

Which Graph Signals Pay for Their Tokens?

cgraphy: A Token-Budgeted Code Knowledge Graph as a Portable Context Layer for AI Coding Agents

Prateek Mohan Garg

prateekmohangarg@gmail.com

July 2026 — draft v0.3

Abstract

AI coding agents spend most of their context re-reading source files to orient themselves — a cost paid on every prompt that grows linearly with repository size. Code knowledge graphs are an increasingly popular remedy, but recent systems bundle many graph signals at once, leaving open *which signals actually improve retrieval enough to justify their tokens*. We present **cgraphy**, an open-source, language-agnostic knowledge-graph layer served over the Model Context Protocol (MCP), and use it as an instrument to answer that question. cgraphy contributes (1) importance-weighted, *token-budgeted* subgraph extraction; (2) git co-change edges fused with the static graph under leakage control; (3) an *agent-self-enrichment* protocol in which the host agent writes persistent, content-hash-keyed semantic summaries — no server-side LLM or API keys; (4) edit-loop tools (budgeted symbol reading, blast-radius impact analysis, diff-to-symbol review context) with usage-aware retrieval; (5) optional hybrid retrieval fusing FTS with tiny static embeddings by reciprocal-rank fusion; and (6) two reproducible, grading-free benchmarks: a commit-derived localization suite (450 tasks, 9 repositories, 7 languages) and an issue-driven suite built from SWE-bench Lite instances evaluated at their pre-fix snapshots. Across 9 repositories, the signal stack lifts mean hit@10 from 0.602 (lexical baseline) to 0.733, with semantic fusion the largest single contributor at zero added payload; on real GitHub issues the full stack gains +15.8 points hit@5 and hit@10 (0.456 → 0.614). Gains concentrate where lexical search is weakest (C++ +28, JavaScript +24 hit@10) — and on one repository fusion *hurts*, underscoring that graph signals are measurable options, not assumed wins. Every configuration answers in a few hundred to ~1,300 tokens; on real private codebases, orientation via the graph is 134–420× cheaper than reading the code.¹

1 Introduction

Modern coding agents (Claude Code, Codex, Gemini CLI, Cursor) operate by listing directories, grepping, and reading files. This *exploration tax* recurs on every task: the agent reconstructs, from raw text, a mental model that is almost entirely stable across prompts — what the modules are, who calls whom, where the important entry points live. For large repositories the tax dominates context budgets; for vague prompts (“fix the login bug”) it compounds, because the agent does not know where to begin.

A natural remedy is to precompute that mental model as a graph and let the agent query it. Recent systems validate the idea: RepoGraph [1] plugs a line-level Python code graph into SWE-bench agents; CodexGraph [2] exposes a code graph database to an agent; LocAgent [3] frames localization as graph-guided search; Codebase-Memory [4] serves a tree-sitter knowledge graph

¹Code, benchmarks, and data: <https://github.com/pmgarg/cgraphy>

over MCP and reports order-of-magnitude token savings; the practitioner tool Aider popularized PageRank-ranked, token-budgeted repository maps [6]; and classic mining-software-repositories research showed that files which changed together in the past change together in the future [8, 7].

These systems bundle signals — structural edges, importance ranking, communities, history — and report aggregate wins. What is missing is a *decomposition*: which signal earns its place, on which kinds of codebases, at what token price? The question matters because every token of graph payload competes with tokens of actual code, and because Agentless [9] demonstrated that simple retrieval is a stubbornly strong baseline in this domain.

We build cgraphy to answer it. Design goals are strict: any language (two-tier tree-sitter extraction), any agent (MCP; one config line each for Claude Code CLI/VSCode/desktop, Codex, Gemini CLI, Cursor), zero infrastructure (one SQLite file), zero credentials (the host agent enriches its own graph), and a hard token budget on every answer.

Contributions.

1. *System*: a language-agnostic code knowledge graph with PageRank importance, git co-change edges, token-budgeted best-first subgraph extraction, and an agent-self-enrichment protocol with content-hash summary persistence, served over MCP (Section 3).
2. *Benchmarks*: two objective, no-human-grading localization suites — commit-derived (450 tasks, 9 repos, 7 languages) and SWE-bench-Lite-derived (real GitHub issues, pre-fix snapshots) — both with explicit leakage control for history-based signals (Section 4).
3. *Findings*: an ablation across six signal levels showing (a) lexical FTS is a strong baseline, (b) semantic fusion is the largest single signal and costs zero extra payload, (c) graph gains concentrate where lexical search is weak (C/C++/JS/Java), including one repository where fusion hurts, (d) co-change helps only where history is dense, and (e) all configurations stay 1–3 orders of magnitude cheaper than reading code (Section 5).
4. *Practice*: deployment mechanics that research systems omit: agent steering via instruction files, staleness-driven incremental re-indexing, and content sniffing that survives real user directories (a saved-webpage bundle with 73,794 minified JS files hung a filename-based indexer for hours; an 8 KB head-sniff fixed it) (Section 6).

