GraphAudit: Measuring Help and Harm in Code-Graph Retrieval

Matched graph interventions and protected lexical context for local coding models

Tejas Gluth | Research paper for the Gemma 4 Developer Agent Paper Track

Abstract

Repository graphs can change both the relationships available to a coding system and the code selected for its context. Without a structural control, a retrieval gain does not establish that dependency semantics caused it. We present GraphAudit, a reproducible paired-intervention protocol that holds source snapshots and lexical seeds fixed while replacing edges with degree-preserving null graphs. We evaluate all 300 SWE-bench Lite tasks and 129 official public competition tasks, and run a local quantized Gemma 4 candidate-selection study. On official tasks, graph fusion raises pooled file Recall@10 from 63.4% to 73.7%, but randomized graphs also improve it. The real-minus-null repository-macro contrast is 3.1 percentage points (95% interval [+0.7, +5.6]). On 54 tasks, Gemma with lexical candidates raises pooled top-one accuracy from 42.6% to 68.5%; graph-selected candidates reach 61.1%. A protected lexical prefix provides a simple inclusion guarantee without claiming to eliminate all retrieval harm. The contribution is an auditable resource for distinguishing graph effects, allocation effects, and implementation errors, rather than a claim of improved repair resolution.

1. Motivation and contribution

Local software-engineering models have limited context and inference budgets. A graph may help a bug report reach a related helper, but a graph-ranked candidate can also displace useful lexical evidence. Reporting only average accuracy conceals these losses. A stronger test asks whether the graph helps beyond a structurally randomized alternative using the same source and initial search results.

GraphAudit contributes three connected artifacts: a label-isolated intervention harness with exact-commit provenance; paired gain/loss and graph-null analyses across two task collections; and a prefix-protected allocation rule whose guarantee is deliberately narrower than a performance claim. Existing graph builders and localization benchmarks remain essential baselines. We do not introduce BM25, personalized diffusion, rank fusion, code graphs, or Gemma localization.

2. Protocol and implementation

Each task supplies its original issue and public repository base commit. Repository archives are parsed without executing fetched code. A fixed path filter excludes directory names for tests, docs, examples, benchmarks and vendored code, plus test filenames and setup/conftest scripts; other tutorial directories can remain. Migration implementations are retained. Oversized or undecodable files and syntax failures are recorded. File hashes, archive hashes, dataset revision, code hashes, query hashes, parameters and full rankings are saved. Neither hints nor reference patches enter retrieval.

The lexical baseline uses identifier-split BM25 (k1=1.2, b=0.75) over 80-line windows with a 60-line stride; each file receives its maximum window score. Whole-file BM25 is an additional baseline. Five lexical files seed a 12-step random walk with restart 0.5 over an undirected dependency graph. Reciprocal-rank fusion uses constant 60. All arms return ten distinct files from the same corpus.

External graphs contain statically resolved imports. For official tasks, provided symbol IDs are mapped to files using the longest exact module prefix, then cross-file call edges are collapsed; unmatched symbols are recorded. This changes the graph representation, so our outcomes should not be compared directly with symbol-level graph evaluations. Isolated files remain lexical candidates.

For each graph, seeds 11, 29 and 47 drive double-edge swaps. Swaps reject self-loops and duplicate edges and preserve every vertex degree. We target ten accepted swaps per edge with a bounded attempt count. We record accepted swaps and

overlap with the original graph; randomized graphs need not be independent uniform graph samples. Empty-graph and 50%-edge-deletion controls are included. A null contrast averages the three null outcomes within each task before inference; seeds never increase sample size.

The protected rule keeps the first m lexical files, then fills the remaining k-m slots from the fused order, skipping duplicates. For any gold set T, its recall is at least the fraction of T contained in that protected prefix. This is a basic set-inclusion invariant. It guarantees neither parity with the original top-k list nor better code repair. We fix m=3 and show 0/1/3/5/10 descriptively. Equal file slots are not equal token counts; source-character totals are recorded separately.

