selective-retrieval seeds). The 116-issue Code-Crucial slice reports the retained point estimates; no paired interval is available. These rows compare the recorded procedures, rather than the matched preview-only intervention of Table 2. Appendix F retains the remaining code-reading variants.

Procedure	Full-test Recall@1 (%)	Code-Crucial Recall@1 (%)
Path-only	28.92 ± 0.56	35.34
Three retrieved functions per file	28.44 ± 0.62	40.52

but by -14.0 and -7.7 under PathSwap (Table 2B). We read these controls behaviorally: code serves as a fallback when readable paths are absent, and the redaction probe does not separate body semantics from the retained headers.

The 14B recipes of §3 show the same blind spot. The path-augmented reranker loses 4.9 points under PathAlias on Lite (CI [-8.3, -1.7]) against 21.8 for the no-augmentation reranker, and its extra loss relative to opaque paths is smaller by 7.27 points on Lite (CI [1.87, 12.73]) and 3.54 on Verified ([0.47, 6.58]). Yet the recipes' original-path difference is unresolved on both Lite (+4.0 points, CI [-0.01, 8.3]) and Verified (+0.3, CI [-1.9, 2.5]; Appendix E). On GREPO, likewise, a reranker that never sees code reaches 29.7% file Recall@1, and its differences from two 14B code-in-prompt rerankers (28.9% and 29.2%) are unresolved.

Where code changes the ranking. The matched comparisons show that ordinary file recall can hide different uses of code, but they do not imply that code helps only after a synthetic path intervention. We also examine cases where the original path is a poor guide. Code-Crucial contains 116 GREPO issues selected by low issue–path overlap, misleading path cues and available code evidence. Using the top three functions ranked by BM25 from each candidate file yields higher Recall@1 on this slice than path-only scoring (Table 4). Merely supplying a longer preview does not give the same result: the tested short-preview residual and zero-shot long-context variants both score about 13% on this slice (Appendix F). What code reaches the scorer matters as well as whether code is present.

The selective-retrieval score is higher on Code-Crucial without a detectable full-test advantage; it also loses 2.1 points on the duplicate-filename slice. Thus the positive slice result and the similar aggregate scores are compatible. The original paths are left intact here, so this comparison complements the synthetic interventions with cases in which code can be useful under the ordinary input. Taken together, the results motivate reporting where a training or input change helps, alongside its aggregate file recall.

Nor did cheap fixes reliably remove the dependence (Appendix H): an opaque-path training continuation passed its screens, but a matched readable-alias continuation failed when Verified PathSwap accuracy fell 2.6 points; telling the reranker that paths are unreliable changed nothing measurable; and a preregistered rescoring rule failed its criterion.

5 File Hits Often Leave the Function Unlocalized

5.1 How often a file hit reaches the function

A file-level hit records that the first-ranked file is changed by the reference fix. The complete localization procedure must still select a function: each system scores functions drawn from its ten highest-ranked files, and a function hit occurs when the first-ranked function matches a changed function by file path and function name. We measure transfer as the fraction of binary file-hit records that also yield a function hit. This measures the procedure's selected output, rather than whether the first file contains a changed function.

The two outcomes need not be nested. A correct first file can be followed by a wrong function selection, while a changed function can be recovered from a lower-ranked file when the first file is wrong. Figure 2 illustrates both possibilities and reports their joint outcomes. Evaluating each system

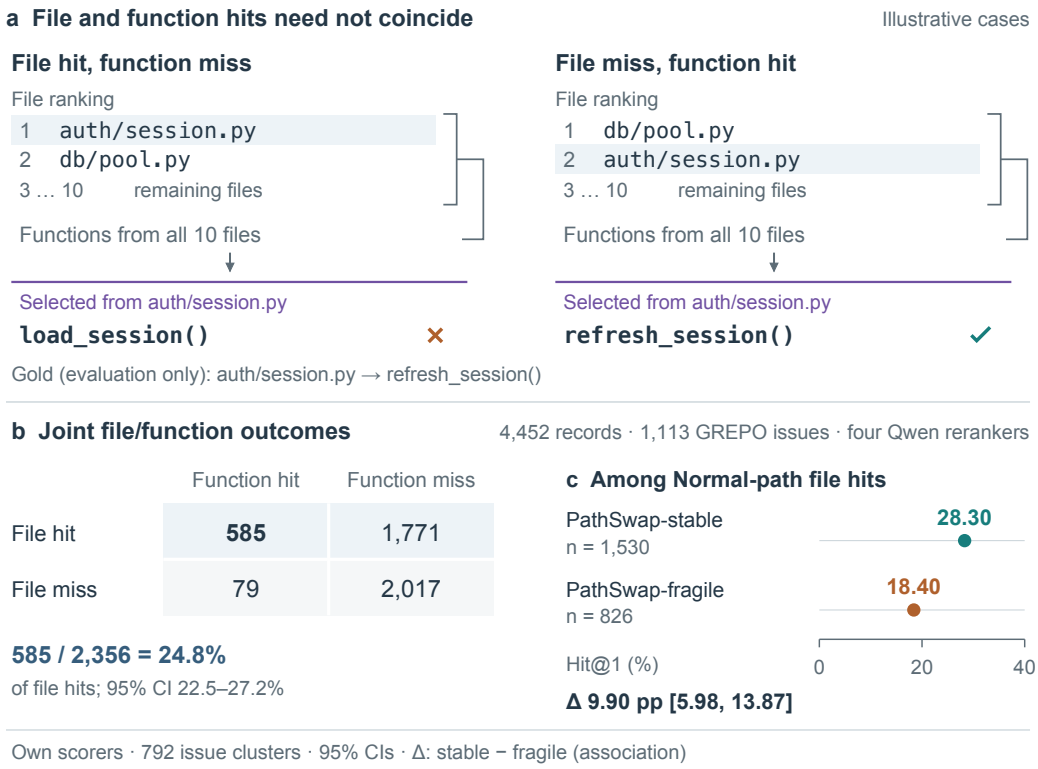


Figure 2: **A file hit need not deliver a function hit.** (a) Illustrative selections from all ten highest-ranked files; gold file/function labels are used only for evaluation. (b) Joint outcomes for four Qwen rerankers on 1,113 common label-defined GREPO issues, each using its own function scorer. (c) Normal-path file hits that remain hits under PathSwap (stable) versus those that become misses (fragile); stability is observed, not assigned. These are function-localization outcomes, not repair outcomes. Table 17 retains the complete numerical table.

with its own function scorer describes the complete procedure; fixing a shared function scorer in the next comparison separates that procedure-level result from what the file reranker contributes.

The population is 1,113 issues labeled for four Qwen2.5 rerankers (7B and 14B, with and without path augmentation): 4,452 system–issue records, of which 2,356 are file hits. Only 585 of these file-hit records also yield a function hit (24.8%, issue-clustered CI [22.5, 27.2]; Figure 2b). The 79 function hits under a wrong first file come from lower-ranked files in the top-10 search. A preregistered Lite replication gives 24.2% (123 of 509 file-hit records); a blinded audit flags 3 of 100 sampled labels, and excluding all eight flags among the 279 labels gives 24.6%. How a hit was earned matters: hits that survive PathSwap transfer more often than hits PathSwap destroys, 28.3% against 18.4% (+9.9 points, CI [6.0, 13.9], clustered over 792 issues; Figure 2c), a gap resolved for three of the four rerankers and for a Llama-3.1-8B replication. Stability is observed rather than assigned, so this is an association, and most path-stable hits still miss the function.

5.2 Separating File Ranking from Function Scoring

Complete procedures and file rerankers answer different comparison questions. Table 5A evaluates complete procedures with their own function scorers on the same 1,113 issues: function Hit@1 is 25.1% for the path-only procedure and 18.7% and 18.4% for the two 14B procedures, despite unresolved differences in file Recall@1. Here, "path-only" describes the file-ranking input, not the inputs of every stage in the complete procedure. Table 5B instead holds each system's file-hit records fixed and changes the function scorer. The fixed-neutral scorer is an unadapted Qwen2.5-7B-Instruct model receiving the bug report, real file path, file summary, and candidate function name. The name-blind probe uses the same weights and prompt but replaces the candidate-name field with an

Table 5: Complete procedures and scorer-controlled comparisons. Panel A gives marginal file and function scores on 1,113 common GREPO issues using own function scorers. Panel B changes the function scorer on each system's fixed label-available file-win records; the two systems have different record sets. Panel C compares each named reranker with path-only on their common wins, fixing the shared function scorer. For Panel C, Δ is the named reranker minus path-only. All four paired repository–issue bootstrap 95% CIs include zero.

A. Complete procedures on the same 1,113 GREPO issues

System	File Recall@1 (%)	Function Hit@1 (%)
Path-only	29.70	25.1
14B Codeaware	28.89	18.7
14B no-augmentation	29.22	18.4

B. Within-system scorer changes on fixed file-win records

14B reranker	n	Own (%)	Fixed neutral (%)	Name-blind (%)
Codeaware	649	28.5	8.32	5.86
No augmentation	652	27.9	7.82	5.37

C. Paired reranker contrasts on common file wins

Shared scorer / 14B reranker	n	Δ (pp)	95% CI (pp)
Fixed neutral / Codeaware	468	-0.21	$[-1.71, 1.28]$
Fixed neutral / no augmentation	476	-0.63	$[-2.52, 1.26]$
Name-blind / Codeaware	472	-0.85	$[-2.75, 1.06]$
Name-blind / no augmentation	481	-1.04	$[-2.91, 0.83]$

opaque token such as `fn_0000`; the file path and summary remain available. On the same 649 Codeaware file hits, transfer is 28.5% with the system's own scorer, 8.3% with the fixed-neutral scorer, and 5.9% with the name-blind probe.

The shared-scorer comparisons ask whether file rerankers yield different function outcomes on issues where both rank a gold file first (Table 5C). Each reranker retains its own top-10 function pool. Across 468–481 common-win records, code-visible-minus-path-only differences are -0.2 to -1.0 percentage points; all four intervals include zero. Neither shared scorer establishes a code-visible advantage, and unresolved contrasts do not imply equivalence. Fixed-neutral and name-blind use different path-only snapshots (4-bit/top-200 and bf16/top-100), so each row remains paired within its own population. Name masking retains file paths and summaries, leaving function-body localization untested. Panel A compares complete own-scorer procedures.

6 Function Correctness and Executed Repair

A repair model given a whole file must choose where to edit; one given a function starts from a specified location. We examine function correctness under both inputs using separate SWE-bench Lite repair populations. In each population, certified file hits are those where the localization procedure also selects a correct function, and generated patches are evaluated with the official SWE-bench tests (Table 6).

Each observation is a localization record supplied to a fixer; one issue can contribute records from more than one locator. Confidence intervals therefore resample issues rather than treating all attempts as independent. The primary non-empty-patch rate measures success conditional on producing a patch. The all-attempt rate also counts empty outputs as unsuccessful. Reporting both makes the denominator explicit before comparing the two input interfaces.

For file-given repair, an outcome-independent source check excluded 24 of the original 208 attempts, all uncertified, because the runner had received another locator's file. The corrected population contains 184 attempts over 105 issues. Among non-empty patches, 18 of 50 certified records resolve the issue, compared with 15 of 76 uncertified records, a difference of +16.3 percentage points with a 95% issue-clustered interval of [-4.1, 36.7]. The all-attempt difference is also unresolved

Table 6: Executed repair under two supplied inputs. Counts are resolved/eligible records. Panel A excludes 24 source mismatches; both corrected issue-clustered 95% intervals include zero. Panel B uses separate function-conditioned populations and non-empty patches; Qwen3 fixers cover twelve Lite repositories, Llama seven. These positive associations are not matched improvements over Panel A. Full patch yields, coverage and source rules are in Appendix D.

Input / analysis or fixer	Certified	Uncertified	Δ (pp)	95% CI
A. File-given, Qwen3-14B; corrected 184 attempts / 105 issues				
Non-empty (primary)	18/50	15/76	+16.3	[−4.1, 36.7]
All attempts (secondary)	18/67	15/117	+14.0	[−1.2, 29.6]
B. Function-conditioned; non-empty patches				
Qwen3-14B	23/75	1/85	+29.5	[16.4, 43.4]
Qwen3-8B	20/65	0/58	+30.8	[16.1, 46.0]
Llama-3.1-8B	17/67	1/125	+24.6	[11.4, 38.6]

(+14.0 points, [-1.2, 29.6]). Thus the recorded file-given attempts do not establish whether function certification distinguishes repair success when the fixer chooses its own edit site.

When the repair interface supplies a specific function to rewrite, correct supplied functions are associated with higher resolution among non-empty patches. Qwen3-14B resolves 23 of 75 such patches for correct functions, versus 1 of 85 for incorrect functions (+29.5 percentage points; 95% CI [16.4, 43.4]); the Qwen3-8B and Llama-3.1-8B contrasts are also positive. These associations describe repair when the fixer receives a specific function. They do not establish a matched improvement over file-given repair or a causal effect of changing the supplied location; the corrected file-given result remains inconclusive. Appendix D retains execution coverage, the original pre-correction diagnostics, and the mixed released-system checks.

7 Related Work and Conclusion

Input ablations and counterfactual edits motivate our controls: the former test what one input can support, while the latter change a cue with the remaining input fixed (Poliak et al., 2018; Gururangan et al., 2018; Jia & Liang, 2017; Yefet et al., 2020; Rabin et al., 2021). In localization, SACL studies textual bias (Gupta et al., 2025), LogicLoc studies localization with keyword-agnostic queries (Xu et al., 2026), and RepoMirage perturbs repository context for agents (Li et al., 2026). Our intervention instead changes displayed paths at one scoring stage over a fixed candidate pool. Readable PathAlias, opaque PathSwap and historical renames distinguish different changes to the same source of information.

Work on hierarchical localization and repair shows why finding a file is an intermediate outcome (Chang et al., 2025; Al Awad & Ivanov, 2026; Sepidband et al., 2026). We follow file hits through both complete function-selection procedures and shared-scorer comparisons, then examine repair under the location actually supplied to the fixer. Appendix K places these checks within the broader retrieval, repository-graph and agent literature.

Limitations. The path results concern pointwise rerankers on fixed pools; listwise systems and agents that explore repositories may behave differently, and the native SweRank check found no significant PathSwap change. The main models use Qwen2.5 and our training recipes. Lite shares repositories and Verified shares issues with adaptation data, with one Verified repository exceeding the concentration limit; the zero-shot Llama check missed its adequacy floor. Function rates depend on the scorer and label population, repair contrasts are associations, and mitigation pilots are single-seed. The relevant protocol details and counterexamples accompany each analysis in the appendices.

Conclusion. File recall can conceal sensitivity to displayed paths and leave function localization incomplete. Path interventions, matched code-input controls and scorer-specific function checks reveal these gaps. Repair must be assessed under the file or function supplied to the fixer.

Reproducibility Statement

The main text and appendix specify the fixed-pool estimands, interventions, model and evaluation configurations, pre-registration amendments, and uncertainty procedures. The anonymous supplementary material contains analysis and verification code, pre-registered result summaries, and a machine-readable claim-to-evidence index for the included supplementary files, with integrity identifiers; items not included anonymously are identified as post-publication releases. The Verified overlap correction only relabels populations; evaluated cells, seeds and outcomes stay fixed. Benchmark records and path interventions are constructed as described in the methods; model predictions and repair patches are experimental outputs, not human labels.

Ethics Statement

All data used in this work come from public software-engineering benchmarks, including SWE-bench Lite and Verified, GREPO, BeetleBox, and related public benchmark materials, and all evaluated models are publicly released. The study involves no human participants and no personal data. PathAlias, PathSwap, and MaskPath are controlled evaluation perturbations; PathMask is applied during training. None of these interventions renames real repositories. Materials released with the anonymous submission do not identify the authors.

AI Use Statement

The research questions, methods, experimental design and scientific conclusions were developed by the authors. Generative AI tools assisted the organization and presentation of research materials, language editing and translation, figure preparation, and checks of experimental and analytical rigor, including related supporting code. These tools were used under author direction; scientific judgments and responsibility for the final manuscript remain with the authors.

References

Mohammad Nour Al Awad and Sergey Ivanov. Loc2repair: A framework for evaluating the impact of file-level issue localization in repo-level LLM repair, 2026.

Genevieve Caumartin, Tse-Hsun Peter Chen, and Diego Elias Costa. Retrieval-oriented code representations in agentic bug localization, 2026. URL https://arxiv.org/abs/2607.11046.

Partha Chakraborty, Mahmoud Alfadel, and Meiyappan Nagappan. Blaze: Cross-language and cross-project bug localization via dynamic chunking and hard example learning. *IEEE Transactions on Software Engineering*, 51(8):2254–2267, 2025. doi: 10.1109/TSE.2025.3579574.

Jianming Chang, Xin Zhou, Lulu Wang, David Lo, and Bixin Li. Bridging bug localization and issue fixing: A hierarchical localization framework leveraging large language models, 2025. URL https://arxiv.org/abs/2502.15292.

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 the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8697–8727, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.426. URL https://aclanthology.org/2025.acl-long.426/.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 758–759. ACM, 2009. doi: 10.1145/1571941.1572114.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jiarui Feng, Donghong Cai, Yixin Chen, and Muhan Zhang. GRIP: In-parameter graph reasoning through fine-tuning large language models. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2026. doi: 10.1145/3770855.3817895.

Luyu Gao, Zhuyun Dai, and Jamie Callan. Rethink training of BERT rerankers in multi-stage retrieval pipeline. In *Advances in Information Retrieval (ECIR)*, volume 12657 of *Lecture Notes in Computer Science*, pp. 280–286. Springer, 2021. doi: 10.1007/978-3-030-72240-1_26.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020. doi: 10.1038/s42256-020-00257-z.

Dhruv Gupta, Gayathri Ganesh Lakshmy, and Yiqing Xie. SACL: Understanding and combating textual bias in code retrieval with semantic-augmented reranking and localization. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 25052–25065, 2025. doi: 10.18653/v1/2025.findings-emnlp.1365. URL https://aclanthology.org/2025.findings-emnlp.1365/.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 107–112, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2017. URL https://aclanthology.org/N18-2017/.

Xuan Huo, Ming Li, and Zhi-Hua Zhou. Learning unified features from natural and programming languages for locating buggy source code. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1606–1612, 2016. URL https://www.ijcai.org/Proceedings/16/Papers/230.pdf.

Mathew Jacob, Erik Lindgren, Matei Zaharia, Michael Carbin, Omar Khattab, and Andrew Drozdov. Drowning in documents: Consequences of scaling reranker inference. In *Proceedings of the 4th Workshop on Reaching Efficiency in Neural Information Retrieval (ReNeuIR 2025) at SIGIR 2025*, Padua, Italy, 2025. URL https://arxiv.org/abs/2411.11767.

Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2021–2031, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1215. URL https://aclanthology.org/D17-1215/.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Zhonghao Jiang, Xiaoxue Ren, Meng Yan, Wei Jiang, Yong Li, and Zhongxin Liu. Issue localization via LLM-driven iterative code graph searching. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 3034–3045. IEEE, 2025. doi: 10.1109/ASE63991.2025.00249.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. Bug localization with combination of deep learning and information retrieval. In *Proceedings of the 25th IEEE/ACM International Conference on Program Comprehension (ICPC)*, pp. 218–229. IEEE, 2017. doi: 10.1109/ICPC.2017.24.

Hanyu Li, Yichi Zhang, Speed Zhu, Hang Su, Jun Zhu, and Yinpeng Dong. RepoMirage: Probing repository context reasoning in code agents with perturbations, 2026. URL https://arxiv.org/abs/2605.26177.

Shanchao Liang, Spandan Garg, and Roshanak Zilouchian Moghaddam. The SWE-bench illusion: When state-of-the-art LLMs remember instead of reason. In *LAW 2025: Bridging Language, Agent, and World Models for Reasoning and Planning (Workshop at NeurIPS 2025)*, 2025. URL https://openreview.net/forum?id=ZJCyrBpgnW.

Xiangyan Liu, Bo Lan, Zhiyuan Hu, Yang Liu, Zhicheng Zhang, Fei Wang, Michael Qizhe Shieh, and Wenmeng Zhou. CodexGraph: Bridging large language models and code repositories via code graph databases. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 142–160, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.7. URL https://aclanthology.org/2025.naacl-long.7/.

Md Afif Al Mamun and Gias Uddin. BLAgent: Agentic rag for file-level bug localization. *ACM Transactions on Software Engineering and Methodology*, 2026. doi: 10.1145/3830238. URL https://doi.org/10.1145/3830238.

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 *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=dw9VUsSHGB.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737. PMLR, July 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. Hypothesis only baselines in natural language inference. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics (*SEM)*, pp. 180–191, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/S18-2023. URL https://aclanthology.org/S18-2023/.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5835–5847, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.466. URL https://aclanthology.org/2021.naacl-main.466/.

Md Rafiqul Islam Rabin, Vincent J. Hellendoorn, and Mohammad Amin Alipour. Understanding neural code intelligence through program simplification. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pp. 441–452. ACM, 2021. doi: 10.1145/3468264.3468539.

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 *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=OnkRqbNhe3.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019.

Melika Sepidband, Hung Viet Pham, and Hadi Hemmati. On the role of fault localization context for LLM-based program repair, 2026.

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 *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=iyJOUELYir.

Juntong Wang, Libin Chen, Xiyuan Wang, Shijia Kang, Haotong Yang, Da Zheng, and Muhan Zhang. GREPO: A benchmark for graph neural networks on repository-level bug localization. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 9916–9926, 2026. doi: 10.1145/3770855.3817529.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: Advancing LLM reasoning via reinforcement learning on open software evolution. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, 2025. URL https://openreview.net/forum?id=ULblO61XZ0.

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):FSE037, 2025. doi: 10.1145/3715754.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=zeFrfgyZln.

Xiufeng Xu, Xiuheng Wu, Zejun Zhang, and Yi Li. Neurosymbolic repo-level code localization, 2026. URL https://arxiv.org/abs/2604.16021.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bofei Zheng, Bowen Yu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Noam Yefet, Uri Alon, and Eran Yahav. Adversarial examples for models of code. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):162, 2020. doi: 10.1145/3428230.

Annie T. T. Ying, Gail C. Murphy, Raymond T. Ng, and Mark C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574–586, 2004. doi: 10.1109/TSE.2004.52.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pp. 1592–1604. ACM, 2024. doi: 10.1145/3650212.3680384.

Thomas Zimmermann and Nachiappan Nagappan. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th International Conference on Software Engineering (ICSE)*, pp. 531–540. ACM, 2008. doi: 10.1145/1368088.1368161.

Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005. doi: 10.1109/TSE.2005.72.

A Evaluation Protocols and Supplementary Guide

A.1 Questions and reading routes

The appendices follow the main questions: displayed paths in Appendix B, function transfer in Appendix C, and repair inputs, exclusions and outcomes in Appendix D. Recipe controls, supplementary diagnostics and scope studies follow in Appendices E, F and G. Exploratory outcomes and stopping decisions are in Appendix H; the native SweRank reproduction is in Appendix I; implementation and reference analyses are in Appendix J. Extended background is in Appendix K. Each retains its own population and protocol; order implies neither a causal chain nor a common experiment.

Table 7: **Evidence and primary boundary for each check.** COMP denotes the wrong-function-routing contrast.

Check	**Evidence**	**Boundary**
Sensitivity	PathAlias drops Recall@1 by 21.8pp on adaptation-disjoint Lite and 17.1pp on Verified. Under the matched path-augmented recipe the PathAlias drop shrinks to 4.9pp on Lite (CI $[-8.3, -1.7]$; Table 10).	Lite has 0/300 adaptation-ID overlap and strict Alias–PathSwap ordering; Verified has 384/500 overlap and is an adaptation-overlap-limited sensitivity. The result is specific to the tested training recipe; strict lexical closure remains open.
Transfer	Under no tested fixed shared scorer does a code-aware system obtain a reliable certification advantage over the code-blind comparator.	The 24.8% within-system level is scorer-sensitive; cross-system comparisons require one fixed shared scorer.
Consequence	Both source-corrected file-given repair contrasts include zero. On the original 208 attempts, only COMP excludes zero among the prespecified non-additive routing and residual contrasts.	The corrected and original populations differ. Positive function-conditioned contrasts are sanity checks under supplied localization; they do not resolve the file-given NULL or identify a causal routing effect.
Reporting	Pair file Recall@1 with a path intervention and within-system own-scorer function transfer.	Both-scorer certification means certification under both fixed shared scorers; it measures labeling sensitivity, not resolution.

Table 8: Comparison objects, units and outcomes for the three checks. Their reading order is not a sequence of per-instance causal effects.

Check	Fixed or supplied inputs	Analysis unit	Outcome and boundary
Path sensitivity	Fixed report, code, candidates and pool; changed displayed path	Paired evaluation record	Fractional file Recall@1 change under the specified path intervention
Function transfer	Functions from top-10 predicted files; named scorer	File-win record in a declared label/overlap set	Scorer-referenced conditional function Hit@1; common-win contrast
Repair outcome	Localized function or file; source-exclusion rule	Repair attempt or non-empty patch	Certified–uncertified resolution difference; association, not a causal effect

A.2 Populations and scoring instruments

GREPO (Wang et al., 2026) contains 47,166 Python reports across 86 repositories; we use 7,883 training examples and, from the 2,175-example test split, the 1,704 Python-changing test examples. SWE-bench Lite (n=300) is the primary cross-benchmark sensitivity population: it is issue-held-out within the same 12 adapted repositories by construction (0/300 issue overlap with the prespecified 1,870-issue SWE adaptation population). Verified (n=500) is a sensitivity population limited by adaptation overlap because 384 IDs occur in that adaptation set. GREPO candidate generation starts from BM25 top-500 and merges graph-expanded candidates. The primary SWE-bench sensitivity

analyses use a recipe trained without path-shuffling augmentation and **Codeaware**, a matched recipe trained with 50% path-shuffled examples; both rerank fixed BM25 top-100 pools unless a section states otherwise (retrieval details in Appendix J). These pools define different estimands, so their percentages are not comparable across pools. Unless noted, we report fractional Recall@1 with paired bootstrap (B=10,000). McNemar tests apply to binary top-1 any-hit for file contrasts, or to the explicitly named binary function/flip outcomes. We write NULL for a contrast whose interval includes zero. Audited systems are Qwen2.5-7B/14B-Instruct LoRAs (Yang et al., 2024), with Llama-3.1-8B (Dubey et al., 2024), Qwen3-8B (Yang et al., 2025), and Mistral-7B (Jiang et al., 2023) breadth checks; hyperparameters, data separation and graph provenance are in Appendices J.2.1–J.2.2.

Organic rename drift is the ecological calibration: on 862 instances it replaces the gold's displayed current path with its historical `git` path while holding content and pools fixed (Appendix B.5). In the appendix tables, "path-only baseline" means the Qwen2.5-7B three-seed GREPO mean of $28.92 \pm 0.56\%$; the cross-model perturbation tables instead use the single sparse-checkpoint baseline 27.01, so their relative drops are not comparable with the three-seed rows. We provide analysis and verification code in the anonymous supplementary material; transformed paths and candidate pools will be released upon publication.

Table 9: Main analysis populations and instruments. Counts name different units; they are not a common sample across checks.

Analysis	Population / unit	Instrument / boundary
PathAlias Lite	300 issues; 0 adaptation IDs	14B no-aug / Codeaware; fixed top-100
PathAlias Verified	500 issues; 384 adaptation IDs	sensitivity; ten alias mappings
Own-scorer transfer	1,113 issues; 4,452 system–instance pairs	2,356 wins; 585 joint hits; 792 win clusters
Shared-scorer transfer	468 / 476 / 472 / 481 common-win records	fixed-neutral / name-blind; own scorer excluded
Corrected file-given repair	184 attempts; 105 issues; 126 non-empty	24 source mismatches excluded; all uncertified
Original routing analysis	208 attempts	pre-exclusion; non-additive contrasts
Conditioned Qwen3-14B	264 eligible file wins; 160 non-empty	supplied function; two errors remain excluded

A.3 Amendments and material availability

The outcome-blind identifier audit changed the PathAlias population priority after scoring, without changing cells or outcomes (Appendix B). Fixed source exclusions and noncomputable repair contrasts are documented in Appendix D. Analysis code, current summaries and approved minimal internal count inputs have distinct scopes; missing raw labels, transformed paths, candidate pools and checkpoints are not made reproducible by their count projections. Frozen records retain their original versions; omitted assets retain the stated post-publication release boundary.

B Displayed-Path Interventions and Calibration

B.1 PathAlias construction, populations and uncertainty

PathAlias replaces every nonstructural path component with a readable, form- and tokenizer-matched alias drawn from unrelated repositories.

Most aliases are whole held-out donor components; a disclosed fallback composes held-out donor subtokens for 14/2,721 Lite and 140/5,739 Verified components per seed. The alias is a real path component from elsewhere: it preserves natural form but breaks the correct path–repository/content correspondence. Donor components are harvested from held-out repositories and matched on extension, path depth, separator style, case morphology (underscore, hyphen, trailing-digit, uppercase—checked independently), character-length bin, and BPE-token-count bin under the evaluated model's tokenizer. The mapping is one-to-one and repository-wide (the same original component maps to the same alias across all instances of a repo, shared across benchmarks per seed); it is applied to *all* candidate paths, not only gold. Opacity filters reject donors occurring in the target repository's own path vocabulary,

including at the lexical/BPE-subtoken level against the target repository's canonical name; accidental alias–report overlap is reported at three granularities, never hand-removed: exact whole-alias matches in issue text are rare (ten-seed means 0.43% Verified / 0.59% Lite of component–instance pairs); lexical subtokens (length $\geq$3) of aliases appear in the issue text for 4.6–5.4% (Lite) and 8.7–9.8% (Verified) of component–instance pairs, dominated by generic tokens such as `port` or `remove`; a broader lexical-or-BPE-piece diagnostic reaches 15–22%. With $\sim$300 aliased components per instance, nearly every instance contains *some* generic-token coincidence, so an exclusion-based sensitivity is non-identifying at the instance level. Residual coincidence has no fixed bias direction because aliases are applied to every candidate: overlap with a gold alias can attenuate the measured drop, whereas overlap with a distractor alias can inflate it. Across ten independent alias draws, every aggregate no-augmentation drop remains positive and the largest within-cell seed range is 5.1pp. This makes a single unlucky draw implausible, but does not exclude a generic lexical-coincidence component shared across mappings.

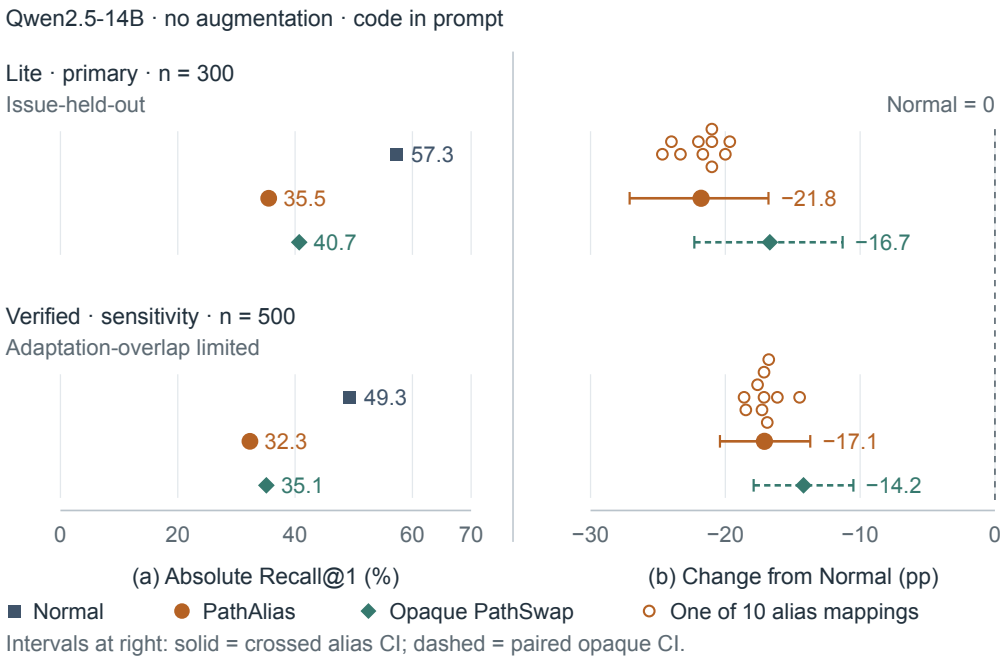


Figure 3: **Absolute ranking levels and paired changes under displayed-path interventions.** Both panels use the same no-augmentation, code-in-prompt 14B recipe. Open circles are ten alias mappings; filled circles and solid intervals show the retained PathAlias means and crossed 95% CIs. Diamonds and dashed intervals show the deterministic opaque reference and instance-paired CIs. The absolute panel shows point estimates only. Vertical separation prevents overlap; it is not a data axis.

Components appearing in $\geq$10% of repositories across the full benchmark corpus (e.g., `src`, `tests`, `__init__`) keep their original form, except that a component equal to any target repository's canonical name is never treated as structural.

The chain in UTC is: pre-registration 2026-07-09 23:48; implementation amendment 2026-07-10 03:30; full-population amendment 2026-07-10 04:27; recorded alias-map inventory 2026-07-10 05:28; first score 2026-07-10 05:42. Thus construction was registered before scoring. The identifier-only overlap audit materialized on 2026-07-11, after inference; the manuscript evidential hierarchy was reclassified on 2026-08-04, with Verified treated as sensitivity and Lite as primary, without changing any cell, seed, or outcome. This was a post-hoc adaptation-overlap correction, not a pre-outcome promotion. The PathSwap comparison is likewise a secondary post-preregistration follow-up. For the additional PathAlias intervals, a crossed bootstrap resamples both instances and the ten alias seeds (B=10,000, seed 42), yielding $[-27.1, -16.8]$pp on Lite and $[-20.4, -13.7]$pp on Verified. Every pre-registered per-benchmark estimand uses the full population, matching the PathSwap anchors. The implementation amendment records implementation fixes; the full-population amendment restores the full-population estimand.

Table 10: PathAlias per-cell results (paired drop vs. the same-batch Normal run; outcomes averaged over ten preselected seeds). The two 14B no-augmentation rows use the crossed instance-by-alias-seed intervals described above. Lite is the adaptation-disjoint primary population. Verified rows are sensitivities limited by adaptation overlap because 384/500 evaluation IDs occur in the adaptation population. The PathAlias/single-pair path-transplant harness Normal baselines (57.3 for Lite no-augmentation and 61.0 for Lite Codeaware) use the 30-line bf16 harness and differ by at most 1pp from the legacy seed-42 rows.

Cell	Benchmark	Normal	PathAlias	Alias−Normal (pp; 95% CI)	Normal−Alias (pp; per-mapping range)
14B no-aug (primary)	Lite	57.3	35.5	-21.8 $[-27.1, -16.8]$	19.7–24.7
14B no-aug (adaptation-overlap sensitivity)	Verified	49.3	32.3	-17.1 $[-20.4, -13.7]$	14.5–18.6
7B SWE-only unaugmented recipe (replication)	Lite	56.0	25.0	-31.0 $[-36.0, -26.0]$	30.0–32.3
7B SWE-only unaugmented recipe (adaptation-overlap sensitivity)	Verified	46.9	27.5	-19.4 $[-22.7, -16.2]$	17.2–22.3
14B Codeaware (recipe sensitivity)	Lite	61.0	56.1	-4.9 $[-8.3, -1.7]$	3.7–6.3
14B Codeaware (recipe/adaptation-overlap sensitivity)	Verified	49.2	48.7	-0.5 $[-2.2, +1.1]$	-0.1–1.1

Of the 300 Lite issues, 93 also occur in Verified; on the remaining 207, the same seed-mean paired estimator gives a -21.5pp drop, CI $[-27.5, -15.6]$.

Verified stratum	n	Drop (pp)	95% CI
Adaptation-disjoint	116	-14.2	$[-20.2, -8.8]$
In-adaptation	384	-17.9	$[-21.7, -14.3]$

The post-hoc adaptation-membership stratification uses the same seed-mean paired estimator as the primary analysis; the in-adaptation-minus-disjoint stratum-difference CI is $[-10.4, +3.3]$ and crosses zero. The Verified drop is therefore not attributable to adaptation overlap alone; this is a non-confirmatory sensitivity, not a new primary claim.

Repository-composition sensitivity. A post-hoc repository-stratified standardization supports only this bounded reading. For both PathAlias and the PathSwap reference, every adjusted and unadjusted in-adaptation-minus-disjoint 95% CI includes zero, while the adjusted point-estimate sign changes across the disjoint, common-support in-adaptation-mix, and pooled-common-support targets. Exact standardization of the disjoint stratum to the full in-adaptation repository mix is not estimable: $3/384$ target cases occur in repositories absent from the disjoint stratum, and the common-support results are projections, not substitutes. Sign instability reflects repository mix, not equivalence.

The pre-registration designated Verified as primary before the outcome-blind overlap audit; the audit changes its evidential role, not its cell, seeds, or outcomes. Per-seed Normal−Alias drops (pp), seeds 42–51, are all reported. Lite: $23.3, 21.7, 24.0, 21.0, 20.0, 21.0, 19.7, 24.7, 22.0, 21.0$. Verified sensitivity: $17.1, 17.6, 17.3, 18.6, 17.1, 14.5, 16.1, 16.9, 16.8, 18.5$. The full-population amendment's secondary diagnostic (gold-reachable-in-top-100 subset, deterministic zeros removed from both arms) gives larger drops, as expected: Verified n=333, -25.6pp CI $[-30.2, -21.0]$; Lite n=260, -25.2pp CI $[-30.9, -19.7]$.

Recipe interaction in the Alias–PathSwap loss difference. For recipe m, define

$$a_m = (R_m^N - \overline{R_m^A}) - (R_m^N - R_m^W) = R_m^W - \overline{R_m^A},$$

where R is fractional Recall@1, N denotes Normal, A denotes PathAlias averaged over the ten frozen mappings, and W denotes the single deterministic opaque PathSwap mapping. Positive a_m means that Alias causes the larger loss. The prespecified interaction is $a_{\text{noaug}} - a_{\text{Codeaware}}$; Lite is primary and Verified is secondary and adaptation-overlap-limited.

The shared-instance paired bootstrap averages the alias mappings first and resamples the same instances for both recipes. It gives no-augmentation asymmetries of $+5.17$pp (95% CI $[+0.67, +9.70]$) on Lite and $+2.86$pp ($[-0.03, +5.72]$) on Verified, and Codeaware asymmetries of -2.10pp ($[-4.57, +0.37]$) and -0.68pp ($[-1.99, +0.54]$), respectively. The resulting interactions are $+7.27$pp ($[+1.87, +12.73]$) and $+3.54$pp ($[+0.47, +6.58]$). Both interaction intervals exclude zero under this instance-resampling analysis, conditional on the ten recorded alias mappings.

A separate crossed instance-by-mapping analysis of the no-augmentation asymmetry gives $[+0.4, +10.0]$ on Lite and $[-0.2, +5.9]$ on Verified. These are within-recipe asymmetry intervals, not interaction intervals. The crossed analysis resolves the strict Alias–PathSwap ordering only for no-augmentation on Lite. The Codeaware intervals include zero; we make no equivalence, non-inferiority, or demonstrated-disappearance claim. The interaction concerns the tested recipes and interventions, not the original path's causal contribution or a deployment improvement.

Cross-vendor zero-shot floor boundary. A preregistered Llama-3.1-8B-Instruct zero-shot, code-visible replication on the same Verified population (n=500) reached only 15.40% Normal Recall@1, below the pre-registered 20% scorer-adequacy floor. Descriptively, applying the same ten PathAlias mappings and averaging over them reduced Recall@1 to 8.78% (Normal$-$Alias $+6.62$pp, paired 95% CI $[4.06, 9.26]$; all ten seed-wise drops were positive), and an independent implementation reproduced the means and CI sign. Under the preregistered rule this cannot support a cross-family generalization claim because the Normal score is below the pre-registered adequacy floor; we do not use this result to strengthen the primary finding.

Post-hoc leave-one-repository-out robustness. Excluding django leaves n=269 and increases the Verified Normal-minus-arm PathAlias/PathSwap drops from full-sample $+17.06$pp (95% CI $[+13.92, +20.23]$) and $+14.20$pp (95% CI $[+10.53, +17.87]$) to $+19.46$pp and $+18.40$pp, respectively. No single-repository omission reverses either aggregate point estimate (Table 11). This post-hoc sensitivity neither resolves the Verified PathAlias concentration-criterion failure nor establishes uniform repository-level effects or repository-population generality.

Preregistered concentration and lexical-coincidence boundaries. On Verified, django accounts for 0.386 of the summed absolute repository contributions, exceeding the pre-registered 0.35 concentration cap; on Lite, the small seaborn (n=4) and xarray (n=5) repositories have mean drops of -17.5pp and -6.0pp. We also attempted a stricter lexical-exclusion construction that excluded all evaluator-visible issue vocabulary. It failed its pre-registered feasibility criterion because Sphinx required 23.66% composed aliases, above the 20% cap; no scoring model was loaded and no outcome was produced. Thus that construction supplies neither positive nor NULL model evidence, and strict lexical-coincidence closure remains open.

Single-pair path transplant (preregistered, bounded cue-substitution audit). For each pair-eligible issue, the single-pair path-transplant audit exchanges the displayed paths of one gold–distractor pair while holding their code contents and the rest of the fixed candidate pool unchanged. Lite is the primary population (300 issues; structurally scored n=260/248/260 for the top-ranked distractor, same-directory sibling, and random distractor tiers). The secondary Verified cohort contains only the 116 adaptation-disjoint IDs, with structurally scored n=54/49/54. Let $G_i = 1$ denote that Normal ranks the selected gold file first. Capture is $P(\text{transplanted top-1} = d \mid G_i = 1)$, and the table reports $\sum_i G_i$ as its conditional denominator. The Normal reference is the full-tier rate at which d was already Normal top-1; it therefore uses a different denominator. Path share is the recorded aggregate share of the Normal winning margin on the observed $G_i = 1$ subset.

The paired path-exchange results are reported in main Table 1.

On Lite, no-augmentation capture is $77.9/80.2/63.4\%$ for the top-ranked distractor, same-directory sibling, and random distractor tiers, versus full-tier Normal references of $33.8/18.5/0.4\%$ and path-margin shares of $80.8/79.3/72.1\%$. Under this deliberately constructed path–content conflict,

Table 11: Post-hoc repository robustness. Positive values are Normal-minus-arm Recall@1 drops (pp). Repository rows give descriptive 95% paired-bootstrap CIs; the Lite/Verified counts are shown in the Lite panel. LORO rows are instance-weighted point estimates after one repository is omitted, with no LORO CIs. Repository CIs are unadjusted for multiplicity and can be unstable for small n.

		Lite	
Repository	**Lite/Verified n**	**PathAlias**	**PathSwap**
astropy	6/22	+45.00 [+11.67, +78.33]	+50.00 [+16.67, +83.33]
django	114/231	+20.44 [+11.84, +29.04]	+12.28 [+2.63, +21.93]
matplotlib	23/34	+23.48 [+7.39, +39.13]	+8.70 [−8.70, +26.09]
seaborn	4/2	−17.50 [−45.00, +5.00]	+25.00 [−50.00, +100.00]
flask	3/1	+53.33 [+50.00, +60.00]	+33.33 [+0.00, +100.00]
requests	6/8	+36.67 [+10.00, +66.67]	+33.33 [+0.00, +66.67]
xarray	5/22	−6.00 [−38.00, +16.00]	+0.00 [+0.00, +0.00]
pylint	6/10	+51.67 [+23.33, +80.00]	+66.67 [+33.33, +100.00]
pytest	17/19	+21.76 [+2.94, +42.35]	+29.41 [+0.00, +52.94]
scikit-learn	23/32	+9.57 [−4.35, +24.35]	+8.70 [+0.00, +21.74]
sphinx	16/44	+45.00 [+23.12, +66.25]	+25.00 [+0.00, +50.00]
sympy	77/75	+19.61 [+9.61, +29.87]	+15.58 [+5.19, +25.97]
LORO range	not applicable	+20.53–+22.85	+15.65–+19.35
Max shift	not applicable	sphinx −1.31 → +20.53	django +2.69 → +19.35

	Verified	
Repository	**PathAlias**	**PathSwap**
astropy	+6.21 [−6.52, +19.24]	+13.64 [−4.55, +31.82]
django	+14.26 [+9.86, +18.76]	+9.31 [+4.44, +14.18]
matplotlib	+20.34 [+6.52, +33.73]	+20.59 [+5.88, +35.29]
seaborn	+5.00 [+0.00, +10.00]	+25.00 [+0.00, +50.00]
flask	+60.00 [+60.00, +60.00]	+100.00 [+100.00, +100.00]
requests	+36.25 [+10.00, +66.25]	+25.00 [+0.00, +62.50]
xarray	+31.36 [+13.64, +50.00]	+25.00 [+0.00, +50.00]
pylint	+11.25 [−0.50, +28.50]	+15.00 [−10.00, +45.00]
pytest	+42.37 [+22.11, +63.16]	+47.37 [+26.32, +68.42]
scikit-learn	+9.69 [+3.75, +16.56]	+4.69 [+0.00, +12.50]
sphinx	+21.44 [+10.68, +33.33]	+8.71 [−1.89, +20.08]
sympy	+15.81 [+8.40, +23.68]	+19.56 [+10.44, +29.12]
LORO range	+16.06–+19.46	+12.89–+18.40
Max shift	django +2.40	django +4.20

exchanging a single pair's displayed paths while holding both files' code contents fixed can redirect the no-augmentation reranker in the Lite structural population. The bounded interpretation is cue-substitution/intervention sensitivity under the fixed scorer, prompt, pool, arm, population, and tier; the result does not establish normal-input path reliance or a causal channel share in congruent operation. Codeaware capture is descriptively $18.6/11.7/1.1\%$ on Lite; all Codeaware full-tier McNemar families are Holm-NULL, and no between-arm capture test was preregistered. Because capture and Normal reference use different denominators, the corrected paired full-set $T-B$ metadata treats the Lite Codeaware top-ranked-distractor/same-directory-sibling tiers and the Verified Codeaware same-directory-sibling tier as neutral. The preregistered γ correspondence term has mixed signs across tiers and arms; no additive decomposition follows.

POST-HOC incongruency validity diagnostic. This post-hoc score-output check was specified after the single-pair path-transplant results were seen. Measured against the corresponding recorded Normal score distribution, incongruent C is more Kolmogorov–Smirnov (KS) distant than congruent D in $12/12$ cells, and pooled incongruent $B{+}C$ is more KS-distant than pooled congruent $A{+}D$ in $11/12$. The diagnostic rules out only gross score-scale degeneration; it does not establish that the incongruent prompts are in-distribution and does not weaken the boundary that normal-operation path reliance remains unresolved.

Table 12: Single-pair path transplant. Except for n and G, entries are percentages. Here n is the structurally scored tier size and $G = \sum_i G_i$ is capture's conditional denominator. Each capture cell reports $P(\text{transplanted top-1} = d \mid G_i = 1)$ [95% CI], followed by G; each ref./path cell reports the full-tier Normal rate at which d was already Normal top-1 and the path share on $G_i = 1$. Lite is primary; adaptation-disjoint Verified is a bounded secondary population.

Tier	n	Arm	Capture [CI]; G	Ref./path
Lite (primary)				
Top-ranked distractor	260	No-aug	77.9 [71.5, 83.7]; 172	33.8/80.8
		Codeaware	18.6 [13.1, 24.6]; 183	29.6/15.3
Same-directory sibling	248	No-aug	80.2 [74.1, 86.4]; 162	18.5/79.3
		Codeaware	11.7 [7.0, 17.0]; 171	15.3/16.4
Random distractor	260	No-aug	63.4 [55.8, 70.3]; 172	0.4/72.1
		Codeaware	1.1 [0.0, 2.7]; 183	0.4/5.8
Verified (adaptation-disjoint secondary)				
Top-ranked distractor	54	No-aug	84.1 [72.7, 93.2]; 44	18.5/82.1
		Codeaware	4.9 [0.0, 12.2]; 41	24.1/21.8
Same-directory sibling	49	No-aug	79.5 [66.7, 92.3]; 39	16.3/83.1
		Codeaware	11.1 [2.8, 22.2]; 36	20.4/16.5
Random distractor	54	No-aug	77.3 [63.6, 88.6]; 44	0.0/75.8
		Codeaware	0.0 [0.0, 0.0]; 41	1.9/8.0

B.2 Structural-Masking Contrast

Table 13: **Pre-registered test-time MaskPath intervention contrasts on the fixed 300-issue Lite population; this is not the training-time PathMask of Table 56.** N=Normal (original path), M=MaskPath (registered structural masking of path text), X= the preselected cross-repository PathAlias map of Appendix B (the same 10 mapping seeds 42–51), and W= the within-repository-donor alias arm. δ_{NM}=$N - M$ uses an issue bootstrap; δ_{MA} is M minus the X seed mean and uses an issue-by-seed crossed bootstrap. Both use B=10,000 and NumPy seed `2026081901`. The reserved same-repository-donor comparisons N–W/W–X lack available token/truncation equivalence, so they are invalid and not estimable; they enter Holm with a 1.0 surrogate. Decisions were registered before scoring; positive support requires Holm p<0.05 and a positive estimate. Code-aware rows are secondary descriptive outside the family.