3. Data, estimands and audit corrections

The initial external sample selected up to eight tasks per repository by a label-independent hash: 24 development and 54 evaluation tasks across 12 repositories. We then extended to the remaining 222 tasks with ranking parameters unchanged. The extension spans six repositories. The resulting full resource covers all 300 Lite tasks; the original 78 remain separately identified. All 129 official public tasks form a transfer analysis over four repositories, including only one httpx task.

Reference patches identify changed Python files only in the scoring process. Added files absent at the base commit remain unreachable and reduce candidate coverage. Hit@k asks whether any gold file is present, Recall@k measures the fraction present, and all-files coverage requires every gold file. These differ on multi-file competition tasks. Reference edits are one valid solution, not proof that every alternative localization is wrong.

The primary estimand is repository-macro Recall@10; task-pooled results are also reported. Paired hierarchical intervals resample repositories and then tasks within sampled repositories, using 4,000 replicates. Few repository clusters limit population claims. Intervals are descriptive and unadjusted for the broader ablation family.

The protocol was recorded locally, not independently preregistered. Code and coverage audits found two defects: package roots beneath lib/ were not resolved correctly, and an overbroad migrations/ exclusion removed Django's migration engine. Both were corrected with regression tests, old artifacts retained, and complete source/index runs repeated. The corpus correction followed outcome inspection; consequently the corrected extension is not claimed as pristine unseen confirmation. The resource exposes these mistakes rather than silently dropping affected tasks. Final selected/scored counts are 300/300 externally and 129/129 officially; run errors are 0 and 0, respectively.

4. Retrieval results

Method	External extension macro R@10 (%)	Official macro R@10 (%)	Official pooled R@10 (%)
Chunk BM25	79.3	78.7	63.4
Real graph + RRF	73.0	84.2	73.7
Protected prefix (m=3)	73.0	84.2	73.7
Graph diffusion	69.8	77.3	61.2
Mean degree-preserving null	72.7	81.1	70.1

The external extension contains 222 scored tasks. Its real-graph RRF change against lexical retrieval is -6.3 percentage points, with interval [-19.5, +4.4]; task-level Hit@10 gains/losses are 11/15. This does not support adopting graph fusion universally.

On official tasks, RRF adds 12 Hit@10 successes and loses 0 against lexical retrieval. However, null graphs also recover tasks. The real-minus-mean-null macro contrast is 3.1 points, interval [+0.7, +5.6]; this supports a role for the supplied topology in this specific file-collapse pipeline, with the four-repository caveat. The equivalent external contrast is 0.3 points, interval [-5.8, +6.2]. The empty-graph control recovers the lexical outcome. Protecting the prefix preserves early lexical choices while retaining the measured top-ten graph results; it is not evidence of global no-regression behavior.