Cell	Contrast	Estimate (pp)	95% CI	Raw p	Holm p	Pre-registered decision
Pre-registered decision family						
14B no-aug (primary)	δ_{NM}	+21.7	[16.3, 27.0]	0.0002	0.0008	supported (positive)
14B no-aug (primary)	δ_{MA}	+0.2	[−3.6, 3.9]	0.92	1.00	not detected
Reserved same-repository donor arm	N–W	not estimable	not applicable	not applicable	1.0	not estimable (invalid construction)
Reserved same-repository donor arm	W–X	not estimable	not applicable	not applicable	1.0	not estimable (invalid construction)
Secondary descriptive contrasts (outside the decision family)						
14B Codeaware (secondary)	δ_{NM}	+5.7	[2.3, 9.0]	not tested	not tested	not tested
14B Codeaware (secondary)	δ_{MA}	−0.8	[−2.6, 1.0]	not tested	not tested	not tested

On 300 Lite issues, and within the same Check 1 stage-local estimand (§3), the pre-registered test-time MaskPath (structural masking of the displayed path text; Table 13), not the training-time PathMask of Table 56, lowers Recall@1 by 21.7pp (95% CI [16.3, 27.0]) in the primary 14B no-augmentation cell; this is a distinct intervention and number from the PathAlias −21.8pp cell in Figure 1. The 0.2pp MaskPath-minus-PathAlias difference for the preselected cross-repository PathAlias map is not detected ([−3.6, 3.9]), and non-detection is not equivalence; the two reserved same-repository-donor arms are unestimable.

B.3 ABSOLUTE SCORES AND THE OPAQUE REFERENCE

Table 14: Absolute Recall@1 (%) for the same-harness, no-augmentation Qwen2.5-14B code-in-prompt reranker. PathAlias averages ten mappings. Lite is the primary issue-held-out population; Verified is an adaptation-overlap-limited sensitivity analysis. Figure 3 displays these point estimates separately from the paired-change intervals and mapping-level results.

Population	Normal	PathAlias	PathSwap
Lite	57.3	35.5	40.7
Verified	49.3	32.3	35.1

For the matched opaque-reference rerun, Normal minus PathSwap is 16.7pp on Lite (instance-paired 95% CI [11.3,22.3]) and 14.2pp on Verified ([10.5,17.9]). These original intervals come from the `lite.sha_ci` and `verified.sha_ci` fields of the retained matched-reference result; they resample instances for a single deterministic mapping. The main-figure Alias intervals also resample alias mappings. The paired headline, crossed Alias and matched reanalysis summaries retain their separate identities; the direct crossed ordering remains resolved only on Lite.

B.4 OPAQUE-REFERENCE CONSTRUCTION

To test sensitivity to readable path content, we evaluate with deterministic opaque identifiers. Each path component is replaced with a consistent identifier per repository, preserving stable component identity, tree depth, separators, and `.py` extension while removing readable lexical content, as in the first PathSwap definition.

Table 15: Opaque-identifier path control (GREPO, 1704 examples). "Opaque identifiers" replaces each path component with a deterministic opaque identifier; "Shuffled" randomly permutes file names across the repository.

Condition	**Recall@1**	**R@5**	**R@10**	**Cond. Acc@1 (%)**
Original paths	24.51	44.18	52.12	46.51
Opaque identifiers	1.19	3.94	7.07	3.82
Shuffled paths	1.10	3.43	5.77	3.56

The opaque-identifier condition reduces Recall@1 from 24.5% to $\sim$1.2% for this tested path-only scorer. That is a large sensitivity to removing readable lexical path content, measured under the structure-preserving PathSwap reference condition; it is not a PathAlias natural-form test. The condition retains stable identifiers and hierarchy, so it is not complete path-channel neutralization. Opaque-identifier and shuffled paths both score near 1% in these cells; that similarity does not establish absence of structural use. The residual is of the same order as the inverse average pool size ($\sim$163 candidates), but no calibrated random baseline was run.

B.5 HISTORICAL-RENAME CALIBRATION

Construction. From GREPO we identify each repository's rename/move history with Git's rename tracking and chain-walk it to map each current gold path to its earliest historical path. The organic-drift subset is the n=862 test instances whose gold file was genuinely renamed and remains in the BM25 top-100 candidate pool. For the drift arm we replace the gold's displayed path with its historical path in both the candidate list and the ground-truth key; a lookup map redirects code reads back to the file's real on-disk content, so *only* the displayed path token changes. Both arms use the 14B checkpoints of Table 28 (path-augmented "Codeaware", aug=0.5, and the matched no-augmentation LoRA), bf16 inference, code-in-prompt, BM25 top-100, seed 42.

Historical-path effects depend on the cutoff. Unlike Recall@1, top-k recall for k>1 *increases* under historical-path drift for both models: R@5 rises from $44.22 \rightarrow 54.91$ (Codeaware) and $43.71 \rightarrow 50.54$ (no-aug); R@10 rises from $50.22 \rightarrow 63.14$ and $49.90 \rightarrow 59.81$. The paired difference-in-differences is significant at these cutoffs ($+3.85$pp CI $[+2.02, +5.78]$ at R@5; $+3.01$pp

CI $[+1.59, +4.59]$ at R@10), but these directions alone establish neither lexical ease as the mechanism nor confounding of the top-k estimates. We base the robustness conclusion in §3 on Recall@1 and treat higher-cutoff results as complementary outcomes, not general robustness.

Power. The subset is capped at the number of genuinely-renamed golds (n=862); it cannot be enlarged without changing the perturbation. At this n the Recall@1 difference-in-differences (+1.25pp) has a 95% CI of $[-0.37, +2.87]$ and is of the same magnitude as batch-size-induced floating-point nondeterminism, leaving power insufficient to establish a Codeaware robustness advantage.

Table 16: **Organic rename drift (GREPO, n=862 genuinely-renamed golds) calibrates the path intervention on repository rename history.** Each gold's displayed current path is replaced by its historical repository path; content, candidate pool, and all other paths are held fixed. **Recall@1** (%) with paired bootstrap 95% CIs (B=10,000). Historical-path drift significantly lowers Recall@1 for both the 14B path-augmented (Codeaware) and no-augmentation LoRAs, providing an ecological calibration alongside the opaque-identifier condition. The Codeaware−baseline difference-in-differences is directionally favorable but *not* significant, so we make no robustness claim from this test. Recall@5/Recall@10 increase under drift and are reported as complementary cutoff-dependent outcomes in Appendix B.5.

Model	Current path	Historical path	Paired drop (95% CI)
No-augmentation	22.13	18.66	-3.47 $[-5.12, -1.82]$
Codeaware (path-aug)	23.16	20.94	-2.22 $[-3.64, -0.84]$
Difference-in-differences (magnitude: |no-aug drop| − |Codeaware drop|)			$+1.25$ $[-0.37, +2.87]$ *n.s.*

C File-to-Function Transfer

For a file-win indicator F_{is} and function-hit indicator $H_{is}^{(g)}$, let scorer g rank function candidates from system s's top-10 predicted files. A function hit matches the gold file path and bare function name. A file win that also yields a function hit is called *certified*; on a declared label-defined set $\mathcal{E}$, its rate is

$$\widehat{c}_s(g; \mathcal{E}) = \frac{\sum_{i \in \mathcal{E}} F_{is} H_{is}^{(g)}}{\sum_{i \in \mathcal{E}} F_{is}}. \tag{2}$$

We specify each function scorer and record set before comparing certification. Historical configuration sensitivities, missing-label rules, replication and PathSwap strata retain separate populations.

C.1 Transfer Populations and Scoring Instruments

Table 17: Function outcomes within file-level success. Panel A contains 4,452 system–instance records from 1,113 common label-defined GREPO issues and four Qwen rerankers; $585/2{,}356 = 24.8\%$ $[22.5, 27.2]$ of file wins also yield an own-scorer function hit. Panel B compares Normal-path file wins that remain hits under PathSwap with those that become misses. The difference is 9.90pp $[5.98, 13.87]$, clustered over 792 issues. Function candidates come from the top-10 files.

A. Joint file/function outcomes

	Function correct	Function wrong
File correct	585	1,771
File wrong	79	2,017

B. Own-scorer transfer among Normal file wins

Group	File-win records	Function Hit@1 (%)
PathSwap-stable	1,530	28.30
PathSwap-fragile	826	18.40

Population accounting. For Table 17, each reranker has 1,704 file-localization records and 1,574 paired function-localization records. The four joins contain 6,296 system–issue pairs. The same 1,113 issue keys have defined function judgments under all four rerankers, giving 4,452 pairs for the confusion table. Among those pairs, 2,356 are file wins and 585 also achieve function Hit@1; the file-win records span 792 distinct issue clusters. This accounting distinguishes the paired-stage population, the label-defined population, and the certification denominator; it does not assess the validity of the function labels themselves.

Complete file-to-function procedures. A reranker that never reads code reaches 29.70% file Recall@1. No statistically reliable difference is detected between it and the 14B code-aware rerankers (28.89% and 29.22%; paired differences $+0.81$pp, CI $[-0.66, +2.30]$, and $+0.48$pp, CI $[-0.99, +1.95]$). Function Hit@1 on all 1,113 examples (not certification among file wins) is 25.1% for path-only, 18.7% for 14B Codeaware, and 18.4% for 14B no-augmentation (paired path-only gaps $+6.4$/$+6.7$pp; both intervals exclude zero). Because scorer and system vary together, these values and intervals describe complete file-to-function procedures rather than differences between file rerankers under a shared function scorer; the direct shared-scorer comparisons address the latter question.

The three full-procedure score rows appear in main Table 5A; this section retains the paired comparisons and population definitions.

The pre-registered Lite replication contains 509 file-level wins spanning 153 issue clusters: 14B Codeaware certifies 32.33% of 133 wins, 14B no-augmentation 35.34% of 133, 7B Codeaware 11.30% of 115, and 7B no-augmentation 15.62% of 128, pooling to 123/509 (24.17%, reported as 24.2%). The independent seed-42 audit flags 3/100 sampled labels (3.0%); excluding all eight flags found in the full 279-label population changes pooled transfer from 24.17% to 24.64%.

Shared-scorer instantiation. Both fixed shared scorers instantiate the base `Qwen2.5-7B-Instruct` checkpoint with no adapter, in bf16 with a 768-token context, file summaries, and the top 10 predicted files. The fixed-neutral scorer retains real function names; the name-blind lexical probe uses the same weights and prompt but deterministically replaces the candidate names within each instance by `fn_0000`, `fn_0001`, and so on, retaining real names only for evaluation. The verbatim prompt is "`Given the bug report, is this function likely to need modification? Bug Report: {issue_text} File: {file_path} File Content: {file_summary} Function: {function_name} Answer:`", with blank lines between the question, report, and file block; the score is the final-token `Yes` logit minus the `No` logit. For path-only, fixed-neutral uses the earlier 4-bit/top-200 file-localization snapshot (603 label-available wins), while name-blind uses the current bf16/top-100 snapshot (598); the two 14B rows use their recorded current snapshots under each shared scorer (649 Codeaware and 652 no-augmentation label-available wins).

Under the pre-registered top-10 bf16 function-scoring protocol, only 20–29% of file-top-1 hits yield a function hit (the within-system own-scorer certification rate is 24.8%, issue-clustered 95% CI $[22.5, 27.2]$). Marginal Spearman correlation across repositories between file- and function-level Recall@1 (not the among-file-win certification rate) is $\rho = 0.31$–0.39 ($n = 61$ repos) but is filter-sensitive (pooling the four systems raises it to 0.57; restricting to repos with ≥ 5 examples lowers it to 0.29). Separately, among repos with ≥ 10 examples, no statistically significant correlation between transfer among file-top-1 hits and file Recall@1 is detected in any tested system ($\rho \in [-0.16, +0.27]$, all n.s.). Some repositories reach 100% file Recall@1 but 0% function Recall@1.

Filename-shuffling sensitivity analysis. All seven system variants retain 23–47% of Recall@1 under filename shuffling (mean ratio 0.29 ± 0.08); code-aware variants are not consistently less sensitive than path-only. The perturbation therefore measures within-system filename sensitivity but does not distinguish system types. For the path-only baseline, repo-level bootstrap intervals for Normal [21.2, 33.5] and shuffled [5.3, 9.9] do not overlap; this does not support a comparative robustness ordering across systems.

Top-ranked-file containment and source of certification labels. A complementary *containment* measurement asks only whether the top-ranked gold file *contains* a gold changed function per the AST function index, without consulting function-level predictions. Containment is 68–78% across the four rerankers (per-repository correlation with file-level Recall@1 near zero, $r \in [-0.02, +0.29]$). This

top-ranked-file diagnostic is not an upper bound on the pre-registered top-10 bf16 function-scoring protocol, which can select a function from another retrieved file. It does not by itself exclude missing labels as an explanation for transfer failures. An earlier measurement of this quantity on a trained path-only reranker's predictions gave $476/753 = 63.2\%$ ($r = 0.091$, $n = 46$ repositories), and an earlier function-ranking procedure (top-50 candidate files) measured 43.9% behavioral transfer with 29% of ground-truth functions appearing in the top-10 ranked functions. We recovered sibling snapshots of both instruments, not the exact snapshot behind the printed 43.9% cells, and reproduce the earlier transfer number at 43.6–43.8%; the per-repository correlation is snapshot-fragile (0.07–0.27). All instrument versions agree on the direction: file-level wins mostly fail to certify function-level localization, and the pre-registered top-10 bf16 function-scoring protocol reports the strictest, fully specified version of the measurement.

Certification, like any measurement, is instrument-referenced. Table 18 records configuration sensitivity on the same trained path-only reranker across two file-localization snapshot vintages. Rows 1–2 share the earlier snapshot but jointly change scorer, top-N, and precision; the 7.3% to 43.6% shift therefore cannot be attributed to scorer choice alone. Row 3 further changes snapshot while returning to top-10/bf16, so its 37.3% level is also not a single-factor contrast. The table motivates declaring scorer, pool depth, precision, and snapshot together; it does not identify a dominant component. The prescribed cross-system claim fixes one predeclared shared scorer and reports an ordering and gap under that declared instrument, never a canonical point value; name-blind scoring is reported separately as a lexical probe.

Table 18: **Archived certification-configuration sensitivity** on the same trained path-only reranker. Here n counts label-available GREPO file-win records; Cert. is their function Hit@1 percentage under the listed configuration. Rows 1–2 share a file-localization snapshot but jointly vary scorer, top-N, and precision; row 3 also changes the snapshot. These rows do not isolate a scorer effect or decompose a dominant factor. The pre-registered protocol fixes the neutral operating point.

Function-scoring configuration	**file-localization snapshot**	n	**Cert.**
Neutral base-model scorer, top-10, bf16 (pre-registered protocol)	earlier snapshot (4-bit, top-200)	603	7.3%
Trained LoRA scorer, top-50, 4-bit (earlier instrument configuration)	same earlier snapshot	603	43.6%
Trained LoRA scorer, top-10, bf16 (matched)	current (bf16, top-100)	598	37.3%

C.2 Shared-Scorer Comparisons

Reported power summaries. The earlier report lists minimum detectable differences of 2.19, 2.63, 2.87 and 2.74 percentage points for fixed-neutral/Codeaware, fixed-neutral/no-augmentation, name-blind/Codeaware and name-blind/no-augmentation, respectively. The calculation metadata and discordant-pair counts for these values are not available. We retain the reported values here but use the paired intervals in Table 5, rather than these power estimates, to characterize the observed contrasts.

Table 19 reports each system's descriptive certification levels on its own label-available wins. The analyses below distinguish this scorer sensitivity from the paired common-win effects reported in main-text Table 5.

Name-blind certification (instrument lexical-surface probe). To test whether certification merely rewards a scorer's lexical surface, we recompute it with every candidate function's identifier replaced by an opaque token in the scoring prompt (real names kept only for evaluation). Masking lowers certification by about 2.5pp for both code-aware systems (14B Codeaware $8.32\%\rightarrow5.86\%$, paired-bootstrap CI $[-4.93, -0.15]$pp, exact McNemar p=0.060; 14B no-aug $7.82\%\rightarrow5.37\%$, CI $[-4.75, -0.15]$pp, p=0.048), providing borderline evidence for a name-lexical share of the instrument's signal—but it does *not* open a gap in their favor: under masking the code-aware systems certify 5.4–5.9%, and no statistically reliable advantage is detected over the code-blind path-only reranker's 6.0%. Across the three scorer conventions we tried—the system's own scorer, a fixed neutral scorer,

Table 19: **Scorer-referenced certification levels on each system's label-available file wins.** These descriptive GREPO levels are function Hit@1 percentages under the column's scorer on each system's label-available file-win records; denominators count positional records, not distinct issues. They are not paired cross-system effects. Direct comparisons use the common-win records in Table 5. Displayed denominators are path-only 598/603/598 (own/fixed-neutral/name-blind), 14B Codeaware 649/649/649, and 14B no-augmentation 652/652/652. The path-only own/name-blind cells use the current bf16/top-100 snapshot, whereas the fixed-neutral cell uses the earlier 4-bit/top-200 snapshot; cells are not a common-denominator estimate of operative all-win certification. All evaluated conventions are shown; the following analyses give contrast intervals and all-win levels.

System	Own scorer (%)	Fixed neutral (%)	Name-blind (%)
Path-only (code-blind)	37.3	7.30	6.02
14B Codeaware	28.5	8.32	5.86
14B no-augmentation	27.9	7.82	5.37

and this name-blind scorer—no shared convention gives a code-aware system a statistically reliable certification advantage over the code-blind one (under the fixed neutral scorer the code-aware point estimates are nominally higher, $8.3\%/7.8\%$ vs. 7.3%, with overlapping CIs). Under the name-blind scorer the two code-aware systems certify at 5.4–5.9% and the path-only baseline at 6.0%, within 0.7pp of one another; this panel bounds the name-lexical reliance of the instrument and does not establish cross-system equivalence. On the matched current path-only snapshot, the same masking probe changes certification from 7.53% to 6.02% (-1.51pp, paired-bootstrap CI $[-4.01, +1.00]$pp; exact McNemar p=0.289). This is also no detected difference, not evidence of equivalence.

Both-scorer certification (pre-specified labeling-sensitivity analysis). On the operative positional win sets (813 14B Codeaware; 834 14B no-augmentation), each system's own scorer certifies $22.8\%/21.8\%$ of wins, whereas only $1.72\%/1.68\%$ are certified under both fixed shared scorers—about a $13\times$ ratio. Both levels require two inseparable denominator caveats. First, $164/813$ and $182/834$ wins lack function-level gold labels and are cast uncertified by rule; conditional on label availability, the corresponding levels are about 28.5% own-scorer and 2.2% under both shared scorers, so the all-win levels are deflated by construction while the ratio survives. Second, the pre-registered analysis specification prose's 730/752 win counts exactly match a last-row-per-issue reduction, which conflicts with the pre-registered positional definition; therefore 813/834 remain operative without rewriting the pre-registered specification. Chance-corrected agreement is low (shared-scorer Krippendorff $\alpha = 0.263/0.289$, pooled 0.276; adding the own scorer gives pooled $\alpha = 0.101$), because scorer-dependent flips outnumber shared positive agreements. This is a labeling sensitivity result only. Shared-scorer labels exist on GREPO while resolution outcomes exist on SWE-bench Lite under own scorers, so the both-scorer-certification-to-resolution link cannot be computed because the scorer labels and repair outcomes come from disjoint populations.

Direct paired shared-scorer contrasts. For these four contrasts, paired percentile bootstraps ($B = 10{,}000$, seed 42) resample the common repository–issue composite keys, preserving both binary function outcomes in each pair. Equal local issue IDs in different repositories remain distinct records.

Restricting to instances where both systems achieve a file-level win removes the case-mix confound. Under both shared scorers, the code-aware-minus-path-only difference is nominally negative in all four comparisons (-0.2 to -1.0pp); every paired-bootstrap CI crosses zero and exact McNemar $p \geq 0.38$. For reference, the pooled four-reranker within-system own-scorer certification rate is 24.8%. The matched label-available own-scorer cells for the two 14B systems are 28.5%/27.9%, whereas their operative all-win own-scorer rates are 22.8%/21.8%.

C.3 Transfer on Path-Fragile and Stable Wins

Path-fragile vs. path-stable function transfer (pre-registered). We pre-registered a stable-vs-fragile contrast to ask whether file top-1 hits that flip under PathSwap also show lower function-level transfer. *Eligible* instances are normal-path file top-1 hits with a defined function-transfer label; a hit is *fragile* if it flips to a miss under PathSwap and *stable* otherwise; the metric is file-qualified

function-level Hit@1. We evaluate five rerankers spanning two model families (14B/7B Qwen $\times$ Codeaware/no-augmentation, plus a Llama-3.1-8B Codeaware replication) on the GREPO function subset and report the difference in function-level Hit@1 between the two groups (the two groups are disjoint subsets of the eligible pool, so all CIs use a two-group, not paired, bootstrap; B=10,000). Per-system CIs resample eligible hits directly; the *pooled* row aggregates *system–instance pairs* across the four rerankers (the same issue recurs once per system), so its CI uses an *issue-clustered* bootstrap—resampling issue clusters and re-partitioning stable/fragile each iteration. Table 20 reports the result. Pooled across the four Qwen rerankers (the Llama row is per-system only; 826 fragile and 1,530 stable system–instance pairs over 792 issue clusters), the gap is Δ=9.9pp (issue-clustered 95% CI $[6.0, 13.9]$; the naive unclustered CI is $[6.4, 13.3]$); it is individually significant for three of the four rerankers and within noise for 7B Codeaware. Path-fragile file hits show weaker function-level transfer. This is a stratified association, not mechanistic evidence: the groups are not assigned by intervention, and the analysis does not establish a causal direction or rule out shared causes.

Table 20: **Path-fragile file hits show weaker function-level transfer.** On the GREPO function subset, eligible instances are normal-path file top-1 hits with defined function-transfer labels. Entries report function Hit@1 for path-*stable* hits (remaining file hits under PathSwap) versus path-*fragile* hits (becoming file misses). Δ is stable$-$fragile with a 95% two-group bootstrap CI (resample + re-partition; not paired; B=10,000). Per-system rows count instances; the pooled row† counts *system–instance pairs* across the four Qwen rerankers only (excluding Llama) and uses an *issue-clustered* bootstrap (792 issue clusters). *CI excludes 0.

Reranker	**stable** n	**stable Hit@1**	**fragile** n	**fragile Hit@1**	**Δ (95% CI)**
14B Codeaware	526	30.61%	123	19.51%	$+11.10$ $[+2.94, +18.89]^\star$
14B no-augmentation	413	31.96%	239	20.92%	$+11.04$ $[+4.26, +17.94]^\star$
7B Codeaware	342	20.76%	178	19.66%	$+1.10$ $[-6.21, +8.25]$
7B no-augmentation	249	27.71%	286	15.03%	$+12.68$ $[+5.80, +19.62]^\star$
Llama-3.1-8B Codeaware	477	37.53%	133	20.30%	$+17.23$ $[+9.08, +25.09]^\star$
Pooled†	1,530	28.30%	826	18.40%	$\mathbf{+9.90}$ $[\mathbf{+5.98}, \mathbf{+13.87}]^\star$

D Executed-Repair Outcomes

D.1 Inputs, units and fixed source exclusions

The repair populations use SWE-bench Lite own-scorer labels. File-given repair supplies a file and allows edit-site choice; function-conditioned repair supplies the localized function. The fixed, outcome-independent rule excludes exactly 24/208 pre-existing locator-source mismatches, all uncertified. This leaves 184 attempts over 105 issues and 126 non-empty patches. The 44 and 77 within-group issue counts overlap; they must not be added to obtain the number of issues. These exclusions change neither certification labels nor outcomes.

Table 21: Source-corrected file-given repair: complete uncertainty summaries. Primary/secondary are the original arm roles. The clustered intervals, not selection of a significant permutation result, determine the reported NULL.

Set	Cert. resolved	Uncert. resolved	Δ [95% CI]	perm. p
Non-empty (primary)	18/50	15/76	16.3 $[-4.1, 36.7]$	.064
All attempts (secondary)	18/67	15/117	14.0 $[-1.2, 29.6]$	.028

D.2 Original-attempt routing and residual diagnostics

Weights of the non-additive contrasts. Let C be own-scorer certification, Z indicate that the chosen edit function is gold, and Y indicate executed resolution. All probabilities below are empirical

proportions on the original 208 attempts, before source exclusion.

$$\text{COMP} = \sum_{z\in\{0,1\}} [\widehat{P}(Z = z \mid C = 1) - \widehat{P}(Z = z \mid C = 0)]\widehat{P}(Y = 1 \mid Z = z), \tag{3}$$

$$\text{RESID} = \sum_{z\in\{0,1\}} \widehat{P}(Z = z)[\widehat{P}(Y = 1 \mid C = 1, Z = z) - \widehat{P}(Y = 1 \mid C = 0, Z = z)]. \tag{4}$$

COMP weights edit-site composition differences by pooled stratum success rates; RESID weights within-stratum differences by pooled stratum shares. Multiply by 100 for percentage points. Their different pooled standardizations do not partition the total gap; COMP/total is not an identified explained fraction. Z is a realized, post-localization selection, not a randomized mediator. These formulas document the existing computation; frozen specifications retain their historical bytes and do not override the current non-additive interpretation.