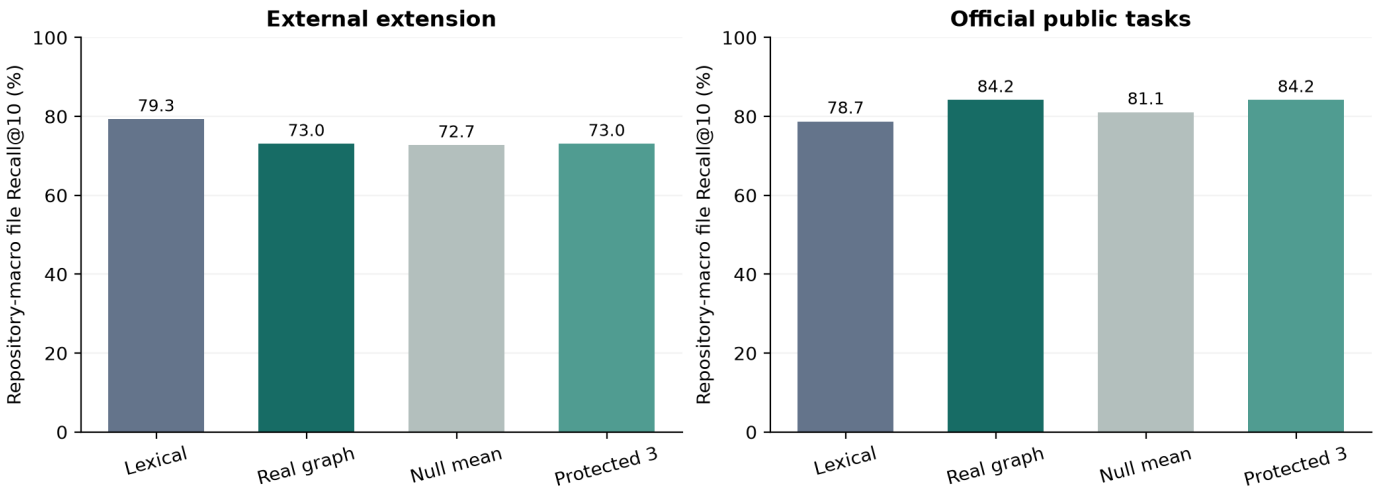


Figure 1. Matched file-slot retrieval. Null bars average three within-task rewired-graph outcomes. External and official columns use different graph sources and task populations; differences between collections are not a causal graph-source comparison.

5. Real local Gemma 4 study

We use the already-installed Gemma 4 GGUF model (7.996B total parameters, Q4_K_M), served by Ollama 0.40.8 on an Apple M4 Pro with 24 GB memory. The full model digest is stored in the manifest. This is neither the official QAT checkpoint nor the competition evaluator. No model is fine-tuned.

All 54 initial evaluation tasks are included. Gemma chooses one file from lexical, original import-graph RRF, or null-graph RRF candidates. Candidates receive their highest-scoring source window capped at 1,400 characters, and their order is shuffled deterministically independent of arm and gold. The issue is capped at 10,000 characters. No graph edges are shown; this tests source-set selection, not model graph reasoning. These stored candidate sets precede the implementation corrections and are explicitly an exploratory v1 study.

Decoding uses temperature 0, seed 42, thinking disabled, a 16,384-token context ceiling, and 128 output tokens, with a one-file JSON schema. Exact prompts, responses, actual prompt-token counts and durations are logged. There are 162 arm requests and 160 unique prompts. The desktop runtime updated itself mid-run; 78 completed predictions were preserved, and only 84 transport failures were replayed using an isolated, checksum-verified 0.40.8 runtime and the identical model digest. Failed choices were never retried.

Candidate source	Macro Hit@1 (%)	Pooled Hit@1 (%)	Wins/losses vs lexical set
Lexical rank 1, no model	44.4	42.6	3/17
Lexical candidates	68.8	68.5	0/0
Real-graph candidates	62.5	61.1	2/6
Null-graph candidates	63.2	61.1	1/5

Gemma with lexical candidates improves on lexical rank one by 24.3 macro points, interval [+2.8, +46.5], with 17 gains and 3 losses. For real-graph versus lexical candidates, the paired macro interval is [-17.4, +3.5] points. Three requests shared one prompt; transport recovery replayed them separately and all selected the same file. Character caps do not enforce equal token counts; median measured prompt lengths are 4908, 4902 and 4906 tokens. Candidate-oracle coverage and invalid outputs are included in the artifact. This small single-model study establishes no patch-resolution or general agent advantage.

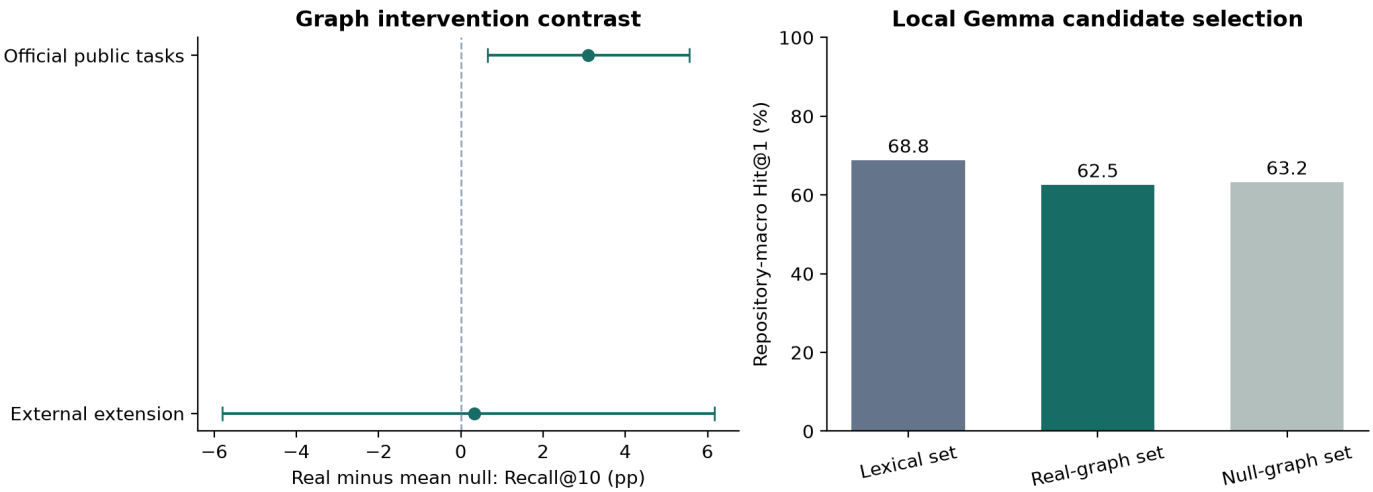


Figure 2. Left: paired real-minus-null graph contrasts with hierarchical intervals. Right: measured local Gemma file choices on the original 54-task cohort. Retrieval and model-choice outcomes are separate estimands.

6. Related work and limitations

SWE-bench [1] supplies real issue/patch pairs; Agentless [2] separates localization, repair and validation. RepoGraph [3] establishes repository graphs for software engineering, and CodeGraph-Loc [4] already provides lightweight graphs, entity labels, graph propagation and Gemma localization agents. Ali Hassan's competition notebook [5] also studies lexical/embedding/graph signals. Our narrower addition is an edge-intervention protocol with explicit nulls, paired harms, provenance and a protected lexical budget. StateTape [6] studies evolving evidence freshness; GraphAudit instead changes topology on fixed source. The Recall Trap [7] demonstrates why retrieval improvements require downstream testing.

Limitations include Python-only static analysis, approximate module resolution, file-level labels, public-data contamination risk, incomplete graph mapping, single-model greedy inference, and few independent repository clusters. Null swaps preserve degree rather than every graph statistic, and measured intervals do not prove equivalence when they span zero. We neither execute repairs nor claim a leaderboard score. The most useful next test is a fresh, token-matched, execution-graded Gemma study with stronger symbol graphs, preserved source provenance and the same graph-null controls.

7. Reproducibility and disclosure

Original code is Apache-2.0. External source and issues retain their original licenses. The companion notebook reconstructs tables, tests invariants and provides optional full reruns from exact public commits. Competition inputs and raw competition-derived labels are not redistributed; entrants can attach the gated competition input to rerun that analysis. Local model reruns require the specified weights and runtime. All reported numbers come from completed execution artifacts. Codex assisted research, design, implementation, analysis and writing; the entrant remains responsible for the submission.

References

[1] Jimenez et al. SWE-bench. ICLR 2024. https://arxiv.org/abs/2310.06770

[2] Xia et al. Agentless. 2024. https://arxiv.org/abs/2407.01489

[3] Ouyang et al. RepoGraph. ICLR 2025. https://github.com/ozyyshr/RepoGraph

[4] CodeGraph-Loc: Laptop-Scale Code Graphs for Small-Agent Bug Localization. https://github.com/Benadict-Git/codegraph-loc

[5] Ali Hassan. Code-Graph Bug Localization Benchmark. Kaggle, 2026.
https://www.kaggle.com/code/alikalwar/code-graph-bug-localization-benchmark-gemma-4-pap

[6] Yu et al. StateTape. 2026. https://arxiv.org/abs/2609.36319

[7] Adkins and Trapaidze. The Recall Trap. 2026. https://arxiv.org/abs/2608.14838