Pre-specified de-conditioning decomposition. On all 208 free-fixer attempts, the certified–uncertified gap is $+15.5$pp (issue-clustered 95% CI $[0.8, 31.1]$). A pre-specified standardization over the fixer's realized edit-site correctness reports two prespecified contrasts on the same attempts: mechanical wrong-function routing, COMP $= +8.6$pp (CI $[3.6, 14.9]$), and a within-stratum residual, RESID $= +5.9$pp (CI $[-5.8, 17.6]$); they are not additive components of the $+15.5$pp aggregate, and among these prespecified routing/residual contrasts only COMP excludes zero. The descriptive ratio COMP/G, where G is the pre-exclusion total gap, has CI $[0.20, 2.53]$ around a point estimate of 0.55. Because the reported contrasts do not partition the total gap, this ratio is not an identified fraction explained by routing; its interval extends above one. The denominator of this ratio is sensitive to the provenance exclusion: after the fixed 24-row runner-source exclusion, the aggregate $+14.0$pp interval $[-1.2, 29.6]$ crosses zero. Within the correct-function stratum, the certified–uncertified residual is $+11.2$pp (CI $[-9.2, 31.8]$), a NULL that establishes neither a remaining effect nor its absence. The descriptive de-conditioning ratios $0.474/0.524$ are point summaries only. On conditioned rows, certification agrees with the conditioning function being gold at 0.996, so that arm cannot form a within-correct-function residual. These decompositions do not identify causal mediation.

D.3 Function-conditioned outcomes and execution coverage

Table 22: **Function-conditioned repair checks and runner-source-corrected free-fixer sensitivity.** Rows pool the Codeaware and no-augmentation 14B localization systems on SWE-bench Lite, using each localization system's own-scorer certification. Cells give resolved/non-empty patch counts within each certification group, not all-attempt resolution rates. These are associations, not causal effects of certification. The last row removes function-name conditioning and excludes 24/208 runner-file-source mismatches. Δ (pp) is certified$-$uncertified; CIs resample issue clusters and p-values permute issue-level labels within system. The two Qwen3 fixer rows cover all twelve Lite repositories; the Llama-3.1-8B row covers the seven whose images were available at its run time.

Fixer	**Cert.**	**Uncert.**	**Δ (95% CI)**	**Perm. p**
Qwen3-14B (Qwen3)	23/75 (30.7%)	1/85 (1.2%)	$+29.5$ $[+16.4, +43.4]$	$< 10^{-4}$
Qwen3-8B (Qwen3)	20/65 (30.8%)	0/58 (0.0%)	$+30.8$ $[+16.1, +46.0]$	$< 10^{-4}$
Llama-3.1-8B (Llama)	17/67 (25.4%)	1/125 (0.8%)	$+24.6$ $[+11.4, +38.6]$	$< 10^{-4}$
Free-fixer sensitivity (Qwen3-14B)	18/50 (36.0%)	15/76 (19.7%)	$+16.3$ $[-4.1, +36.7]$	0.064

Real test-execution resolution (details). Table 22 gives the breakdown of the downstream-cost experiment (§6). For each 14B reranker we generate a real edit for the system's top-1 (file, function) with a separately run repair model, and run the official SWE-bench harness; *certified* = the reranker's top-1 function is a gold changed function, *uncertified* = it is not. We run three *conditioned* fixers—the model rewrites the localized function—spanning two model families: Qwen3-14B and Qwen3-8B (different capacities), plus Llama-3.1-8B (cross-family), each a pre-registered amendment with the same greedy protocol. The Qwen3-14B arm covers all twelve Lite repositories (n=264 file-wins: of the 53 initially-errored evaluation rows, 51 were recovered—36 of them file-wins that enter the contrast—and 2 remain errors, both on one `requests` instance whose test suite hangs offline); the

Qwen3-8B fixer covers all twelve repositories; the Llama-3.1-8B fixer covers the seven whose images were available at its run time. Restricting to non-empty patches (resolution conditional on producing an edit), certified file-wins resolve $30.7\%/30.8\%/25.4\%$ across the three fixers versus $1.2\%/0\%/0.8\%$ uncertified (all issue-clustered CIs exclude zero and all issue-level permutation $p<10^{-4}$; the Qwen3-14B difficulty-adjusted logistic—Firth-penalized, since the near-empty uncertified cell quasi-separates an unpenalized fit—gives an average marginal effect of $+22.6$pp, penalized odds ratio 34, 95% CI $[6.8, 168]$, $p=1.7\times10^{-5}$).

Under the gold-file restriction specified before outcomes were examined, Qwen3-14B has $n=149$ and is 23/75 versus 1/74 ($+29.3$pp, CI $[15.9, 43.4]$); the same restriction gives $+30.8$pp (CI $[16.4, 46.0]$) for Qwen3-8B and $+24.4$pp (CI $[11.4, 38.5]$) for Llama-3.1-8B. All three are conditioned on the localization by design, so the near-zero uncertified cell is partly mechanical; the *free-fixer* arm (last block) removes explicit function-name conditioning. A runner-source audit found that its runner used the file containing the released function-localization top function rather than the file-localization top file on 24/208 eligible attempts. After fixed exclusion, its non-empty contrast is $+16.3$pp (36.0% vs. 19.7%; CI $[-4.1, 36.7]$, $p=0.064$); its all-attempt contrast is $+14.0$pp (CI $[-1.2, 29.6]$, issue-level permutation $p=0.028$). Both intervals cross zero, so this runner-source-corrected arm is a NULL sensitivity: it establishes neither a free-fixer association nor its absence and is not evidence for a nonmechanical effect. The published-system checks below relate released localization correctness to harness resolution without conditioning fixers on the audited function; all mixed results are reported.

Pre-specified supplementary checks. A within-issue analysis retained only issues for which the two rerankers received opposite certification labels. In the free-fixer all-attempt population this leaves 17 issues, but no resolution-discordant pair: two resolve under both systems and 15 under neither (paired difference 0.0pp, exact $p=1.0$). The pre-specified minimum of five outcome-discordant pairs is not met, so this check is sparse/NULL and does not strengthen the marginal association. Separately, extending the conditioned-arm Firth model with an indicator for whether the issue text exactly names any gold function leaves certification positive (OR 42.0, 95% CI $[8.0, 219.4]$, $p=9.4\times10^{-6}$, AME $+24.9$pp); a conservative specification that excludes the potentially post-localization number-of-predicted-functions covariate is similar (OR 41.9, CI $[7.8, 224.3]$, $p=1.3\times10^{-5}$). These wide-interval models control one measured lexical-mention confound but do not make certification causal. Both analyses were independently recomputed from the recorded rows.

D.4 Published-system outcome checks

Published-system checks (no localization conditioning). Three released-system analyses relate a released system's own localization correctness to its official test-execution resolution (logistic: resolved $\sim$ file-correct + function-correct; the function-correctness coefficient is the certification analog, free of our scorer and of any repair-conditioning). We report all three. The differing point estimates do not establish how localization strength changes the association: these checks differ in benchmark or system version. Agentless-1.5 has few discordant (file-right, function-wrong) cases on Verified; the v0.1 check also uses a different repair stage and derives its function labels from submitted patches rather than a released localization stage.

Table 23: **Published-system checks.** Function-correctness odds ratio for real resolution, controlling for file-correctness, on released materials. All checks are reported, favorable or not.

System / benchmark	n	func OR	p	reading
Agentless-1.5 / Lite	272	2.25	0.009	significant
Agentless-1.5 / Verified	496	1.40	0.096	directionally consistent, n.s.
Agentless v0.1 (GPT-4o) / Lite	265	0.86	0.70	null (weak repair stage; patch-derived labels)

The positive Lite association and two non-significant cells are protocol-specific: benchmark, system-version and label differences prevent interpreting them as a localization-strength trend.

The within-system own-scorer certification association remains observational; this does not establish common-target alignment, candidate-universe validity, or locator invariance under alternative certification specifications. The published-system checks above provide mixed, protocol-specific evidence relating released localization correctness to harness resolution.

D.5 PRE-EXECUTION EDIT-LOCALIZATION PROXY

Edit-localization proxy (pre-execution). We hand each system's top-ranked (file, function) and the function body to a separately run repair model (Qwen3-14B, single greedy generation) and ask it to name the function it would edit. Among file-level wins, the repair model localizes the correct function on 82.5% of *certified* wins but only 44.4% of *uncertified* ones ($\Delta = 38.1$pp, 95% CI $[26.0, 49.4]$; individually significant for both systems tested). The gap is inconsistent with pure echo alone: the model overrides the suggested function on 86% of uncertified wins, whereas a pure-echo baseline scores 4%. Nor is the gap fully explained by the listed difficulty proxies: adjusting for candidate-function count, gold-function count, report length, function-body length, and gold-name rarity leaves $+35.2$pp as a logistic average marginal effect (certified odds ratio 5.7, $p < 10^{-5}$), and $+34.8$pp, CI $[20.7, 47.5]$, under coarsened exact matching; the unadjusted gap is $+38.1$pp. This is association rather than cause, and edit-localization is an easier proxy than producing a passing patch; it is not itself executed repair.

E TRAINING-RECIPE AND CODE-PREVIEW CONTROLS

Each block retains its checkpoint family, data and inferential role. The matched 14B Normal primary contrast, independent 7B factorial grid and legacy operating-point table are not a single experiment.

E.1 MATCHED 14B CODE-PREVIEW COMPARISON (PRE-REGISTERED)

We ask whether a code-preview recipe improves file localization when context length and the evaluation pool are matched. This compares two trained LoRAs, rather than removing the preview from one fixed checkpoint.

The pre-registered comparison uses separately trained Qwen2.5-14B seed-42, aug$= 0$ code=YES and code=NO LoRAs under the same BF16, 768-token, top-100 fixed-pool evaluation; the matched training configurations differ scientifically only in whether the short source preview is displayed. Lite is adaptation-disjoint ($0/300$ overlap) and has one gold per issue, so fractional Recall@1 equals binary Hit@1 here.

Both arms score only the selected checkpoint in each arm: each selected checkpoint is chosen by the mean of the most recent 50 training-loss values, with no test or evaluation metric entering selection; the final and other unselected checkpoints were not scored.

Table 24: Matched 14B code-preview contrasts on the fixed Lite pool. The Normal contrast is primary; the PathSwap contrast is secondary.

Fixed Lite pool	code=YES	code=NO	YES−NO
Normal (primary)	171/300 (57.00%)	167/300 (55.67%)	+1.33pp
PathSwap (secondary)	119/300 (39.67%)	1/300 (0.33%)	+39.33pp

The primary Normal contrast is $+1.33$pp, with paired percentile bootstrap 95% CI $[-3.00, +5.67]$pp ($B = 10{,}000$, NumPy seed 42) and exact two-sided McNemar $b = 23$, $c = 19$, $p = 0.644$. Because the interval includes zero, the pre-registered decision is NULL: it establishes neither a positive nor a negative Normal effect, and it is not an equivalence result. The pre-registered secondary PathSwap contrast is $+39.33$pp; the perturbation-loss difference, $[(\mathrm{PathSwap} - \mathrm{Normal})_{\mathrm{YES}} - (\mathrm{PathSwap} - \mathrm{Normal})_{\mathrm{NO}}]$, is $+38.00$pp, CI $[+31.00, +44.67]$. The secondary contrast measures sensitivity after removing readable names; it does not establish an original-path gain.

Earlier unmatched comparison. An earlier appendix version used a three-cell partial grid in which the seed-42, aug$= 0$, code=NO checkpoint had a 512-token context limit after an OOM, while the code=YES comparator had a 768-token limit. Those code-present versus code-absent contrasts were context-confounded and are superseded by the matched comparison reported here.

Before any new score was produced, a transcription error in the recorded overlap-audit identifier was corrected before scoring by exact prefix and content checks; the correction changed no model, population, pool, metric, or decision rule.

Because the aug=0.5, code=NO cell is missing and all rows are single-seed, this result does not provide a factorial interaction or establish a dominant 14B mechanism. This fixed-pool file-ranking contrast, specific to the tested training recipe, does not support a mechanism, mitigation, repair effect, or within-system own-scorer function certification, and it provides no reliable Normal-input gain.

E.2 7B Three-Seed Factorial Comparison

We ask how code previews and path augmentation separately change Normal-input localization and PathSwap sensitivity in a fully crossed Qwen2.5-7B comparison. All four SWE-only cells were trained from scratch under matched hyperparameters with seeds 42/43/44; code-present cells use 30-line previews and the Lite evaluations share the same $n = 300$ instances. At augmentation 0.5, half of the training batches use shuffled filenames with code fixed. These checkpoints are distinct from the legacy operating-point families in Tables 28–29.

An earlier training-code condition automatically enabled code previews whenever augmentation was nonzero, blocking the augmented no-code cell. After correcting that coupling, all four cells were trained, yielding the complete three-seed factorial grid. Table 25 reports means, seed standard deviations and effect rows; seed-42 paired tests are reported separately in Table 26. Paired bootstrap intervals concern fractional Recall@1 differences, while McNemar tests concern binary any-hit or explicitly named flip outcomes. Small differences from the legacy operating points reflect seed and implementation variation, not a common checkpoint family.

Table 25: **Codeaware fully-crossed 2×2 cross-cut (7B, SWE-only training, checkpoints trained after correcting the coupling, Lite n=300).** Cells are Recall@1 mean±std over 3 seeds (42/43/44). Effect rows use 3-seed means; paired McNemar code-preview contrasts in Table 26 use seed=42 alone (the seed=42 cells are 54.00/1.00% for (0,no) and 55.67/33.00% for (0,yes), giving paired diffs +1.67pp Normal and +32.00pp PathSwap on the seed-42 paired data, vs. the 3-seed mean differences of +1.33pp / +28.78pp; we report both to separate paired seed-42 tests from 3-seed means).

Cell	Recipe	Normal	PathSwap	Δ_{abs}	Δ_{rel} (%)
(0, no)	aug=0, code=No	53.00 ± 0.88	1.44 ± 0.51	−51.56	−97.3
(0, yes)	aug=0, code=Yes	54.33 ± 1.20	30.22 ± 3.24	−24.11	−44.4
(0.5, no)	aug=0.5, code=No	50.00 ± 1.00	3.11 ± 1.58	−46.89	−93.8
(0.5, yes)	aug=0.5, code=Yes	48.56 ± 2.59	36.78 ± 4.86	−11.78	−24.3
Effects					
code-effect at aug=0:	(0,yes)−(0,no)	+1.33	+28.78		
code-effect at aug=0.5:	(0.5,yes)−(0.5,no)	−1.44	+33.67		
aug-effect at code=No:	(0.5,no)−(0,no)	−3.00	+1.67		
aug-effect at code=Yes:	(0.5,yes)−(0,yes)	−5.77	+6.56		
code×aug interaction (PathSwap, descriptive)			+4.89		

The three-seed means in Table 25 identify the largest observed PathSwap contrast with the code-preview factor: +28.78pp at augmentation 0 and +33.67pp at augmentation 0.5. Their Normal-input counterparts are +1.33pp and −1.44pp, respectively. Augmentation without code adds 1.67pp under PathSwap and costs 3.00pp under Normal paths; with code, it adds 6.56pp under PathSwap at a 5.77pp Normal cost. Thus the grid describes a change in sensitivity and an accuracy trade-off, rather than a demonstrated Normal-input gain. The four-cell means give a descriptive PathSwap interaction of +4.89pp. The original table also lists a naive ±6.0pp summary without identifying its uncertainty convention; it is not used as a paired training-seed interval. Table 26 separately reports paired-instance tests for seed 42.

The seed-42 paired tests in Table 26 answer a narrower, instance-level question. At augmentation 0, the PathSwap code-preview difference on that seed is +32.00pp, with 96 discordant hits favoring code and none favoring no code ($p \approx 2.5 \times 10^{-29}$). The Normal contrast on that seed is +1.67pp and is not significant. The corresponding arm-total flip counts are 81 with code and 159 without code ($p \approx 1.2 \times 10^{-17}$ for the paired flip comparison). These results are not uncertainty estimates for the three-seed mean contrasts above.

E.3 SEED-42 PAIRED COMPARISONS

These seed-42 paired-instance results do not estimate uncertainty across the three training seeds in Table 25. A flip is a Normal top-1 hit becoming a PathSwap miss; the 14B flip-rate difference is no-augmentation minus Codeaware.

14B recipe comparison on Lite. Paired bootstrap 95% CIs use $B{=}10000$ resamples of the same 300 Lite instances; changes are in percentage points. The Codeaware Normal-to-PathSwap drop is 6.67 $[3.0, 10.7]$; its Normal contrast against no-augmentation is 4.00 $[-0.01, 8.3]$ (includes zero); the corresponding PathSwap contrast is 14.33 $[9.3, 19.3]$; the no-augmentation Normal-to-PathSwap drop is 17.00 $[11.3, 22.7]$; and the flip-rate difference is 13.67 $[9.0, 18.3]$.

Verified paired bootstrap 95% CIs (14B Codeaware vs. no-aug, $B{=}10000$, paired over $n{=}500$ Verified instances): Recall contrasts and their intervals use fractional Recall@1; the accompanying McNemar p-values use binary any-hit, except for the explicitly named binary flip-rate test. The Codeaware Normal-to-PathSwap drop is $+2.27$ $[+0.22, +4.47]$, McNemar $p{=}0.099$ (borderline); the Codeaware versus no-augmentation Normal contrast (the $+0.31$pp gain) has paired difference $+0.31$ (bootstrap mean $+0.32$, CI $[-1.87, +2.47]$), $p{=}0.871$ *(CI includes 0; no gain detected)*; the corresponding PathSwap contrast is $+12.48$ $[+9.25, +15.82]$, $p{\approx}1.2{\times}10^{-12}$; the no-augmentation Normal-to-PathSwap drop is $+14.43$ $[+10.73, +18.18]$, $p{\approx}8.6{\times}10^{-15}$; and the **flip-rate difference** is $+12.2$ $[+8.6, +15.8]$, McNemar $p{\approx}7{\times}10^{-16}$ (Codeaware flips $19/500$, no-augmentation flips $80/500$). The flip-rate difference is thus detected on both splits (Lite $p{\approx}3{\times}10^{-8}$), while the Verified Normal contrast detects no reliable gain ($+0.31$pp, CI includes zero).

7B code-preview comparison on Lite. On Lite, the paired (0,yes) minus (0,no) contrast in Table 25 is $+1.67$pp $[-2.0, +5.3]$ on Normal (includes zero) and $+32.00$pp $[+26.7, +37.3]$ under PathSwap.

Here b, c count the two discordant directions of paired binary outcomes, reported as $(b + c, |b - c|)$; the 14B flip row labels discordant counts, and the 7B row labels arm-total flips.

Table 26: **McNemar paired tests of binary top-1 any-hit or the explicitly named flip indicators on SWE-bench Lite ($n{=}300$).** The first five rows are 14B Codeaware versus 14B no-augmentation comparisons; the flip-rate difference has $p{\approx}3{\times}10^{-8}$. The final three rows isolate the 7B code-preview contrast at fixed aug=0 (seed=42 paired tests on checkpoints trained after correcting the coupling, used in the App. E 3-seed 2×2): all discordant PathSwap instances favor code-in-prompt ($p{\approx}2.5{\times}10^{-29}$), while the Normal contrast is not significant. Bold rows are the headline PathSwap and flip-rate tests.

Comparison	$(b{+}c, \lvert b{-}c\rvert)$	McN. p	Interpretation
14B Codeaware vs. 14B no-augmentation			
Codeaware: Normal vs. PathSwap	(36, 20)	$1.2{\times}10^{-3}$	PathSwap drop: significant
Codeaware vs. no-augmentation (Normal)	(44, 12)	$9.6{\times}10^{-2}$	+4.0pp Lite gain: n.s.
Codeaware vs. no-augmentation (PathSwap)	(65, 43)	$6.0{\times}10^{-8}$	Codeaware higher: ***
No-augmentation: Normal vs. PathSwap	(87, 51)	$3.3{\times}10^{-8}$	PathSwap drop: ***
Flip-rate difference	**(57, 41)**	$\mathbf{2.7{\times}10^{-8}}$	**Discordant flips: 49 vs. 8; *** **
7B code-preview contrast at aug=0 ((0,no) seed=42 vs. (0,yes) seed=42)			
Code-in-prompt (Normal)	(31, 5)	0.473	Normal contrast: n.s.
Code-in-prompt (PathSwap)	**(96, 96)**	$\mathbf{2.5{\times}10^{-29}}$	**Discordant hits: 96 vs. 0; *** **
Flip-count contrast	**(94, 78)**	$\mathbf{1.2{\times}10^{-17}}$	**Total flips: 81 vs. 159; *** **

Scale-up interpretation. The 14B comparison shows a difference in Normal-hit-to-PathSwap-miss rates: 14B no-aug flips on 23.0% of Lite instances versus 9.3% for 14B Codeaware (paired diff $+13.67$pp; bootstrap 95% CI $[9.00, 18.33]$pp; $p \approx 3{\times}10^{-8}$). Verified shows the same gap ($p \approx 7{\times}10^{-16}$, App. E). By contrast, Normal-path gains are borderline on Lite ($+4.0$pp, $p = 0.096$)

and null on Verified (+0.31pp, $p = 0.871$, CI $[-1.87, +2.47]$). Appendix E reports all discordant counts and comparisons.

Scaling this into a single LoRA (**Codeaware**) *narrows* the PathSwap gap—the 14B no-aug PathSwap drop is 2.6–6.4× Codeaware's—but does not *close* it (a $-11/-4.6\%$ residual). This is a diagnostic training-time contrast, not an established fix.

E.4 Body-Deletion Content Probe

To probe body content, we replace 80% of body lines with `# REDACTED` placeholders, retaining imports, class/def headers and the first non-blank line. Only the candidate-code preview changes; the Codeaware LoRA and data stay fixed.

Table 27: **Body-deletion content probe (7B Codeaware LoRA, code-in-prompt, BM25 top-100 pool).** This is the same-base 7B Codeaware checkpoint, distinct from the 14B recipe-table model; only the candidate-code preview is redacted.

Split / Condition	Baseline R@1	Redacted R@1	**Delta**	McNemar p
Lite Normal	49.33	48.33	−1.00	0.77
Verified Normal	43.32	41.80	−1.52	0.26
Lite PathSwap	40.67	26.67	−14.00	6.5×10^{-6}
Verified PathSwap	32.23	24.53	−7.70	2.2×10^{-5}

Body deletion changes little under Normal paths and substantially lowers Recall@1 under PathSwap. This is consistent with a larger role for body content under opaque identifiers, without separating semantics from the retained structural metadata.

E.5 Legacy recipe results and precision

This appendix groups the fixed-snapshot recipe results and their visualization; it does not merge the separate training protocols.

E.5.1 Recipe Operating Points

Scope of the legacy 7B comparison. The combined-data row and SWE-only Codeaware row in Tables 28–29 share a base model and evaluation pool, but their difference does not isolate training-data composition. The recovered SWE-only training configuration and combined-data launcher differ in negative sampling, context length, and checkpoint-save interval as well as training data. These records do not fully bind the terminal combined-data checkpoint, so they do not establish an otherwise matched comparison. This qualification concerns those two legacy 7B rows; the 14B contrasts and the separate three-seed 2×2 analysis (Appendix E) have their own stated scope.

Normal-path scores. The Normal columns describe operating-point scores under the stated recipes. The legacy combined-data and SWE-only Codeaware rows do not isolate a training-data effect.

Path sensitivity. The PathSwap losses describe path sensitivity under the stated recipes; paired contrasts and uncertainty are reported in Appendix E. Reduced PathSwap loss in a tested recipe establishes neither path invariance nor an end-to-end repair benefit.

Table 28: **Codeaware and same-base reference recipes under a fixed BM25 top-100 evaluation pool** on SWE-bench Lite ($n = 300$) and Verified ($n = 500$). Values are Recall@1 (%). *aug*: path-augmentation fraction at training time; *code*: short source preview; the 7B SWE-only Codeaware and combined-data rows use 50-line evaluation previews. Separately audited 30-line training configurations are run-specific, not a shared training setting for all rows. The combined-data versus SWE-only Codeaware comparison is across training recipes. Normal-path numbers should be read with PathSwap results in Table 29; the reported finding is reduced PathSwap sensitivity in the tested cells, not path invariance (§E.3).

Method	Train	aug	code	Lite	Verified
External baselines (different backbones)					
SweRankEmbed-Large	–	–	–	37.00	–
SweRankLLM-Small	–	–	–	47.93	–
Same-base 7B reference recipes (Qwen2.5-7B)					
SWE-only unaugmented recipe	SWE 1.9k	0.0	Yes	53.67	46.12
Codeaware	SWE 1.9k	0.5	Yes	48.33	44.98
Combined-data path-augmented recipe	SWE+GREPO 5.9k	0.5	Yes	56.67	48.13
Qwen2.5-14B scale-up (same recipe, larger backbone)					
14B no-aug	SWE+GREPO 5.9k	0.0	Yes	56.33	48.62
14B Codeaware (path-aug recipe)	SWE+GREPO 5.9k	0.5	Yes	**60.33**	48.93

Table 29: **PathSwap with opaque component identifiers:** 14B Codeaware has the smallest relative drop among the unaugmented and path-augmented recipe rows in the seed-42 table; the no-augmentation absolute PathSwap drop is 2.6× (Lite: 17.00 vs 6.66pp) and 6.4× (Verified: 14.44 vs 2.26pp) the Codeaware drop. BM25 top-100 pool from Table 28; PathSwap assigns an opaque identifier to every path component while leaving file contents unchanged. $\Delta_{\text{abs}} = \text{PathSwap} - \text{Normal}$ (pp); $\Delta_{\text{rel}} = \Delta_{\text{abs}}/\text{Normal}$. †*full repository-identifier cleaning*: 16 repository-specific identifier tokens in the 50-line snippet are consistently replaced with opaque identifiers; cleaned Δ_{rel} is -5.9% versus -11.0% uncleaned, a descriptive difference consistent with residual code-identifier association but not isolating a leakage mechanism.

Method	aug	Normal	PathSwap	Δ_{abs} (pp)	Δ_{rel} (%)
SWE-bench Lite (n=300)					
SWE-only unaugmented recipe	0.0	53.67	33.33	-20.34	-37.9
Codeaware SWE	0.5	48.33	41.00	-7.33	-15.2
Combined-data path-augmented recipe	0.5	56.67	47.33	-9.34	-16.5
14B no-aug	0.0	56.33	39.33	-17.00	-30.2
14B Codeaware (path-aug recipe)	0.5	**60.33**	**53.67**	**-6.66**	**-11.0**
Full repository-identifier cleaning control (14B, Lite):					
14B Codeaware clean†	0.5	56.67	53.33	-3.34	-5.9
SWE-bench Verified (n=500)					
SWE-only unaugmented recipe	0.0	46.12	26.43	-19.69	-42.7
Codeaware SWE	0.5	44.98	36.02	-8.96	-19.9
Combined-data path-augmented recipe	0.5	48.13	45.02	-3.11	-6.5
14B no-aug	0.0	48.62	34.18	-14.44	-29.7
14B Codeaware (path-aug recipe)	0.5	**48.93**	**46.67**	**-2.26**	**-4.6**

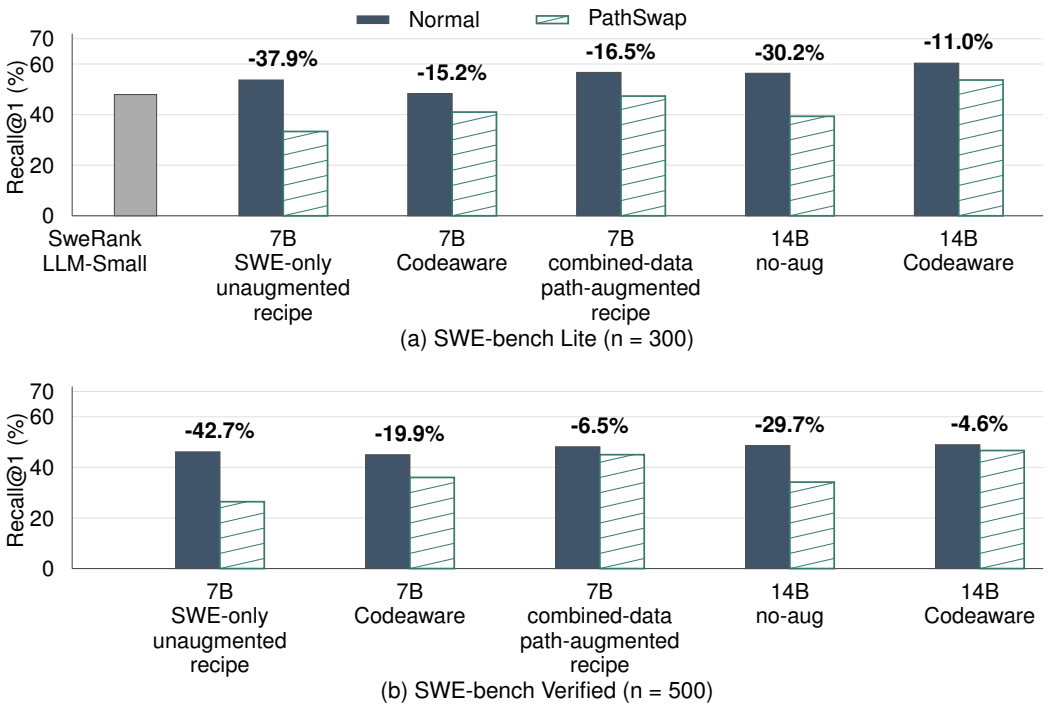


Figure 4: **Diagnostic visualization of Tables 28–29:** for the trained 7B and 14B LoRA cells, Normal Recall@1 and PathSwap drops; PathSwap relative drops are the primary sensitivity signal. The Verified normal-path gain over the same-base no-augmentation control is small ($+0.31$pp on $n{=}500$, paired bootstrap 95% CI $[-1.87, +2.47]$pp, including zero); the PathSwap-drop gap (Δ_{rel}) is the more reliable signal. Bar pairs show Recall@1 under normal paths (solid, colored) and under PathSwap (hatched, outlined). Every plotted value is exactly the value listed in Tables 28–29. Numbers above the bars show Δ_{rel}. Top: SWE-bench Lite ($n{=}300$); bottom: SWE-bench Verified ($n{=}500$). Same BM25 top-100 file pool, pointwise Recall@1; SweRankLLM-Small same-pool baseline shown for Lite only. This appendix figure is a lever diagnostic, not a system or mitigation proposal.

Figure 4 visualizes the same fixed-snapshot diagnostic reported in Tables 28–29.

E.5.2 Precision Sanity Check

A matched-precision sanity check on SWE-bench Lite (bf16 inference with identical prompts, candidates, and seed) across all three rerankers (SWE-only unaugmented recipe, Codeaware, combined-data path-augmented recipe) $\times$ normal/PathSwap (six cells total) finds small absolute differences ($|\Delta| \leq 2.0$pp, mean $|\Delta| = 1.0$pp). The PathSwap relative drop is negative in both precisions for all three rerankers: the SWE-only unaugmented recipe has a 4-bit drop of -37.9% versus a bf16 drop of -36.1% (gap 1.8pp); Codeaware has -15.2% versus -19.5% (gap 4.3pp); the combined-data path-augmented recipe has -16.5% versus -16.0% (gap 0.5pp). The largest precision-induced relative-drop gap is 4.3pp (Codeaware), and both precisions preserve the qualitative ranking: the SWE-only unaugmented recipe is the most path-dependent, while Codeaware and the combined-data path-augmented recipe are comparably less PathSwap-sensitive and 2–2.5$\times$ less dependent. We therefore report 4-bit numbers only for the checkpoint comparisons in Tables 28–29 and tables derived from them. The PathAlias cells (§3, Table 10), organic-drift arms (Appendix B.5), and pre-registered certification protocol (Appendix C) are bf16 as stated in their sections; their Normal baselines are run-specific. For Lite, the two 14B PathAlias Normal baselines in Table 10 differ from the corresponding legacy baselines by at most 1pp, whereas the 7B SWE-only Normal is 56.0% rather than 53.67%. This cross-run comparison is not the six-cell matched-precision comparison above.

F METADATA AND CODE-USE DIAGNOSTICS

Table 30: Displayed-path controls. Panel A: path-only GREPO cross-encoders; fractional Recall@1 (%, $n = 1{,}704$); parentheses give the relative change from each model's single-checkpoint Normal baseline. Delexicalize replaces path tokens also present in the issue with opaque identifiers. Panel B: historical gold-path replacement on 862 genuinely renamed examples with code-in-prompt 14B recipes; higher-cutoff results are in Table 16.

A. GREPO displayed-path interventions

Intervention	Qwen2.5-7B	Llama-3.1-8B	Qwen3-8B
None (Normal)	27.01	28.35	29.38
Remove module names	27.70 (+3%)	25.48 (−10%)	26.43 (−10%)
Flatten to single level	20.93 (−23%)	0.63 (−98%)	0.67 (−98%)
Swap leaf directories	17.19 (−36%)	1.93 (−93%)	2.03 (−93%)
Shuffle directories	14.72 (−46%)	2.17 (−92%)	2.17 (−93%)
Delexicalize	8.85 (−67%)	3.72 (−87%)	3.90 (−87%)
Shuffle filenames	7.24 (−73%)	7.11 (−75%)	7.60 (−74%)
PathSwap (opaque IDs)	1.06 (−96%)	1.30 (−95%)	1.38 (−95%)

B. Historical-rename calibration

Recipe	Current	Historical	Change pp [95% CI]
No augmentation	22.13	18.66	−3.47 [−5.12, −1.82]
Codeaware	23.16	20.94	−2.22 [−3.64, −0.84]

F.1 STATIC METADATA ASSOCIATIONS

Paths predict graph structure. Logistic regression on path features alone predicts co-change edges with AUC= 0.667 and import edges with AUC= 0.728 (Appendix F.5). The most predictive features are Jaccard path-component overlap (+5.01 log-odds for import) and leaf-filename similarity (+1.43 for co-change). These path features predict held-out edge labels above chance; this association does not show that the reranker encodes graph topology or identify a causal mechanism.

Path overlap tracks reranker gains in the tested strata. Stratified by the token overlap between the bug report and file paths, high-overlap examples gain +19.0pp from reranking, medium +12.8pp, and low only +6.2pp—a pattern consistent with the reranker *amplifying* path-associated signal; this analysis does not isolate how much independent code signal remains. Input-gradient attribution ($|\nabla_{\mathbf{e}} \cdot \mathbf{e}|$) on 300 GREPO examples provides a concordant, model-specific diagnostic: path tokens receive $4.8\times$ higher per-token attribution density than issue tokens (22.9% of total attribution from ∼4% of tokens; Table 39). Large path-perturbation sensitivity recurs across the tested model families, benchmarks, and languages; the gradient-attribution statistic itself is from one Qwen2.5-7B model. The SweRankEmbed row is a separate path-document encoder diagnostic, not evidence about its native function-body retrieval-and-ranking procedure (§G.1; Tables 46 and 48).

F.2 FILENAME AND DIRECTORY PERTURBATIONS

Table 30 compares separate test-time interventions against each model's own GREPO baseline. Filename-only results for additional scoring procedures follow in Table 31.

The complete seven-perturbation matrix is in Table 30A. The following analyses retain the cross-model interpretation and filename controls for additional scoring procedures.

The three model columns in Table 30 retain different directory-level sensitivity patterns despite their large filename and opaque-path drops.

Across four tested scales from 0.5B to 7B, rerankers collapse to 1–2% Recall@1 under PathSwap, and the relative drop increases over those cells (−96.1% at 7B versus −91.8% at 0.5B; Appendix J.4.1).

The additional rows change scoring procedure as well as model, so their magnitudes are read within the stated metric family.

Table 31: Filename shuffling diagnostic on GREPO-Test (1704 examples). **Gap** = Standard − Shuffled. **Ratio** = Shuffled / Standard. Lower Gap and higher Ratio indicate less path dependence. Cross-encoder and code-residual rows use fractional Recall@1; generative and bi-encoder use Hit@1 (binary top-1 hit); relative drops should be interpreted within metric family rather than compared directly across metrics.

Model	Standard	Shuffled	Gap ↓	Ratio ↑
Generative (Qwen 7B)	41.55	19.54	22.01	0.47
Path-only (Qwen3-8B)	29.38	7.60	21.78	0.26
Path-only (Qwen2.5-7B)	27.01	7.24	19.77	0.27
Path-only (Llama-3.1-8B)	28.35	7.11	21.24	0.25
Code-centric reranker	28.74	6.62	22.12	0.23
Combined candidates (no reranker)	22.51	not evaluated	not applicable	not applicable
Code-residual reranker	22.68	6.04	16.64	0.27
Bi-encoder (E5-large)	9.50	2.92	6.58	0.31

F.3 Code-input and selected-subset diagnostics

We separate prompt-content comparisons, trained code-based scorers, code-availability subsets and additional protocol controls. Aggregate results and selected subsets answer different questions; each comparison retains its stated checkpoint and pool.

F.3.1 Scorer Prompt and Training Comparisons

Table 32 records prompt-content and training-input comparisons under their stated configurations.

Table 32: Recorded scorer prompt and training comparisons, not a verified same-checkpoint prompt ablation. The path-only row has the highest listed Recall@1; a separate 3-seed path-only mean is 28.92±0.56% (Table 72). This is not a general claim that code content has no value.

Condition	Recall@1	R@5
Path only (default)	**27.01**	**49.17**
Path + content summary	23.01	39.89
Path + content (init from path-only)	22.94	39.47
Content training → path-only eval	22.28	41.28

Path-associated signal is large in the tested scorers. Adding file content does not displace the path signal: a content-aware reranker that augments paths with file-content summaries achieves Recall@1= 23.01%, lower than the path-only baseline despite having strictly more information, and all content-integration variants underperform the same baseline (Table 32; we return to where code *should* help in §F.3.3).

F.3.2 Decomposed and Content-Based Scorers

Decomposed reranker setup. Pointwise scoring is a black box: it can route on any combination of path tokens and code tokens. To disentangle the channels, we train a code-residual reranker that receives paths whose components use opaque identifiers plus full file code content, encouraging reliance on code signals rather than path identity. This model achieves Recall@1 = 22.68%—substantially below the path-only model (28.92±0.56%).

Alpha-sweep fusion. We linearly fuse the code-residual and path-only scores: $s_{\text{fuse}} = \alpha s_{\text{code-residual}} + (1-\alpha)s_{\text{path}}$, sweeping $\alpha \in \{0, 0.1, \ldots, 1\}$ on the held-out validation split. The optimal α on the validation set is $\alpha = 0$—*pure path-only*—so no additive validation benefit from code content is detected in this configuration (Table 33).

Counterfactual and code-centric controls. A matched-compute code-centric scorer (real paths + 50 lines of code + AST signatures) achieves Recall@1 = 28.74%—no better than path-only

Table 33: Decomposed reranker: path-only vs. code-residual scores and linear fusion. The validation α-sweep selects α=0; no additive validation gain is detected for this residual. Code-residual and fusion rows are single-seed; the 6+pp gap exceeds the path-only seed standard deviation ($\pm$0.56). The previously reported code-only endpoint is withheld because its checkpoint, split, and score normalization are not bound to this sweep.

Model	Recall@1 (%)
Path-only (Qwen2.5-7B, 3-seed)	**28.92 ± 0.56**
Code-residual reranker (opaque-identifier paths + code)	22.68
Best fusion (α sweep on val)	28.92 (α=0 selected)

(28.92$\pm$0.56%, $p > 0.20$)—and collapses equally under filename shuffling (Table 42). Path-only wins on all subsets except issue-path mismatch (2.4% of the 1,704-example test (n=41), +7.2pp for code-centric); training with 50% shuffled paths achieves 6.54% under shuffling.

Stronger code baselines: long context and selective retrieval. We train two additional code-aware rerankers reported below: (i) a **long-context** scorer (200 lines, 2048 tokens), and (ii) a **selective retrieval** scorer (BM25 top-3 functions per file). These two scorers process 8–16$\times$ more tokens than the path-only comparison. Neither is detectably better than path-only in aggregate: selective retrieval 28.44$\pm$0.62% (2 seeds), long-context 28.56$\pm$0.23% (3 seeds), vs. path-only 28.92$\pm$0.56%. All listed code-aware variants show large relative decreases under filename shuffling ($-$71% to $-$77%). Per-slice analysis reveals a nuanced picture:[1] selective retrieval gains +5.2pp on the predefined Code-Crucial slice but loses $-$2.1pp on the same-stem slice, while the aggregate comparison is near zero. These slice associations do not identify a general code-use mechanism.

Table 34: Stronger code-aware baselines vs. path-only. Neither is detectably better than path-only in aggregate on the unshuffled comparison. In the unshuffled Recall@1 column, parentheses give seed standard deviations in units of the last displayed digit. The path-only shuffle value 7.24 also appears against the single-checkpoint Normal baseline 27.01 in Tables 30 and 46 ($-$73% there); the displayed $-$75% instead uses the three-seed Normal mean 28.92 as its denominator. Without a verified seed-matched shuffle aggregate, $-$75% is not a multi-seed paired robustness estimate.

Model	Code tokens	Recall@1 (%)	Shuffle Recall@1	Drop
Path-only (3-seed)	0	28.92(56)	7.24	$-$75%
Code-centric (50L, 768 ctx)	∼375	28.74	6.62	$-$77%
Selective retrieval (top-3, 2-seed)	∼800	28.44(62)	7.47	$-$74%
Long-context code (200L, 3-seed)	∼1500	28.56(23)	7.37(6)	$-$74%
Code-residual long-ctx (opaque-identifier paths)	∼1500	25.19	7.38	$-$71%

F.3.3 Subsets Where Code Helps

Where code may matter. Aggregate nulls can hide subset effects. We therefore construct **Code-Crucial**: 116 GREPO examples (6.8% of the test set) with bottom-quartile issue/path overlap, misleading path cues, and a code-availability witness. On this subset, selective function retrieval (BM25 top-3 functions per candidate) reaches 40.52% Recall@1 against the path-only 35.34% (+5.18pp), while brute-force code variants (residual model with paths with opaque component identifiers and 30 lines: 12.97%; zero-shot 200-line scorer: 12.93%) fail. The shuffle collapse (the Global cell shared with Table 44) drops from 90% globally to 75% within the correct file (Table 35), showing a smaller collapse in this conditioned slice; the design does not identify a resolution threshold. Sensitivity over seven tested definitions shows that code-residual underperformance persists ($\Delta = -1.5$ to -5.1pp; Table 41). The feature-defined Code-Crucial used here has Jaccard 0.18 against an earlier model-defined variant, so the two slices differ substantially; the low overlap does not establish construction independence. Of 25 Code-Crucial failures audited, 15 ground-truth

[1] Slice-level comparisons are computed on fixed subsets: Code-Crucial (116 examples, §F.3.3), low-overlap (bottom quartile by issue-path Jaccard), and same-stem (904 examples with duplicate candidate filenames).

files are code-identifiable; per-repository path-only Recall@1 varies by 14.4pp between descriptive and opaque repositories (Appendix F.3.3). Net: aggregate code-aware comparisons are NULL or negative, while selective function retrieval improves on this predefined path-failure slice; this bounded contrast motivates the diagnostic analyses in §E.2.

Per-example decomposition. Path-only is uniquely correct on 17.1% of examples (2.4× code's 7.2%); code helps on a small mismatch slice (2.4% of the 1,704-example test (n=41), +7.2pp), but that slice is too rare to shift aggregates. Further decomposition of code signals (identifier obfuscation, attention analysis, issue-path mismatch slicing) appears in Table 42 and Appendix F.4.

Repository Path Descriptiveness We classify each repo as "descriptive" or "opaque" by its median path–issue token overlap, split at the across-repository median. Path-only reaches 35.8% Recall@1 on repos above and 21.4% on repos below the median path–issue token overlap (−14.4pp). This split partly explains the association; it does not identify a causal path-informativeness effect.

F.3.4 Additional Protocol Controls

Two-stage and code-only controls. Historical records report Recall@1 = 21.50% for a path-first, content-reranked two-stage procedure and 24.51% for path-only. Missing executed checkpoint and pool bindings prevent treating these values as a matched comparison. Removing paths entirely and scoring by code content alone ("content-replace") collapses to 6.85%. Post-scoring graph propagation is also negative, and mixed-retriever-pool training drops Recall@1 from 27.0% to 20.0%.

Candidate-pool scope check. Scoring **all Python source files** per repository (avg. 541 files) yields path-only Recall@1 = 29.71%, matching the BM25-based procedure (28.92±0.56%), showing that the high path-only score is not confined to the BM25 candidate pool; this comparison does not isolate an intrinsic scorer property.

Hierarchical localization. Does code matter more when the search scope is narrowed? We compare function-level localization in global vs. gold-file-conditioned settings (Table 35).

Table 35: Conditioned-scope function-localization controls. The Global comparison is reported once in Table 44: function Recall@1 10.95 to 1.06 and function Hit@1 32.52 to 4.58, about1200 candidates. The gold-directory and gold-file rows here retain their distinct candidate scopes; no causal mechanism follows.

Scope	Baseline Recall@1 (%)	Shuffle Recall@1	Drop	Avg cands
Gold directory	13.76	1.62	−88%	∼2033
Gold file	17.91	4.47	−75%	204

The 90% global drop falls to 75% within the correct file. This narrower, conditioned comparison shows lower shuffle sensitivity, without identifying a resolution threshold or semantic mechanism.

Table 36: PathSwap across scales: stable structure-preserving opaque identifiers; no random baseline or claim beyond these scales.

Model	Setting	Recall@1	Δ	R@5	Δ	R@10	Δ
0.5B	original	20.02	reference	38.69	reference	47.46	reference
0.5B	PathSwap	1.65	−91.8%	4.97	−87.2%	8.50	−82.1%
1.5B	original	21.54	reference	41.16	reference	48.74	reference
1.5B	PathSwap	1.63	−92.4%	4.52	−89.0%	8.16	−83.3%
3B	original	23.37	reference	43.58	reference	51.24	reference
3B	PathSwap	1.55	−93.4%	3.70	−91.5%	6.86	−86.6%
7B	original	27.01	reference	49.17	reference	56.45	reference
7B	PathSwap	1.06	−96.1%	4.09	−91.7%	6.86	−87.8%

Probe and breadth conclusions. Heuristic labels (rules unspecified) on correct top-1 predictions are keyword match (∼52%), substring match (∼17%), override (∼20%), and zero textual signal (∼10%); duplicate-filename directory selection is 67.2%. Graph-neighbor conditional accuracy is 45.9% for graph training versus 50.1% without it, so graph training does not reduce this confusion and the diagnostic does not identify hardness, topology, or another cause.

F.4 Path Controls and Attribution Diagnostics

Readable-path examples, opaque controls, attribution and slice diagnostics answer separate questions.

Table 37: Readable paths and the same hierarchy with opaque component identifiers.

Readable path	Same hierarchy with opaque component identifiers
`cirq/value/linear_dict.py`	`d_f4e8/d_a406/m_008e.py`
`cirq/value/linear_dict_test.py`	`d_f4e8/d_a406/m_d8be.py`
`cirq/__init__.py`	`d_f4e8/m_3414.py`
`cirq/ops/__init__.py`	`d_f4e8/d_0cf0/m_3414.py`
`cirq/circuits/circuit_test.py`	`d_f4e8/d_1dc6/m_921d.py`
`cirq/google/sim/xmon_sim_test.py`	`d_f4e8/d_d493/d_0736/m_7fbe.py`
`cirq/sim/__init__.py`	`d_f4e8/d_0736/m_3414.py`

Table 38: PathSwap scale results on GREPO-Test ($n = 1704$).

Model/pool	Standard	PathSwap	Drop
0.5B graph	20.02	1.65	−92%
3B graph	23.37	1.55	−93%
7B graph	27.01	1.06	−96%
0.5B hybrid	24.12	1.36	−94%
3B hybrid	28.82	0.92	−97%
7B hybrid	20.00	0.39	−98%

Table 39: Gradient attribution (mean ± std); density is $\times 10^3$.

Category	Mean tokens	Token share	Attribution	Density
Path	6.2	3.7%	22.9(118)%	40.0
Issue	140.0	82.5%	51.2(185)%	8.3
Template	23.6	13.9%	25.9(110)%	11.0

Table 40: Path predictability (1,321 issues, 66K triples, GroupKFold).

Feature set	AUC
Path lexical overlap with issue	0.636
Path TF–IDF	0.640
Issue + Path TF–IDF	0.589
Path structure only	0.587

Path and attribution conclusions. PathSwap replaces each directory and filename stem with deterministic opaque identifiers, preserving depth, extension, content, graph topology, and labels while changing token familiarity and distribution; PathAlias carries the natural-form argument. Identical `__init__` stems and shared directory prefixes retain package and co-location regularities. Applying opaque identifiers to all components changes filename and directory components jointly, so its 92–98% collapse versus 53–77% for filename shuffling does not decompose their contributions. On 300 GREPO examples, path tokens are about 4% of tokens but receive 22.9% of attribution, a $4.8\times$ per-token density advantage over issue tokens; the density columns are ordinal rather than arithmetically recoverable from aggregate means. Grouped path-only TF–IDF reaches AUC 0.640,

Table 41: Code-Crucial sensitivity; the 116-example definition in Section F.3.3 is stricter.

Definition	N	Path	Code-residual	Δ
Original (frac<0.1, witness)	265	19.1	17.5	-1.6
Strict (frac<0.05, witness)	215	22.2	19.8	-2.4
Loose (frac<0.2, witness)	489	20.8	17.6	-3.2
No witness (frac<0.1)	454	11.7	10.2	-1.5
No witness (frac<0.2)	727	14.4	11.8	-2.6
Jaccard<0.05, witness	962	31.7	26.5	-5.1
Jaccard<0.1, witness	1271	31.8	26.9	-4.9

and adding issue text lowers it to 0.589; this linear specification neither quantifies the neural path share nor identifies its mechanism. Across seven broader Code-Crucial definitions, code-residual is -1.5 to -5.1pp below path-only; without intervals or tests this is directional only, and the 116-example definition in Section F.3.3 is stricter.

Table 42: Code-aware versus path-only; CC is real paths plus code and CR is PathSwap plus code. Parentheses in the Path-only full-test value give the three-seed standard deviation in units of the last displayed digit: $28.92(56) = 28.92 \pm 0.56$. Cells marked not evaluated were not run for that model.

Test condition	Path-only	CC (50L)	CC (200L)	CR (100L)
Full test ($n = 1{,}704$)	28.92(56)	28.74	28.94	24.19
Shuffle filenames	7.24	6.62	not evaluated	6.33
Same-stem ($n = 904$)	24.04	23.08	not evaluated	17.67
Hard ($n = 904$)	14.98	14.18	not evaluated	11.01
Issue-path mismatch ($n = 41$)	27.59	34.81	not evaluated	not evaluated

Code-signal and counterfactual conclusions. Path-only is uniquely correct on 17.1%, code on 7.2%, both on 32.3%, and neither on 43.5%; the $2.4\times$ asymmetry is descriptive and the tested fusion selects $\alpha = 0$. On 41 extreme-mismatch examples, code-centric is 34.81% versus path-only 27.59% ($+7.2$pp): the only diagnostic point estimate on which code-centric is higher, and on a 2.4% slice of the 1,704-example test. Identifier obfuscation with PathSwap changes 22.68% to 17.39% (-5.3pp), versus 1.1pp with real paths; these are sensitivities, not a signal decomposition. Last-layer attention density is $12.2\times$ higher for path than code tokens despite code being 53% of input; attention is descriptive, not causal. On the 1,546 records with valid swap pairs, the path-only counterfactual inputs and 32.84% results are identical by construction, a consistency check rather than a finding; this subset excludes examples without valid swap pairs. Path-only wins or ties within seed variance on the displayed code-aware subsets except issue-path mismatch. In the GREPO-only, no-code-in-prompt cell, 50% shuffle augmentation scores 26.01% normally versus $28.92 \pm 0.56\%$, and 6.54% shuffled versus 7.24%; 100 versus 30 code lines score 24.60% versus 24.09%, without reported uncertainty. This does not identify which joint difference from the later Codeaware recipe matters. On five path-selected subsets—de-leaked ($n = 2{,}175$), same-stem ($n = 904$), hard ($n = 904$), same-directory confused ($n = 334$), and path-misled ($n = 229$)—path-only scores 18.05%, 24.04%, 14.98%, 9.04%, and 8.45%, respectively, versus 28.92% on the 1,704-example test; selection prevents a causal reading.

F.5 Path Features Predict Graph Structure

To quantify how well path-derived features predict repository graph edges, we train logistic regression classifiers to predict co-change and import edges from path-derived features alone. For each repository, we construct positive pairs (file pairs connected by the target edge type) and sample $5\times$ negative pairs (unconnected files). Four features are computed per pair:

1. **Jaccard path components**: Jaccard similarity over the set of directory/file components in each path.
2. **Shared directory depth**: number of shared leading path components.

3. **Leaf filename similarity**: character-level Jaccard similarity of the filename (without extension).
4. **Path edit distance**: Levenshtein distance between full paths, normalized by max length.

Table 43: Path features predicting graph edges. AUC averaged across repositories with $\geq$10 positive pairs.

Edge type	AUC	Top feature (log-odds)
Co-change	0.667	Leaf filename similarity (+1.43)
Import	0.728	Jaccard path components (+5.01)

The path-feature classifier predicts held-out edge labels above chance. This does not show that the cross-encoder encodes graph topology in path semantics.

F.6 Function-Level Extension

When we extend CodeGRIP to function-level localization (ranking functions within predicted files), two evaluation protocols give different numbers depending on which candidate pool is used: (i) on a pool of $\sim$1200 function-level candidates per instance (top-50 BM25 files expanded by precomputed function index), the reranker achieves Func Hit@1 = 32.52% (Table 44, baseline row); (ii) on a flat function pool that includes functions from other files, we obtain Hit@1 = 19.86% vs. random 7.46% (2.7$\times$). These are protocol-specific operating points rather than one comparable function-level score; their differences reflect pool size and whether non-predicted files contribute candidates.

Function-level path perturbation. To test whether the path prior extends beyond file-level localization, we evaluate at function granularity. For each test example, we expand the top-50 BM25 file candidates into $\sim$1200 function-level candidates using a precomputed function index, yielding path-qualified function candidates.

Table 44: Function-level evaluation (1113 evaluable examples out of 1583 with function GT). Shuffling function names across files causes a 90% descriptive drop under this function-name intervention; results condition on evaluability.

Condition	Func Recall@1 (%)	Func Hit@1 (%)
Baseline	10.95	32.52
Shuffled (names permuted across files)	1.06	4.58
Drop	**-90%**	**-86%**

Shuffling function names across files causes a 90% drop in function-level Recall@1 (10.95% $\rightarrow$ 1.06%) in this cell, showing sensitivity to the displayed function-name surface. This is not a file-path intervention or a PathAlias result.

F.7 Content-Aware Training Control

Training the reranker with file content summaries in prompts, then evaluating path-only, yields Recall@1 = 22.28% versus path-only training (29.38% with best checkpoint). This point comparison is consistent with a prompt mismatch in this cell, but without an uncertainty interval it does not establish a general training rule.

G Scope and Candidate-Pool Sensitivity

G.1 Breadth across Tested Settings

We ask whether filename-shuffle sensitivity is confined to one benchmark, scoring interface or language. Each table retains its own population and metric family.

For the frontier-API rows, the archived captions disagree on whether the top-one metric is binary Hit@1 or fractional Recall@1. We retain the historical values pending run-level metric binding and do not use these rows for cross-metric comparisons.

Table 45: Historical top-one operating points and filename-shuffle sensitivity by setting. These rows retain their individual model, benchmark and sample-size identities. Metrics must be interpreted within the recorded protocol; the summary is not a cross-metric ranking. Path-document encoder cells are distinct from native function-body retrieval.

Setting	Arch.	Bench.	N	Normal	Drop
Qwen2.5-7B (GREPO cross-enc.)	Cross-enc.	GREPO	1704	27.01	−**73**%
Llama-3.1-8B	Cross-enc.	GREPO	1704	28.35	−**75**%
Qwen3-8B	Cross-enc.	GREPO	1704	29.38	−**74**%
Qwen2.5-7B SWE-adapted	Cross-enc.	SWE-Lite	300	50.67	−**78**%
Bi-encoder	Bi-enc.	GREPO	1704	9.50	−**69**%
Generative listwise	Listwise	GREPO	1704	41.55	−**53**%
Qwen3-Max + code	API	GREPO-sub	152	55.26	−**45**%
Qwen3-Max agentic	Agentic	GREPO-sub	77	54.55	−**62**%
SweRankEmbed-Large	Dense (path-doc)	SWE-Lite	300	37.00	−77.5%
Python reranker	Cross-enc.	BeetleBox	740	14.43	−**60**%

Filename-shuffle sensitivity is large across the tested GREPO/SWE/BeetleBox settings, scorer interfaces, and Python/Java cells. Additional single-seed scope cells cover the tested 1.5B–14B scales (Table 51), Qwen/Llama/Mistral families (Table 52), and in-domain Java data from BeetleBox and Bench4BL (Tables 54 and 55). Table 45 consolidates filename-shuffle drops across settings spanning three axes: **benchmark** (GREPO, SWE-bench Lite, BeetleBox; Verified sensitivity is analyzed separately in Tables 28–29), **architecture** (cross-encoder, bi-encoder, generative listwise, frontier API, agentic, plus a dense encoder under a path-document diagnostic), and **language** (Python, Java). The individual historical protocol rows include Qwen3-Max with 100 lines of code per candidate (−45%) and Qwen3-Max in an agentic chain-of-thought setup with 80-line previews of the top-5 candidates (−62%). GREPO-trained models also transfer zero-shot to SWE-bench Lite at 48–51% Hit@1, competitive with the SWE-adapted reranker (50.67%; Table 47). When applied to file-path documents, the released SweRankEmbed-Large encoder is also sensitive (56–78% Recall@1 reduction under filename, directory, and flatten perturbations; Table 48), showing that the path-string channel is usable by that encoder in this diagnostic. This does not establish sensitivity of the native function-body retrieval-and-ranking procedure, which is evaluated separately. A Python-trained reranker applied zero-shot to 740 Java bugs from BeetleBox (Chakraborty et al., 2025) achieves Recall@1= 14.43% and collapses by −60% under filename shuffling. That result shows sensitivity in this zero-shot Python-to-Java cell, but does not establish language-general transfer.

Cross-benchmark asymmetry of the directory-level prior. On SWE-bench Lite (Table 47, bottom), the directory-level pattern inverts relative to GREPO (Table 30). Qwen2.5 is now the *most* sensitive model to directory shuffling (−71%) and flattening (−39%), while Llama-3.1 and Qwen3 are comparatively more robust. Directory sensitivity is benchmark-dependent in these cells; the design does not identify whether repository naming regularities or another benchmark difference causes the reversal. Filename-level vulnerability, by contrast, is consistently large in the tested cells (−77% to −80% on Lite; GREPO rows are in Table 30). This supports breadth across the tested model families and benchmarks, not architecture- or benchmark-invariance.

Repository-level breadth. Filename shuffling causes >20% Recall@1 drop in 94% of repositories (44/47 repos with $n \geq 5$ in this BM25 top-100 subset), with a mean drop of 69% (median 75%). Even in the bottom-quartile repos by baseline Recall@1, the mean drop is 58%. Combined with cross-benchmark (SWE-bench, 12 repositories, −77–80%) and cross-language (BeetleBox Java, −60%) evidence, the observed sensitivity is not confined to one repository set. This does not establish invariance across naming conventions or languages.

Table 46: Path perturbation across recorded scoring interfaces. Cross-encoder rows use fractional Recall@1; bi-encoder, generative listwise and agentic rows use binary Hit@1. The Python-to-Java cell uses fractional Recall@1. Frontier API rows retain their own sample sizes (77–152 depending on model) and their historical values, but their binary-versus-fractional metric identity is unresolved. No cross-metric aggregation is reported.

Architecture	Training	Baseline (%)	Shuffle (%)	Drop
Cross-encoder (Qwen2.5-7B)	Fine-tuned	27.01	7.24	−73%
Cross-encoder (Llama-3.1-8B)	Fine-tuned	28.35	7.11	−75%
Cross-encoder (Qwen3-8B)	Fine-tuned	29.38	7.60	−74%
Cross-encoder (Qwen2.5-14B)	Zero-shot	16.21	4.70	−71%
Bi-encoder (E5-large-v2)	Zero-shot	9.50	2.92	−69%
Generative listwise (Qwen2.5-7B)	Zero-shot	41.55	19.54	−53%
Frontier API (DeepSeek-V3.2)‡	Zero-shot	5.06	2.72	−46%
Frontier API (Qwen-Max)‡	Zero-shot	22.45	12.57	−44%
Frontier + code (Qwen3-Max, 100L)‡	Zero-shot	55.26	30.26	−45%
Agentic + code (Qwen3-Max, CoT)‡	Zero-shot	54.55	20.78	−62%
Cross-language (Python→Java, BeetleBox, 740 examples)				
Cross-encoder (Qwen2.5-7B)	Fine-tuned	14.43	5.79	−60%

Table 47: **Top:** SWE-bench Lite adaptation and zero-shot transfer (the adapted row is not zero-shot). **Bottom:** SWE-bench Lite path perturbation; the Qwen2.5 row uses the SWE-adapted checkpoint, while Llama3.1 and Qwen3 use GREPO checkpoints. Filename shuffling causes a 77–80% collapse across all three model families. Bold marks the overall best (top) and per-column best (bottom). For Δ%, black denotes a 5–50% drop and red a 50–90% drop.

Model	Hit@1 (%)
Qwen2.5 *adapted*	50.67
Qwen2.5 GREPO	48.00
Llama3.1 GREPO	48.67
Qwen3 GREPO	**51.33**

Model	Filename	Dirs	Flatten
Qwen2.5 adapted	11.0 (−78%)	14.7 (−71%)	30.7 (−39%)
Llama3.1	9.7 (−80%)	**22.0** (−55%)	34.3 (−29%)
Qwen3	**11.7** (−77%)	20.0 (−61%)	**39.0** (−24%)

Table 48: Path perturbation on SweRank (SWE-bench Lite, n=300). *Filename shuffle* permutes file names within each directory; *shuffle dirs* permutes directory names; *flatten* collapses the directory hierarchy. SweRankEmbed-Large uses the file-path string as the retrieval document; under perturbation only the path string changes. This isolates encoder sensitivity to path-string content and does not establish sensitivity of SweRank's native function-body retrieval-and-ranking procedure, which is evaluated separately and shows no significant PathSwap change.

	Small (137M)		Large (7B)	
Perturbation	Recall@1	Drop	Recall@1	Drop
None (baseline)	26.33	reference	37.00	reference
Shuffle filenames	6.00	−77%	8.33	−77.5%
Shuffle directories	not evaluated	not applicable	15.00	−59.5%
Flatten directories	not evaluated	not applicable	16.33	−55.9%

The encoder responds differently to filename, directory and flattening interventions. These path-document cells remain separate from the native function-body workflow in Appendix I.

G.2 Retrieval Rebuild under PathSwap

To address the fixed-pool conditioning concern, we rebuild BM25 retrieval over the corpus with PathSwap identifiers (filenames use opaque identifiers but content is intact), regenerate top-500 candidates, and retain the historical Codeaware and no-augmentation results below; these do not use the rerank-only checkpoints. The rebuilt pool has lower top-100 oracle recall on Lite (79.00% vs. the original 86.67%) and higher top-100 oracle recall on Verified (82.80% vs. the original 66.60%; the original binary-any-hit count is 333/500).

Split	Codeaware Recall@1	no-aug Recall@1	Delta	McNemar p	rerank-only gap
Lite e2e	36.67	27.33	+9.33	3×10^{-4}	+14.33pp p=6e-8
Verified e2e	37.72	27.20	+10.52	6.9×10^{-8}	+13.40pp p=1e-12

The historical audit assigns the rebuilt-pool rows and rerank-only gaps to different checkpoint families; on Verified, the latter gap is binary Hit@1, not fractional Recall@1. Original paired run bindings are missing, so this table cannot establish preservation of a matched contrast after retrieval rebuild or invariance to other candidate-generation procedures.

G.3 Held-out repositories and cross-language pools

Table 49: Inductive evaluation on 560 examples from 12 held-out repositories.

Method	**Recall@1**	**R@5**	**R@10**	***p*-value**
Combined pool (BM25 ordering)	20.52	38.47	46.69	reference
Inductive reranker	20.77	46.70	55.37	R@5/R@10: < 0.0001 ***

Table 50: GREPO-trained Python reranker on BeetleBox Java ($n = 740$, BM25 top-200).

Condition	**Recall@1**	**R@5**
Baseline	14.43	29.89
Shuffle filenames	5.79	not reported
Drop	-60%	not evaluated

In the Java cell, BM25-only outperforms graph-hard, but the recipes differ in more than one way, so the comparison does not identify Python-specific graph structure as the cause; this zero-shot cell does not establish language-general transfer.

G.4 Model-Scale Sensitivity Study

We extend the Codeaware recipe down to smaller scales (1.5B and 3B QLoRA) to test whether reduced PathSwap sensitivity also appears at smaller model scales. Each model is trained with the same recipe (augmentation 0.5 with 50% path-shuffled examples, code-in-prompt with 30 source lines, LoRA rank 32, two epochs) on the SWE-bench train split (n=1,870 issues), single seed.

The Codeaware cells have smaller PathSwap drops than the stated no-aug comparisons at each tested scale: relative PathSwap drops on Lite are 12.7% (1.5B), 16.8% (3B), 15.2% (7B), 11.0% (14B), and on Verified 11.2% (1.5B), 10.5% (3B), 19.9% (7B), 4.6% (14B). All of these are below the comparison drops (29.7–42.7% relative across the 7B/14B comparisons); the paired no-aug controls for 1.5B and 3B appear below. The relative drops are nonmonotonic across these single-seed cells; they show the pattern at each tested scale without establishing scale invariance or excluding a regime change.

Paired no-aug controls (1.5B and 3B). Codeaware vs. a matched no-aug LoRA, same recipe minus path-augmentation:

Table 51: Single-seed Codeaware scale sensitivity: fractional Recall@1 on Lite/Verified (1.5B–14B).

Model	Lite Normal	Lite PathSwap	Lite drop pp	Verified Normal	Verified PathSwap
1.5B Codeaware	42.00	36.67	−5.33	41.42	36.80
3B Codeaware	45.67	38.00	−7.67	43.60	39.02
7B Codeaware	48.33	41.00	−7.33	44.98	36.02
14B Codeaware (seed 42)	60.33	53.67	−6.66	48.93	46.67

Model	Split	CA Normal	no-aug Normal	CA PathSwap	no-aug PathSwap (binary-hit Δ, p)
1.5B	Lite	42.00	44.67	36.67	21.33 (+15.33, p=2.3×10^{-7})
1.5B	Verified	41.42	44.32	36.80	12.64 (+26.20, p=5.3×10^{-32})
3B	Lite	45.67	48.33	38.00	16.00 (+22.00, p=2.4×10^{-13})
3B	Verified	43.60	46.23	39.02	12.84 (+28.40, p=4.6×10^{-35})

Recall@1 columns are fractional Recall@1; the PathSwap Δ and McNemar p use paired binary top-1 any-hit correctness on matched instances. The tabled binary-any-hit PathSwap differences include 1.5B; the fractional Normal-mode cost is small (-2.6 to -2.9pp, n.s. to marginal)—a single-seed sensitivity trade-off, not a Normal-accuracy gain or mitigation claim.

G.5 Cross-Family Sensitivity Check

To test whether the PathSwap-sensitivity pattern is confined to Qwen, we trained Llama-3.1-8B-Instruct and Mistral-7B Codeaware LoRAs with the same configuration (path augmentation 0.5, 30 code lines, LoRA rank 32, two epochs) on a fixed 5,916-example combined training set selected before evaluation from three source slices (including 1,870 SWE-bench rows; 4,573 unique issue IDs), using seed 42.

Table 52: Single-seed Qwen/Llama/Mistral Codeaware sensitivity: fractional Recall@1 on Lite/Verified; Lite drops are relative.

Model	Lite Normal	Lite PathSwap	Lite drop	Verified Normal	Verified PathSwap
Qwen2.5-7B Codeaware	48.33	41.00	−15.2%	44.98	36.02
Llama-3.1-8B Codeaware	50.00	44.67	−10.7%	46.15	43.35
Mistral-7B Codeaware	46.67	40.00	−14.3%	44.92	39.70

Llama-3.1-8B and Mistral-7B both show the same descriptive PathSwap-sensitivity direction (Lite relative PathSwap drops 10.7% and 14.3%; Verified 6.1% and 11.6%); both point estimates are below the Qwen-7B row. These single-seed cells show that the observed pattern is not confined to the tested Qwen checkpoint family; they do not establish architecture-general behavior.

G.6 Multi-file (6+ files) Scope Limitation

A Codeaware–no-aug PathSwap contrast is detected for the Verified single-file subset ($n = 430$, drop ratio 0.50, $p = 0.0001$), while the tested high-file-count subsets are floor-limited and show no such detected contrast. On SWE-bench Verified 6+-file fixes (n=2), both Codeaware and no-aug produce 0% Recall@1; on SWE-bench full 6+-file fixes (n=51), again both 0% Recall@1; on SWE-bench-Live 6+ (n=116, fallback subset due to repo availability), Recall@1 is 4–5% Normal and 2–3% PathSwap with no detectable contrast (McNemar p=1.0). We report an observed top-1 outcome floor for these high-file-count subsets and candidate pools, not a method-specific or universal structural failure; we recommend evaluating cross-module repair at function-level or with R@K (K$\geq$5) in future work.

Full SWE-bench-Live n=231: the n=116 fallback subset above was extended to all 231 instances after cloning the missing repos. The floor effect persists: Codeaware Normal 4.96%, PathSwap 3.62%; no-aug Normal 5.32%, PathSwap 3.55% (no detectable contrast, paired McNemar p$\approx$1.0).

Table 53: **Scope of the high-file-count checks.** Every 6+-file subset sits at or near the observed top-1 outcome floor, so these rows delimit the multi-file interpretation without estimating a contrast.

Subset	n	Observed result	Reading
Verified single-file	430	drop ratio 0.50, p=0.0001	detected contrast
Verified 6+ files	2	0% Codeaware / 0% no-aug	too small, no top-1 hits
SWE-bench full 6+ files	51	0% Codeaware / 0% no-aug	top-1 floor in this pool
SWE-bench-Live 6+ files	116	4-5% Normal; 2-3% PathSwap	no detected contrast, p=1.0

The floor-limited pattern persists in the full 231-instance set. It does not prove a universal structural ceiling for top-1 file localization or rule out other candidate pools and systems.

G.7 In-domain Java PathSwap Sensitivity Check

To complement the negative zero-shot Java result, we trained an in-domain Java Codeaware LoRA using BeetleBox Java data (n=1,174 paired examples after corpus filtering, drawn from Elasticsearch and Apache DolphinScheduler), single seed. Both Codeaware (aug=0.5) and a no-aug control (aug=0) were trained with the same recipe as our 7B Python Codeaware. On the held-out 100-instance Java PathSwap test set, both in-domain trained models have a 0% PathSwap score, matching the zero-shot floor. Thus the tested in-domain recipe at n_{train}=1,174 supplies no detected Codeaware–no-aug PathSwap contrast; it does not identify whether more data, broader repository coverage, or a different perturbation would change the result.

Bench4BL at n=7,836: we use an in-domain Java corpus (10 Java projects, 10,017 raw reports; 7,836 train / 872 test after filtering). Its 6.7× larger training set is confounded with corpus, project, and test-set changes.

Table 54: BeetleBox Java sensitivity: fractional Recall@1 on $n = 100$.

Cell	Normal Recall@1	PathSwap Recall@1
Java Codeaware (aug=0.5, in-domain trained)	6.37%	0.00%
Java no-aug (aug=0.0, in-domain trained)	9.00%	0.00%
Java zero-shot Python-Codeaware	9.64%	0.00%
Java zero-shot Python-no-aug	11.73%	0.00%

Table 55: Bench4BL Java sensitivity: fractional Recall@1 on $n = 872$.

Cell (Bench4BL, n=872)	Normal Recall@1	PathSwap Recall@1
Codeaware (aug=0.5)	20.70%	3.94%
no-aug (aug=0.0)	19.84%	3.00%

On Bench4BL, the fractional Recall@1 difference under PathSwap is $+0.94$ percentage points. The paired binary any-hit difference is $+1.61$ points (28 versus 14 discordant outcomes among 872 instances; exact McNemar $p = 0.044$). The binary test does not establish significance of the fractional contrast. BeetleBox (n_{train}=1,174) and Bench4BL (n_{train}=7,836) differ in corpus, projects, and splits as well as in training-set size, so the detection difference identifies neither data volume nor another single cause; effects of more or more-diverse Java data or a language-adapted perturbation remain open.

H Training and Inference Interventions

These studies vary training, instructions or inference-time scoring to test changes in localization. Each uses its own comparison and stopping rule. We report successful screens, unresolved contrasts, failed screens, and unbound historical candidates separately, because they provide different evidence about possible mitigation. None of the following studies establishes a link from shared-scorer certification to executed repair.

H.1 TEST-TIME RESCORING

A pre-registered test-time certification-aware re-scoring of the scores already produced by the localization procedure (file score, best-function score, margin, path-only score; tuned on development data with repository-blocked splits) does not meet its pre-registered criterion in either rank- or score-based form (best $+1.0$pp, CI crosses 0), and the tuned weight on the path penalty is zero. Neither is an established fix, so we prescribe a reporting protocol rather than a patch.

H.2 TRAINING-TIME PATHMASK

The path-augmentation in *Codeaware* corrupts paths by within-group *shuffling* (each candidate's path is replaced by another candidate's path, preserving the path-token distribution). A complementary training-time intervention replaces readable path content with stable, structure-preserving opaque PathSwap identifiers. We call this **PathMask**; unlike PathSwap, which is a held-out test-time diagnostic, PathMask is applied during training only. The training-time PathMask of Table 56 is distinct from the pre-registered test-time MaskPath of Table 13.

Setup. We train two 7B SWE-only LoRAs (matched in data and hyperparameters to the Codeaware row of Table 28, single seed=42) varying only the masking fraction: **PathMask-50** masks 50% of training groups; **PathMask-100** masks 100%. Both use code-in-prompt, no path-shuffle augmentation, and the same number of training steps as Codeaware.

Table 56: **PathMask vs. no-augmentation and 50-line Codeaware (7B SWE-only).** Same BM25 top-100 pool and reporting convention as Table 29. Against the matched 30-line Codeaware run of Table 27, PathMask-100 has no detected difference on Verified (paired McNemar p=0.30 Normal, p=0.72 PathSwap; 80% MDE 4.71pp for the fractional PathSwap comparison) and is significantly better than the SWE-only unaugmented recipe on Verified PathSwap ($p \approx 1.0 \times 10^{-5}$). PathMask is exploratory appendix evidence, not a proposed mitigation or equivalence result.

Method	aug / mask	Normal	PathSwap	Δ_{abs}	Δ_{rel} (%)
SWE-bench Lite (n=300)					
SWE-only unaugmented recipe	0.0	53.67	33.33	−20.34	−37.9
Codeaware (50-line)	0.5 shuffle	48.33	41.00	−7.33	−15.2
PathMask-50	0.5 mask	53.33	34.00	−19.33	−36.2
PathMask-100	1.0 mask	45.00	39.33	−5.67	**−12.6**
SWE-bench Verified (n=500)					
SWE-only unaugmented recipe	0.0	46.12	26.43	−19.69	−42.7
Codeaware (50-line)	0.5 shuffle	44.98	36.02	−8.96	−19.9
PathMask-50	0.5 mask	44.60	30.32	−14.28	−32.0
PathMask-100	1.0 mask	41.82	32.74	−9.08	−21.7

Results. Table 56 reports Recall@1 on the same BM25 top-100 pool used for Codeaware. PathMask-100 narrows the no-aug PathSwap relative drop (Lite -12.6% drop vs. no-aug -37.9% and Codeaware -15.2%; Verified -21.7% drop vs. no-aug -42.7% and Codeaware -19.9%), at a -3 to -4pp Normal-path cost relative to Codeaware. Paired McNemar (recall>0 binary): **Against the matched 30-line Codeaware run of Table 27, PathMask-100 has no detected difference on Verified** on either Normal (n_{10}=19, n_{01}=27, p=0.30) or PathSwap (n_{10}=36, n_{01}=32, p=0.72); the fractional-Recall@1 PathSwap delta is $+0.51$pp against the matched 30-line Codeaware run of Table 27, but the 80%-power minimum detectable effect (MDE) is 4.71pp, so this is a no-detected-difference result rather than equivalence. PathMask-100 versus the SWE-only unaugmented recipe on Verified PathSwap: paired-binary $+8.80$pp ($p \approx 1.0 \times 10^{-5}$); aggregate Recall@1 $+6.31$pp. After Holm-Bonferroni correction within the PathMask family, 6/14 tests survive at 0.05, so PathMask should be read as exploratory appendix evidence.

Interpretation. The matched Verified comparison uses n=500. The -3 to -4pp Normal-path cost relative to Codeaware is consistent with masking being a more aggressive intervention than shuffling

(the model sees no real paths, versus 50% real / 50% shuffled). The PathMask-50 underperformance vs. Codeaware on PathSwap (-6.7pp, Verified, paired binary any-hit, p=0.017) is compatible with residual reliance on the unmasked half, but does not identify that mechanism.

H.3 In-Domain Path-Trust Replication

We first test reliability instructions on the paper's own path channel; the separate title-channel pilot follows. The canonical Qwen2.5-14B no-augmentation code-in-prompt checkpoint reranked the same fixed top-100 pools for 500 SWE-bench Verified issues under two prompts. Both preserved the original question, "is this file likely to need modification?", verbatim; the only change was a 13-token instruction calling the displayed path reliable (T) or unreliable (D). Candidate identities, issue text, fixed 50-line code previews, checkpoint, precision, and scoring rule were held fixed.

Table 57: In-domain path-trust result. Values are fractional Recall@1 in percent; Δ is D$-$T with a paired-bootstrap 95% CI (B=10,000, seed 42).

Surface	**Trust (T)**	**Distrust (D)**	**Δ (pp)**
Normal	49.22	49.22	0.00 $[-0.80, 0.80]$
PathSwap	35.32	35.37	$+0.05$ $[-1.10, 1.20]$
PathAlias (seed 42)	32.38	31.68	-0.70 $[-2.20, 0.70]$

The pre-registered primary estimand was shrinkage of the Normal–PathSwap gap: $(N_T - P_T) - (N_D - P_D) = +0.05$pp, CI $[-1.30, 1.45]$; the Normal guardrail passed exactly ($D - T = 0.00$pp). The secondary Normal–PathAlias gap change was -0.70pp, CI $[-2.40, 0.90]$. The pre-registered positive criterion is unmet; the interval supports neither a positive nor negative change. Independent code reproduced all means, intervals and decisions from the 500 raw issue records. Under the pre-registered stopping rule we ran neither a Lite replication nor an alternative prompt. An earlier implementation that accidentally replaced, rather than augmented, the task question was invalidated before emitting per-instance records and is excluded from all analyses.

H.4 Cross-Domain Title-Trust Check

PathAlias establishes that path dependence does not require opaque identifier tokens. This pilot asks the narrower construct question of whether the damage from readable false metadata depends on how much the scorer is instructed to trust the metadata channel. We fixed a BEIR-SciFact top-20 pool for 75 queries before scoring and used a zero-shot Qwen3-8B pointwise reranker. The displayed title took one of three forms: unchanged (Normal); replaced by a form-matched readable title from an unrelated corpus (five mappings selected before scoring); or replaced by punctuation-only dot/dash noise preserving whitespace and word lengths. The passage tokens, query, document identities, pool, and user message were fixed. Prompt T stated that the title was reliable metadata and should be used with the passage; prompt D stated that titles might be unreliable and relevance should be judged from passage content. A disjoint pre-scoring title-use probe passed before confirmatory scoring.

A source-form audit was exact for all 1,204 distinct mappings; after independent 64-token truncation, exactly one unique scored query–document pair was a model-visible exception (the same pair under both prompt templates), leaving 1,499/1,500 exact pairs per template. We retain and disclose this fixed exception; no inference or analyzer rerun was performed.

All 14 cells completed (1,500 query–document rows per cell; 21,000 scores total). For prompt p, A_p is the readable-title drop minus the obvious-noise drop; the interaction is $A_T - A_D$. Table 58 reports the preregistered query-level paired bootstrap (B=10,000, seed 42). The continuous interaction is positive for both metrics; for nDCG@10 it is $+4.971$pp (CI $[1.765, 8.799]$). The A_D intervals include zero, so the evidence supports attenuation under D; literal vanishing versus a smaller residual asymmetry is unresolved at this sample size.

The result is bounded cross-domain convergence, not a claim that document titles and repository paths are interchangeable or that a common learned mechanism has been identified. It identifies prompt-induced title reliance in one SciFact fixed-pool scorer; T and D are distinct prompted scorers, not learned training regimes. It therefore supports the bounded interpretation that, in this fixed SciFact

Table 58: Cross-domain title-trust results (95% paired-bootstrap CIs; nDCG is shown on a percent scale). A_p is readable-title drop minus punctuation-only noise drop under prompt p; interaction is $A_T - A_D$.

Contrast	nDCG@10	Recall@1
T readable drop	+6.194 [2.461, 10.240]	+9.178 [2.667, 16.089]
A_T	+6.966 [3.507, 10.659]	+10.178 [4.400, 16.222]
A_D	+1.995 [−0.011, 4.186]	+3.867 [−0.533, 8.533]
$A_T - A_D$	+4.971 [1.765, 8.799]	+6.311 [0.356, 13.244]

setup, readable unrelated metadata is more disruptive under the title-trust prompt than under the title-distrust prompt, while leaving the PathAlias causal estimand and the paper's code-localization scope unchanged.

H.5 Paired-view continuation pilots

We ask whether the diagnosed dependence can be reduced by training without changing the candidate pool or adding inference-time calls.

We compare three continuations of the same Qwen2.5-7B parent on the same 1,682 training groups, seed 42, optimizer schedule, 106 updates, and fixed final-checkpoint rule. The control continuation minimizes $\mathrm{CE}(\text{real})$. The opaque-view continuation minimizes $\mathrm{CE}(\text{real}) + 0.5\,\mathrm{CE}(\text{hash}) + 0.5\,\mathrm{KL}(\mathrm{stopgrad}(p_{\text{real}}) \| p_{\text{hash}})$, where the second view replaces path components with deterministic opaque identifiers while retaining code and supervision. The readable-alias continuation uses the loss-matched objective $\mathrm{CE}(\text{real}) + 0.5\,\mathrm{CE}(\text{readable}) + 0.5\,\mathrm{KL}(\mathrm{stopgrad}(p_{\text{real}}) \| p_{\text{readable}})$, substituting readable unrelated path aliases for the second view. Evaluation uses the same bf16 pointwise harness, fixed BM25 top-100 pools, and 30-line code previews for all arms.

Table 59: Paired-view consistency pilots. Entries are fractional Recall@1 in percent; each Δ is the indicated paired arm minus the control continuation in percentage points. PathAlias uses the preselected seed-42 alias map; the populations contain 300 Lite and 500 Verified instances.

Split	**View**	**Control continuation**	**Opaque-view continuation**	Δ_H	**Readable-alias continuation**	Δ_R
Lite	Normal	53.67	52.00	−1.67	53.00	−0.67
	PathSwap	40.33	43.33	+3.00	40.33	0.00
	PathAlias (seed 42)	38.00	41.00	+3.00	43.00	+5.00
Verified	Normal	48.80	47.53	−1.27	48.28	−0.52
	PathSwap	37.60	39.65	+2.05	35.00	−2.60
	PathAlias (seed 42)	38.90	42.03	+3.13	40.68	+1.78

The original opaque-view continuation screen required Lite PathSwap $\Delta_H \geq +3$pp, Lite Normal $\Delta_H \geq -2$pp, Verified PathSwap $\Delta_H \geq 0$pp, and Verified Normal $\Delta_H \geq -2$pp; all four pass. On the later preselected seed-42 PathAlias cells, its point estimates are also positive: $+3.00$pp on Lite (95% CI $[0.00, 6.33]$) and $+3.13$pp on Verified (CI $[1.17, 5.13]$).

The readable-alias continuation screen applied the same Normal and PathSwap rules and additionally required PathAlias $\Delta_R \geq +3$pp on Lite and ≥ 0pp on Verified. Five of six checks pass. Its matched PathAlias deltas are $+5.00$pp (CI $[1.67, 8.67]$) and $+1.78$pp (CI $[-0.70, 4.23]$), and both Normal guardrails hold; however, Verified PathSwap is -2.60pp (CI $[-4.90, -0.40]$), so the preregistered criterion is not met and no ten-seed extension is run. The primary analysis and a separate implementation independently recomputed all means, paired deltas, intervals, and the decision from raw per-instance files and agreed exactly. Relative to the control continuation, the opaque-view continuation is positive on the preselected PathAlias map while the readable-alias continuation does not meet the Verified PathSwap criterion; this is a bounded consistency finding, not mechanism identification or general mitigation: one training seed and one alias seed do not quantify training-seed or alias-seed uncertainty.

H.6 HISTORICAL TRAINING-FREE FUSION RECORDS

Two-score rule recorded in the historical manuscript. Let $s_{\mathrm{real}}(f)$ and $s_{\mathrm{hash}}(f)$ denote a candidate's scores under the real-path and opaque-identifier-path views, respectively. The historical rule is

$$u_\alpha(f) = \alpha s_{\mathrm{real}}(f) + (1-\alpha)\big[s_{\mathrm{real}}(f) - s_{\mathrm{hash}}(f)\big]. \tag{5}$$

Thus $\alpha = 1$ returns the real-path score, whereas $\alpha = 0$ returns the real-minus-opaque score difference. This difference is an intervention-dependent scoring quantity, not an identified code-only component. It is also distinct from the separately trained code-residual scorer and its fusion in Table 33.

Record conflict and evidential status. For SWE-bench Lite, the historical text lists the grid $\{0, 0.25, 0.5, 0.75, 1\}$ but also names $\alpha = 0.6$ as the optimum. Since 0.6 is absent from that grid, the two statements cannot describe a selection restricted to the listed grid. No executed-grid/run binding resolves the conflict. We retain the exact rule as a documented historical candidate, not as a reproduced procedure or a validated improvement; we neither infer an additional search nor substitute another optimum. The unbound performance values remain in the project archive and are not used as evidence of a gain. This is a record limitation, not a NULL result from a bound performance contrast.

Separate rank-fusion record. The retained Reciprocal Rank Fusion record for Codeaware and SweRankEmbed-Small selects pure Codeaware ($\alpha = 1.0$, 52.33%) and reports 34.00% for the dense retriever alone on its stated pool. It gives no gain over pure Codeaware. It leaves the two-score run binding unresolved and establishes neither general mitigation nor limits on other inference-time methods.

I NATIVE SWERANK REPRODUCTION

This appendix records the native SweRank procedure, the unresolved absolute reproduction gap, and the associated function-body controls. The separate path-document encoder probe is in Appendix G.1.

Retrieval-and-reranking reproduction setup. SweRank's retrieval-and-reranking procedure—SweRankEmbed-Large (7B) retrieves a top-100 function pool from the full-repository function index, then SweRankLLM-Small (7B) listwise-reranks—is reported by Reddy et al. (2026) at **78.10%** strict file Acc@1 on SWE-bench Lite. We attempted to reproduce this number end-to-end using the official retrieval and reranking launchers under the paper-pinned environment (vLLM 0.8.5.post1, PyTorch 2.6.0, Python 3.11, and Transformers 4.51.3; verified against the documented dependency specification and official launchers). Our reproduction yields **20.8%** strict file Acc@1 after reranking, a gap of -57.30pp.

We tracked this to low retriever-side recall: strict file-level Acc@100 from SweRankEmbed-Large alone (with the official instruction prefix, released model-card weights, and released SWE-bench Lite function corpus and relevance labels) is only **37.23%**, so even an oracle reranker cannot exceed that ceiling. The gap is not explained by sequence-length truncation (queries p99 $\approx$ 676 tokens, corpus documents p99 $\approx$ 863 tokens; both fit in the default 1024-token budget) nor by an obvious launcher divergence. We were unable to identify the root cause within our compute budget.

We report only the controlled same-pool reference comparison to SweRankLLM-Small (BM25 top-100 file pool, partial Recall@1), where 14B Codeaware is higher than SweRankLLM-Small by $+12.4$pp under our same-pool diagnostic protocol on Lite (Table 28). Normal and PathSwap share the same protocol within each reproduced row, so we report only the within-row relative drop; the absolute reproduction gap remains unresolved.

I.1 ATTEMPT FROM PUBLICLY RELEASED MATERIALS

We attempted to reproduce SweRank using the public GitHub code, the README-linked Google Drive corpus, the public Hugging Face SweRankEmbed-Large and SweRankLLM-Small checkpoints, and the documented retrieval-prefix option. On the official SWE-bench Lite filtered evaluation set (N=274), our reproduced SweRankEmbed-Large retriever obtained 1.46% file Acc@1 and 37.96% file Acc@100 in this separate attempt using the publicly released code, models, and corpus (the pinned-environment run above gives 37.23% Acc@100), compared with the paper's reported 72.63% retriever

Acc@1 and 78.10% end-to-end Acc@1. We verified that the README-linked corpus matches our copy and found no public Docker image, alternate evaluation corpus, or documented preprocessing step that explains the gap. Because this absolute-number discrepancy remains unresolved from publicly released materials, we do not use the unsuccessful absolute reproduction as evidence against SweRank; instead, we report SweRank only as a same-pool path-document diagnostic and mark the reproduction gap as unexplained.

SweRank procedure under PathSwap. Because the absolute reproduction remains unresolved, we treat the same-protocol PathSwap comparison as descriptive only: we run the same Embed-Large $\rightarrow$ LLM-Small procedure on a PathSwap-perturbed copy of the SweRank-distributed corpus, where file-path components use opaque identifiers while function/class/method identifiers, signatures, and bodies are preserved verbatim. On the same n=274 instances, the procedure's strict file Acc@1 moves from 20.8% (normal paths) to 21.17% (PathSwap) ($\Delta_{\text{rel}} = +1.8\%$, no significant change). This observed change is not statistically detectable and establishes neither path insensitivity for this unreproduced procedure nor a general property of function-level rerankers. Residual body-token leakage (imports, docstrings, string literals, repository-specific identifiers) is one possible explanation for this cell, but the diagnostic below is directional and non-significant at n=274 and therefore does not establish that explanation.

Function-level diagnostic. We additionally trained a function-level Codeaware LoRA (Qwen2.5-7B QLoRA on 3,711 SWE-bench-train + GREPO function-pair instances; same recipe as our file-level LoRAs except prompts include the function body rather than first-50 file lines), and evaluated it on the SweRank function pool under PathSwap (path components use opaque identifiers; class/method names initially remain; cf. §I for limitations). To distinguish displayed-path sensitivity from incidental body leakage—function bodies still mention repository names in import statements, docstrings, comments, and string literals—we additionally replace all repository-specific identifier tokens (`astropy`, `django`, ..., including composite identifiers) consistently across queries and corpus. The 2×2 decomposition (path-only $\rightarrow$ opaque-identifier condition; body-uncleaned $\rightarrow$ body-cleaned) on n=274 SweRank-Lite instances yields: combined effect −3.65pp (−18.9% relative), of which body-cleaning alone accounts for −1.09pp (distribution shift, p=0.38 McNemar) and the opaque-identifier condition on cleaned bodies accounts for −2.55pp (p=0.44). The point estimate is directionally consistent with the file-level Codeaware drop (−15–17% relative on the BM25 file pool), but the cumulative Normal$\rightarrow$full repository-identifier cleaning comparison is *not* statistically significant at n=274 (p=0.26); the final opaque-identifier step on cleaned bodies is also non-significant (p=0.44). The full 5-step cleaning progression (monotonic 19.34 $\rightarrow$ 15.69) is reported in Appendix I.2 with its caveats.

I.2 Function-Level Cleaning Progression

Purpose. To motivate the full repository-identifier cleaning control used in the function-level diagnostic summarized in Appendix I, we report the full 5-step monotonic cleaning progression of our function-level Codeaware LoRA on the released SWE-bench Lite function corpus (n=274). Each step replaces one additional family of repository-specific tokens with opaque identifiers; the trend is monotonic.

Strong caveat. This analysis sits on top of a SweRank reproduction that is 57pp below the published absolute number (20.8% vs. 78.10% strict file Acc@1; cf. Section I). Absolute numbers in this table are therefore *not* comparable to SweRank's published numbers, and none of the individual pairwise differences is statistically significant at n=274 (all McNemar exact two-sided p>0.25). We report the *relative monotonic sequence* only as a descriptive pattern, not as inferential evidence.

Reading. Deeper cleaning monotonically lowers Recall@1 (19.34 $\rightarrow$ 15.69, cumulative −18.9% relative), a descriptive pattern consistent with the hypothesis that residual body tokens (imports, docstrings, class/method names that happen to resemble the opaque path identifier) may mask some path sensitivity. It does not establish that mechanism: individual differences are not significant, and the underlying SweRank procedure has an unresolved 57pp reproduction gap.

Table 60: Function-level cleaning progression on the SweRank-distributed function pool (SWE-bench Lite, n=274), with our function-level Codeaware LoRA as the reranker. Each row additionally replaces one more family of repository-identifier tokens. Absolute numbers sit on an unreproduced SweRank procedure (Section I); only the monotonic trend should be read.

Condition	Additional cleaning	Recall@1	McN. p vs. normal
Normal	none	19.34	reference
Path identifiers only	directories + filenames	17.88	0.70
Plus definition-line identifiers	class / method identifiers	16.79	0.45
Plus imports/docstrings/ module paths	import / docstring / module-path strings	16.06	0.32
Plus all repository-token substrings	pyfits + all repository-token substrings	15.69	0.26

J CodeGRIP: Implementation and Reference Results

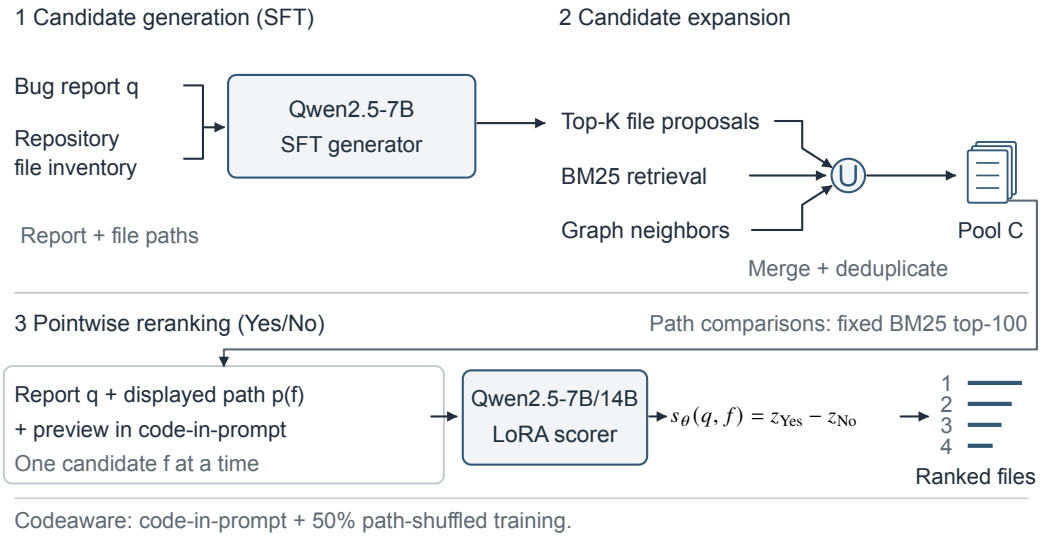


Figure 5: **The CodeGRIP localization workflow.** An SFT generator proposes file paths; retrieval and graph neighbors supply additional candidates. After merging and deduplication, a LoRA scorer ranks files by their Yes/No logit difference, optionally using a code preview. Codeaware is the code-in-prompt recipe with 50% path-shuffled training examples. The primary path comparisons instead fix a BM25 top-100 pool and vary only displayed paths.

Group training objective. Pointwise inference is distinct from learning: the reranker is trained on groups of positive and negative files with a listwise ranking loss. The historical GREPO configuration uses one positive and 16 negatives, mixed from BM25-hard, graph-hard and random files in a 50/25/25 ratio. The documented SWE-bench adaptation configuration instead uses 32 negatives, 75% BM25-hard and 25% random, without graph negatives. Appendix J gives model settings and data separation for each recipe.

For a training group with positive file f_0 and negatives $f_1, \ldots, f_m$, the implementation uses the group-softmax loss

$$\ell(q; f_0, \ldots, f_m) = -\log \frac{\exp s_\theta(q, f_0)}{\sum_{j=0}^{m} \exp s_\theta(q, f_j)}. \tag{6}$$

Each report–candidate pair is processed separately; group normalization couples scores during training, whereas inference again scores files individually. The loss uses no temperature. Code-present variants retain the original candidate source preview when the training path is shuffled;

augmentation alters the displayed cue while the loss and original target remain unchanged. The previously listed GREPO and SWE-bench recipes therefore remain distinct training configurations.

Table 61: CodeGRIP reference operating points. Panel A reports retained GREPO configurations ($n = 1{,}704$), not a causal decomposition; Table 71 retains the full rows. Panel B is a separate candidate-count comparison at approximately equal oracle recall, not runtime.

A. GREPO operating points ($n = 1{,}704$) Configuration	Recall@1 (%)	Pool size
BM25, no reranker	10.00	500
Reranker on BM25	28.45	500
Candidate generation only	21.33	~ 32
Generation + graph/BM25 merge	22.51	~ 163
Complete generation–expansion–reranking	29.38	~ 163
B. Candidate budget at $\sim 90.7\%$ oracle recall		
Graph-expanded	~ 208 candidates	
BM25-only	~ 393 candidates	

Illustrative workflow example retained from the original figure. The original diagram uses the report "Login fails with 401 after token refresh ..." and a file inventory containing `auth/login.py`, `auth/refresh.py`, `auth/session.py`, `db/`, and `utils/`. Its example proposal list contains `auth/login.py`, `auth/refresh.py`, `db/connection.py`, and `utils/helpers.py`. The indicated $K = 50$ is an example, not a universal candidate-generation budget. The BM25 top-500 retrieval source, the roughly 163-file GREPO merged pool, and the depicted ranked top-100 output have different roles; none changes the fixed BM25 top-100 evaluation pool in the primary SWE-bench path comparisons. The original output-score sequence $0.87, 0.62, 0.21, -0.03, -0.41$ is illustrative, not an experimental result or a binding between these scores and the example paths.

The code example is likewise illustrative:

```
def login(user, pwd):
    token = issue_token()
    if not token.valid():
        return 401
    ...
```

The SFT input is the flat file-path inventory, not all repository source lines suggested by the original file-tree illustration. Preview lengths remain recipe-specific: 30 audited lines and some 50-line legacy evaluation rows are separately documented in the training and evaluation records.

Figure 5 introduces the CodeGRIP workflow. This appendix specifies candidate construction, scoring, training, and data separation, then reports its reference operating points and sensitivity to budgets, seeds, checkpoints, and candidate pools. The implementation and its bounded evaluation results support the audit; they are not a matched ranking of competitive systems. Protocol-specific settings and negative results are retained alongside the corresponding positive observations.

J.1 Candidate construction and scoring

J.1.1 Prompt Templates

file reranking: Cross-encoder reranking. The reranker uses a binary relevance prompt, scoring each candidate independently:

```
Given the bug report, is this file likely to need
modification?
Bug Report: {issue_text}
File: {candidate_path}
Answer:
```

The score is $s = z_{\text{Yes}} - z_{\text{No}}$, where z_{Yes} and z_{No} are the logits for the "Yes" and "No" tokens at the generation position after "Answer:". This yields a pointwise scorer without changing the LLM architecture.

Candidate generation (SFT). The SFT model receives the bug report and a flat repository file-path inventory (no co-change or import graph in the prompt), and generates file paths:

```
You are analyzing a bug report for the Python repository
"{repo}".
The repository contains the following Python files:
{file_list}
Bug Report: {issue_text}
From the file list above, identify which Python files need
to be modified to fix this bug. Output file paths only, one
per line, most relevant first.
```

J.1.2 BM25 Retrieval Details

GREPO evaluation. BM25 retrieval indexes all Python files per repository, using file paths and content. The top-500 candidates form the initial retrieval pool. Candidates are merged with candidate expansion graph-expanded files via score-weighted interleaving, yielding $\sim$163 candidates on average.

SWE-bench Lite retrieval. We apply several benchmark-specific retrieval choices over standard BM25 (historical label: "benchmark-specific BM25 recipe"):

1. **Test exclusion**: Files identified as tests are removed from the candidate pool; H@1 increases by $\sim$9pp absolute in this benchmark-specific recipe.
2. **Multi-granularity indexing**: We create two parallel indexes—function-level and chunk-level (500-token chunks)—and fuse rankings via reciprocal rank fusion (RRF) with $k = 60$.
3. **Stemming**: Snowball stemmer applied to both queries and documents.
4. **Post-processing**: Results mapped back to file-level, deduplicated by first occurrence.

An oracle drawing on all retrieval pools (path, content, function, chunk, and their combinations) reaches H@1 = 54%; that figure is a multi-pool envelope, not a single-pool system score or a fusion-method claim.

J.2 Training configurations and data separation

J.2.1 Training Hyperparameters

Table 62 lists the candidate generation and file reranking hyperparameters.

Table 62: Training hyperparameters for candidate generation SFT and file reranking.

Hyperparameter	candidate generation (SFT)	file reranking (Reranker)
Base model	Qwen2.5-7B	Qwen2.5-7B/14B
LoRA rank / α / dropout	32 / 64 / 0.05	
LoRA targets	$\mathbf{W}_{q,k,v,o}$, $\mathbf{W}_{\text{gate,up,down}}$	
Learning rate	2×10^{-5}	5×10^{-5}
Scheduler	cosine (5% warmup)	
Epochs	3	2
Train batch / grad accum	not applicable	1 / 16
Code preview lines	not applicable	30 audited; 50 legacy s42 eval
Max seq length	2048	768 (7B/main); 512 (14B s43–44 OOM)
Negatives / positive (GREPO)	not applicable	16
Optimizer	AdamW ($\beta_1 = 0.9, \beta_2 = 0.999$)	
Gradient clipping	1.0	

J.2.2 Data Separation and Graph Provenance

Co-change edges are computed exclusively from the GREPO training split; no test-split data is used. Import edges are derived from static AST analysis of repository snapshots, independent of any PR timestamp. The BM25 test pools use the repository file tree at the pre-fix repository snapshot, preventing post-fix file-tree information in those pools. The train/test split in GREPO is by issue ID, not by time; we verify that no issue IDs overlap between train (7,883) and test (2,175) splits. All three index builders explicitly restrict input to the training split. 62.2% of test examples have a ground-truth file in the training co-change graph; this is not issue-ID overlap, but it is a structural prior available to the model.

J.2.3 SWE-bench Domain Adaptation

Training data. We construct adaptation data from SWE-bench Full (2,294 instances), excluding all 300 SWE-bench Lite test instances. Filtering for BM25-candidate availability and Python file changes leaves 1,870 examples from the same 12 repositories as Lite. The adaptation holds out issues, not repositories.

Training setup. The adapted model initializes from the GREPO-trained reranker checkpoint (file reranking) and fine-tunes for 2 epochs with lr = 1×10^{-5}, using the same listwise loss with 32 negatives (75% BM25-hard, 25% random; no graph negatives, since SWE-bench lacks co-change history for these repositories).

Training-data counts. CodeGRIP uses 1,870 target-domain adaptation examples—$\sim$36$\times$ fewer than SweRankEmbed-Large's 67K. Including GREPO pre-training (7,883 examples), total supervised data is $\sim$9.8K ($\sim$6.9$\times$ less than SweRank's 67K). These are training-set size comparisons, not sample-efficiency estimates, because the systems use different protocols. The GREPO-trained initialization is reused here, but these cells do not isolate its contribution. Without adaptation, the GREPO-trained model reports H@1 = 51.5% (274 matched) via hybrid fusion; because training source, scorer, and retrieval are bundled, this is a cross-domain operating point rather than an isolated effect of graph-structured training.

J.3 Reference operating points

CodeGRIP serves as an experimental testbed, not a competitive system. Cross-system numbers (e.g., SweRank) are included as reference points, but they are not directly comparable: the training data differs (7,883 vs. 67K examples), the evaluation protocols differ, and our retrieval-expansion-reranking procedure uses graph signals at inference. The most internally consistent comparisons are the controlled ablations (Appendices E and F.3), where all variants share the same base model and candidate pool.

These are benchmark-specific reference snapshots rather than a matched system ranking. Their data, metric, checkpoint and release boundaries are stated in each table.

Table 63: Audit-testbed operating points on GREPO (1,704 test examples). CodeGRIP's R@5 is withheld because its binding to the dense-best checkpoint is unresolved. Cross-system numbers are reference points, not direct comparisons; CodeGRIP is not presented as a SOTA entry (Appendix J.3).

Method	Recall@1	R@5
BM25 (lexical, top-500)	10.00	23.38
SweRank (7B, 67K data)	18.18	34.83
CodeGRIP (complete generation–expansion–reranking procedure)	29.38	—

The CodeGRIP Recall@1 operating point is retained. No R@5 value or paired significance claim is reported here without matched checkpoint records; see Table 71.

The adapted testbed reaches 54.67% H@1 with a 7B model using 1,870 target-domain examples ($\sim$36$\times$ fewer than SweRankEmbed-Large's 67K), but differing data and protocols make this a cross-benchmark diagnostic operating point and do not identify data efficiency or system superiority.

Table 64: Transfer to SWE-bench Lite (Jimenez et al., 2024) (all 300 instances). "Adapted" fine-tunes on 1,870 SWE-bench Full examples (no Lite test overlap). Cross-system rows are reference operating points under different data and protocols; CodeGRIP is the audit testbed, not a competitive-system claim (Appendix J.3). The benchmark-specific BM25 recipe is described in Appendix J.1.2.

Method	**Data**	**H@1 (%)**
Benchmark-specific BM25 recipe	not reported	41.30
LocAgent (Claude-3.5)	not reported	46.30
CodeGRIP (GREPO-trained, raw neural)	7,883	46.30
CodeGRIP (adapted, complete generation–expansion–reranking procedure)	1,870	54.67
CodeRankEmbed (137M)	67K	54.30
SweRankEmbed-Large (7B, 67K)	67K	56.30

Table 65: Cross-setting operating points under different populations and metric families; rows are descriptive and not directly rank-comparable.

Setting	**N**	**Recall@1 / Hit@1**	**R@5**	**Shift type**
GREPO (seen repos)	1,144	29.81	52.18	In-domain
GREPO (unseen repos)	560	20.77	46.70	New repos, same lang
BeetleBox Java	740	14.43	29.89	Cross-language
SWE-bench Lite (zero-shot)	300	48.0–51.3	not reported	Cross-benchmark

SWE-bench Hit@1 range across three GREPO-trained model settings (zero-shot, BM25-top-500 pool); GREPO/BeetleBox use fractional Recall@1.

J.4 Scale and candidate construction

J.4.1 Scale and candidate budgets

Table 66: Model-scale Recall@1; all cells omit candidate generation SFT.

Size	**Graph-hard**	**BM25-only**	**Δ**
0.5B	20.02	20.12	−0.10
1.5B	21.54	20.66	+0.88
3B	23.37	23.18	+0.19
7B	24.54	23.91	+0.63

Scale and expansion. Both improve monotonically with scale; graph-hard differences are non-monotonic; capacity alone is unsupported, and removing SFT initialization does not isolate recipe confounding.

J.4.2 Expansion Policies and Pipeline Stages

These panels compare expansion strata, edge choices, learned budgets and pipeline stages under the stated settings.

Expansion interpretation. Co-change strata show a larger observed gain than import strata (+11.9pp versus +6.7pp on expansion-only examples); edge type is not randomized, so this is not causal evidence. Graph expansion reaches 90.7% oracle recall at roughly 208 candidates versus roughly 393 for BM25. To decompose the contribution of each edge type, we evaluate the graph-hard reranker on pools expanded from *BM25 seeds* (not SFT seeds) using different edge types. BM25-seeded expansion scores 19.96–20.44, below the 28.45 BM25-only baseline; this joint seed/pool change neither isolates edge type nor establishes a dependence on high-quality seeds. Across $\tau_{cc} \in \{0.01, 0.03, 0.05, 0.10, 0.20\}$ and co-change/import neighbor budgets $\{3, 5, 10, 15, 20\}$, oracle recall varies by < 0.1pp and < 2pp, respectively; all 15 settings improve over the BM25-only baseline within the tested grid only. The classifier is trained on GREPO train examples and evaluated on test. The joint-feature learned policy is +5.9–7.5pp above random at every tested budget, but

Table 67: Candidate budget at approximately 90.7% oracle recall.

Retriever	**Budget**	**Interpretation**
Graph-expanded	∼208	Smaller candidate set
BM25-only	∼393	Nearly 1.9× larger

Table 68: Stratified Recall@1; Δ is the complete generation–expansion–reranking procedure minus BM25-only (pp).

Stratification	Stratum	N	Recall@1	Δ
GT files	single-file	668	43.4	+15.0
	multi-file	1036	16.4	+4.6
Repo size	small (< 200)	180	27.2	+2.9
	medium (200–500)	394	29.3	+10.4
	large (> 500)	1130	26.2	+9.0
Expansion	GT only in expanded	130	20.8	+15.8
	GT in both pools	1199	34.6	+9.9

Table 69: BM25-seeded edge-type ablation; BM25-only is 28.45 and the SFT-seeded complete generation–expansion–reranking procedure is 29.38.

Edge type	Recall@1
Co-change only	20.15
Import only	19.96
Both	20.44

Table 70: Learned expansion oracle recall at matched neighbors-per-seed budget K.

Policy	$K = 3$	$K = 5$	$K = 10$	$K = 15$	$K = 20$
Random	29.1	30.2	32.1	34.3	36.4
Classifier	35.0	36.6	39.6	41.7	43.1
Δ	+5.9	+6.4	+7.5	+7.3	+6.6

Full unfiltered expansion: 52.2% oracle recall; average pool 71 files.

Table 71: Dense-best R@1 is retained. Historical R@5/R@10 are withheld because their checkpoint binding is unresolved; R@20=63.48 likewise has no binding in the available records and is withheld. The archived row is not a verified single-checkpoint recall vector. Original displayed cells remain in the scientific-display history index; no new evaluation was run.

Configuration	R@1	R@5	R@10	R@20	Pool
BM25, no reranker	10.00	23.38	29.21	38.54	500
Reranker on BM25	28.45	38.74	46.41	53.40	500
candidate generation only	21.33	29.48	30.86	31.62	∼ 32
+ graph/BM25 merge	22.51	36.64	42.64	49.69	∼ 163
+ reranker	29.38	—	—	—	∼ 163

its co-change, import, path-distance, BM25-rank, and seed-link features do not isolate a graph-only cause; maximum and summed co-change weights have importances 0.44 and 0.35.

Repository and Pipeline Outcome Counts Graph expansion improves Recall@1 in 58% of repositories in this analysis (Wilcoxon $p = 0.0005$, 69 repositories); the split does not identify a causal structural mechanism.

Win/loss analysis. Across 1,704 test examples, the complete generation–expansion–reranking procedure wins 485 and loses 141 compared to the BM25-only baseline (3.4× win ratio). 51%

of wins are reranking improvements, 40% are coverage wins. These categories give only a partial decomposition.

J.5 Seed, checkpoint, and candidate-pool sensitivity

J.5.1 Seed and checkpoint variation

Five-Seed Training Comparison Across five random seeds (42, 1, 2, 3, 4), graph-hard negatives show a mean Recall@1 advantage of +0.74pp (25.01 ± 1.17 vs. 24.27 ± 0.39), but this advantage is *not* statistically significant ($p = 0.33$, paired t-test).

Table 72: Seed robustness (mean ± std).

Method	Recall@1	R@5
Path-only (3)	28.92(56)	not reported
Graph-hard (5)	25.01(117)	45.56(202)
BM25-only	24.27(39)	44.52(58)

Table 73: Checkpoint sensitivity; the dense grid has 23 checkpoints. Across that grid, the graph–hybrid gap has mean 0.49pp and maximum 1.12pp.

Strategy	Graph	Hybrid	BM25	Graph–hybrid gap
Sparse, checkpoints saved every 200 training steps	27.01	19.29	20.08	7.72
Dense best, checkpoints saved every 10 training steps	29.38	29.38	28.45	0.00

J.5.2 Cross-Pool and Training Diagnostics

These panels compare operating points across candidate pools, architectures, checkpoints and training perturbations.

Table 74: Cross-pool scorer architectures; sparse unless marked dense.

Scorer	Graph oracle	Graph R@1	Hybrid oracle	Hybrid R@1	Δ oracle	Δ R@1
Cross-encoder sparse	90.7	27.01	92.1	19.29	+1.4	−7.7
Cross-encoder dense best	90.7	29.38	92.1	29.38	+1.4	0.0
E5-large zero-shot	90.7	10.12	92.1	9.43	+1.4	−0.7
SweRankEmbed zero-shot	90.7	18.18	92.1	18.16	+1.4	−0.02

Table 75: Scale-stratified cross-pool point estimates.

Model	Graph R@1	Hybrid R@1	Δ	Graph R@5	Hybrid R@5	Δ
0.5B	20.02	24.12	+4.1	38.69	46.27	+7.6
1.5B	21.54	26.28	+4.7	41.16	48.74	+7.6
3B	23.37	28.82	+5.5	43.58	51.70	+8.1
7B sparse	27.01	20.00	−7.0	49.17	38.12	−11.1
7B dense best	29.38	29.38	0.0	not evaluated	not evaluated	not evaluated

Cross-pool conclusions. At a single checkpoint, with checkpoints saved every 200 training steps, we previously observed an apparent "oracle fallacy": a hybrid retriever (BM25 + E5-large) achieves +1.4pp higher oracle recall than the graph-expanded pool (92.1% vs. 90.7%), yet the reranker's

Table 76: Sparse-checkpoint score gaps.

	Gap mean		Gap< 0		GT rank	
Model	Graph	Hybrid	Graph	Hybrid	Graph	Hybrid
0.5B	-0.32	-0.15	56.9	51.9	7.1	5.5
1.5B	-0.28	-0.01	54.0	48.7	6.8	5.0
3B	$+0.05$	$+0.26$	50.9	43.4	6.2	4.6
7B	$+0.09$	-0.23	44.8	54.7	5.2	8.1

Table 77: Delexicalization diagnostics.

Method	Graph	Hybrid	Gap
Sparse baseline	27.01	19.29	7.72
Dense baseline	29.38	29.38	0.00
50% overlap-token replacement	29.46	29.61	-0.15
Counterfactual-negative training	28.60	29.52	-0.92

Recall@1 appeared to drop substantially. Here, this retrieval-pool hybrid is distinct from the score-gap switching strategy in Appendix J.6. The sparse-checkpoint Recall@1 gap is about 7pp; dense best-checkpoint evaluation gives 29.38% on both pools, with mean 0.49pp and maximum 1.12pp across 23 checkpoints. Save frequency and selection intensity are confounded, so this does not support checkpoint-invariant coupling. At the sparse checkpoint, among 1,361 shared-top-50 examples, GT mean scores are 4.24 versus 3.46 (-0.78 logit), and graph ranks GT higher in 61.7% versus 30.6%. Architecture cells do not establish architecture-wide coupling. At 0.5B–3B the hybrid-minus-graph estimates are $+4.1$ to $+5.5$pp, while 7B is -7.0pp sparse and < 1.2pp dense; this does not identify capacity. The two sparse hybrid logs differ and are neither reconciled nor pooled. For 7B, distractor-above-GT rises from 44.8% to 54.7%; smaller scales lack dense controls. On SWE-bench, BM25 training uses 1,870 examples; native/hybrid oracle recall is 86.7%/91.7% ($+5.0$pp), both yield Recall@1 39.67%, and dense-only candidates never enter top 200. Candidate competitiveness is therefore only a possible boundary, neither necessary nor sufficient.

Training and headroom conclusions. These are training diagnostics, not mitigation claims. The 50% overlap-token replacement condition replaces overlapping path tokens with stable opaque identifiers on 50% of training examples and scores 29.46%/29.61%, about $+0.2$pp from the 29.38% dense baseline without reported uncertainty. Counterfactual-negative training (25% same-directory, 25% maximal path-token overlap) achieves Recall@1 = 28.60% / 29.52%. Its combination with delexicalization scores 28.33%/29.16%, which does not establish mechanistic redundancy. Graph-expansion changes are largest for single-file bugs ($+15.0$pp) and medium repositories ($+10.4$pp), and 7.6% of examples place GT only in the expanded pool. On SWE-bench (12 repositories, no in-domain graph data), expansion has no effect (Recall@1=18.0 with or without), showing no observed gain in this 12-repository SWE-bench setting; the comparison does not identify why. Two-hop iterative expansion from the top-10 reranked files finds 30.9% of GT files missing from the round-1 pool, raising GT coverage from 51.2% to 66.3%. The scorer is graph-free at scoring time; precomputed graph indexes cost 1.4 CPU seconds, under 0.02% of inference time. All 15 sensitivity settings are above baseline within the tested sweep (Appendix J.4.2).

J.6 Score-gap fallback

The hybrid strategy for SWE-bench combines neural reranking with BM25 fallback based on model confidence:

1. **Score gap**: For each example, compute $\Delta = s_1 - s_2$, the difference between the top-1 and top-2 neural scores.
2. **Confidence switching**: If $\Delta \geq \tau$ (threshold), keep the neural top-1 prediction. Otherwise, use RRF of neural and BM25 rankings.

3. **Non-source-file fallback**: If the neural top-1 is a likely documentation-configuration or migration file, fall back to BM25 regardless of confidence.

Table 78: Score-gap accuracy and coverage for the adapted model (SWE-bench Lite, 274 matched).

Gap threshold τ	N (confident)	Accuracy (%)	Coverage (%)
≥ 1.0	160	68.1	58.4
≥ 1.5	130	73.8	47.4
≥ 2.0	104	79.8	38.0
≥ 3.0	70	84.3	25.5

Higher thresholds yield higher accuracy but lower coverage. For reported hybrid runs, we use $\tau = 1.5$ for the adapted model (73.8% accuracy, 47.4% coverage) vs. $\tau = 2.0$ for the GREPO-trained model. These operating points are not claimed as validation-selected optima or as a formal calibration comparison.

J.7 Repository and difficulty strata

J.7.1 SWE-bench Per-Repository Breakdown

Table 79: Per-repository Hit@1 on SWE-bench Lite (all 300 instances). BM25 = benchmark-specific BM25 recipe (§J.1.2); Neural = unmodified adapted-model predictions at 1024-token context.

Repository	N	BM25 Hit@1 (%)	Neural Hit@1 (%)
django/django	114	39.5	53.5
sympy/sympy	77	41.6	51.9
matplotlib/matplotlib	23	52.2	43.5
scikit-learn/scikit-learn	23	26.1	47.8
pytest-dev/pytest	17	29.4	29.4
sphinx-doc/sphinx	16	68.8	43.8
astropy/astropy	6	83.3	83.3
psf/requests	6	50.0	33.3
pylint-dev/pylint	6	16.7	50.0
pydata/xarray	5	0.0	60.0
mwaskom/seaborn	4	50.0	75.0
pallets/flask	3	66.7	66.7
Weighted sum (raw)	300	**41.3**	**50.67**

The neural reranker has higher point estimates than BM25 on approximately half of repositories, including django (+14.0pp), scikit-learn (+21.7pp), xarray (+60.0pp), and pylint (+33.3pp). BM25 has higher point estimates on sphinx and matplotlib, where the neural margins are -25.0pp and -8.7pp respectively. This per-repository table is descriptive; several denominators are small, and it does not identify naming patterns as the cause. The hybrid rule uses BM25 fallback when the neural score gap is low, but these rows alone do not isolate its incremental benefit.

J.7.2 Hardness Analysis

Table 80: Recall@1 stratified by difficulty (reranker vs. BM25 pool ordering; Δ is in percentage points).

Stratum	N	Reranker (%)	BM25 (%)	Δ (pp)
By BM25 rank of first GT file				
Rank 1 (easy)	645	47.48	59.46	−11.98
Rank 2–5	250	21.90	0.00	21.90
Rank 6–20	232	16.70	0.00	16.70
Rank 20+ (hard)	577	10.48	0.00	10.48
By repository size				
Large (1000+ files)	729	21.12	14.70	6.42

For strata with BM25 rank ≥ 2, the reported +10–22pp gaps partly follow by construction because the baseline's first-ranked file cannot be gold; the large-repository gap is +6.42pp. When BM25 rank is 1, the reranker's gap is -12pp. These rank-conditional gaps do not establish that structural patterns cause the differences.

K Related Work

K.1 Localization granularity and outcome criteria

Related work. Bug localization has progressed from IR methods such as BM25 (Robertson & Zaragoza, 2009) and learned text–code similarity (Huo et al., 2016; Lam et al., 2017) to LLM-based localization systems such as Agentless, AutoCodeRover, LocAgent, and SweRank (Xia et al., 2025; Zhang et al., 2024; Chen et al., 2025; Reddy et al., 2026), alongside the CodeRankEmbed dense retriever (Suresh et al., 2025). Training-side work fine-tunes on software-evolution trajectories (Wei et al., 2025; Pan et al., 2025). Our concern is complementary: contamination (Liang et al., 2025) and surface path overlap raise the possibility that Recall@1 can overstate code understanding. NLI input ablations and counterfactual QA/code perturbations (Poliak et al., 2018; Jia & Liang, 2017; Yefet et al., 2020; Rabin et al., 2021) motivate PathSwap and PathAlias.

Recent work overlaps with individual checks: SACL masks textual code features and evaluates code retrieval, reranking, and SWE-bench-Lite localization (Gupta et al., 2025); a representation study compares paths, source, and generated summaries under retrieval and reranking (Caumartin et al., 2026); BLAgent ablates path augmentation and evaluates an Agentless repair integration (Mamun & Uddin, 2026); and BugCerberus evaluates hierarchical file/function/statement localization with Agentless repair (Chang et al., 2025). Surface path/name sensitivity is already documented in this literature (Xu et al., 2026). Our contribution is the joint measurement design: a fixed-pool displayed-path intervention with readable PathAlias and an opaque reference, a same-win scorer-referenced file-to-function transfer check, and a separately bounded executed-repair outcome check.

Related studies measure different quantities. RepoMirage (Li et al., 2026) perturbs context in end-to-end agents, LogicLoc (Xu et al., 2026) authors keyword-agnostic queries, and the SWE-bench illusion analysis (Liang et al., 2025) studies memorization and contamination. PathAlias instead breaks path–content correspondence using readable, form- and tokenizer-matched components from unrelated repositories. This differs from those agent, query-construction and contamination analyses and from opaque corruption. Matched opaque PathSwap is the reference; strict Alias–PathSwap ordering is supported only on Lite.

RepoMirage's runtime-target masking is closest, but operates inside a multi-hop agent rather than at a single scoring stage. Loc2Repair (Al Awad & Ivanov, 2026) and context ablations (Sepidband et al., 2026) find repair headroom with gold or perfect localization; they support the consequence axis without auditing same-win scorer-dependent certification.

K.2 Repository Structure, Retrieval and Reranking

Co-change history (Zimmermann et al., 2005; Ying et al., 2004; Zimmermann & Nagappan, 2008) and dependency graphs supply signals beyond lexical retrieval. GREPO (Wang et al., 2026), CodexGraph (Liu et al., 2025), CoSIL (Jiang et al., 2025), RepoGraph (Ouyang et al., 2025) and GRIP (Feng et al., 2026) use structural traversal or LoRA-internalized structure. Our path interventions preserve the graph-free scoring interface and fixed candidate pool; training recipes supply supporting diagnostics.

Hard-negative mining (Xiong et al., 2021; Qu et al., 2021) and pool mismatch (Gao et al., 2021; Jacob et al., 2025) motivate the checkpoint-dependent comparisons in Table 74. Reciprocal rank fusion (Cormack et al., 2009) does not reproduce the code-preview PathSwap contrast. The native SweRank result remains 57pp below its published reference (Appendix I); neither it nor the path-document probe establishes native function-body ranking sensitivity.