2 Related Work

Repository graphs for agents. RepoGraph [1] improves SWE-bench agents with ego-graph retrieval over line-level Python graphs; CodexGraph [2] has agents emit graph-database queries; LocAgent [3] performs graph-guided localization; ARISE [5] targets fault localization and repair. Codebase-Memory [4] is closest to our system: a 66-language tree-sitter graph over MCP with Louvain communities and co-change edges, evaluated by author-graded answer quality. cgraphy differs in mechanism (budgeted rank-weighted extraction; agent-written persistent summaries; rank-blended search) and, more importantly, in evaluation philosophy: objective ground truth, signal ablations, and leakage control rather than graded QA — Codebase-Memory itself lists systematic comparison as future work. **Token-budgeted maps.** Aider [6] fits a PageRank-ranked map into a fixed token budget inside one CLI tool; cgraphy generalizes the idea to a queryable, protocol-standard service with per-query budgets. **Change coupling.** Logical coupling is classic MSR [8, 7]; we measure its marginal value for agent-facing retrieval with the fix commit excluded from mining. **Strong simple baselines.** Agentless [9] showed pipeline retrieval rivals agentic exploration; our lexical-baseline finding is consistent and quantified per language. **GraphRAG** [10] popularized

hierarchical community summaries for text corpora; porting them to code is our roadmap. SWE-bench [11] anchors end-to-end evaluation; we derive an objective localization suite from its Lite split.

3 System

Indexing. `cgraphy index` walks the repository (honoring `.gitignore/.cgraphyignore`; skipping binaries, lock files, and minified assets by content sniff), parses files with `tree-sitter`, and writes nodes (`file`, `class`, `function`, `method`) and edges (`contains`, `calls`, `imports`, `inherits`) into a single SQLite database with an FTS5 index over names and summaries. Tier-1 query specifications cover Python, JavaScript/TypeScript, Java, Go, C, C++, Rust; a generic AST walk extracts definitions for any other tree-sitter grammar. Re-indexing is incremental by content hash; cross-file references resolve by qualified name, best-effort, with unresolved names retained.

Ranking. Weighted PageRank (30 iterations, pure Python) runs over all non-containment edges after each index; search blends lexical relevance with importance: $\text{score} = \text{bm25} - 5.0 \cdot \text{rank}$.

Co-change edges. `--git-history` counts file pairs co-occurring in commits (up to 3,000 commits; support ≥ 3 ; commits touching > 50 files discarded) and adds `co_changes` edges with normalized weights.

Budgeted context. `cgraphy_context(symbol, budget)` expands best-first from the target node with priority $w \cdot (0.5 + \text{rank}) \cdot 0.6^{\text{depth}}$, emitting compact node lines until the budget (4 chars/token estimate) is exhausted. Cost scales with the question, not the repository.

Agent self-enrichment. Semantic summaries are written by the host agent through a two-tool loop: `cgraphy_enrich` returns unsummarized symbols (importance-first) with snippets; `cgraphy_store_summaries` persists one-liners keyed by a SHA-256 of the symbol’s source, so an edit invalidates only its own summary. The mechanism needs no server-side LLM, works in every MCP client (unlike MCP sampling, which few clients implement), and turns enrichment into an idle-time agent task.

Hybrid semantic retrieval (optional). Static `model2vec` embeddings (CPU-only, no torch; `cgraphy[semantic]`) are computed per node at index time and fused with FTS5 rankings by reciprocal-rank fusion, which requires no score calibration. This targets the vocabulary gap between issue-style prose and code identifiers that our evaluation identifies as lexical search’s dominant failure mode.

Subsystems and lineage. Deterministic label propagation over the file graph yields subsystem communities surfaced in the overview (a repo→subsystem→symbol zoom); dataset files (CSV headers, SQL `CREATE TABLEs`) receive deterministic schema summaries, making data directories searchable alongside code.

Edit-loop tools. Beyond orientation, three tools serve the change cycle: `cgraphy_read` returns one symbol’s line-numbered source under a budget (replacing whole-file reads); `cgraphy_impact` computes the blast radius of a proposed change — transitive dependents, tests likely affected, historically co-changed files; `cgraphy_diff_context` maps the working git diff to the exact symbols touched, their users, and covering tests. Retrieval is *usage-aware*: symbols an agent repeatedly focuses on receive a small, capped priority boost in later expansions (telemetry never leaves the local database).

Serving and steering. `cgraphy serve` exposes eight tools over stdio MCP; `cgraphy init` installs project configuration (`.mcp.json`) plus steering blocks in `CLAUDE.md/AGENTS.md` that direct agents to consult the graph before reading files and to check impact before editing shared code — servers cannot force tool use, only steer it. Tools self-heal: a cheap staleness check triggers incremental re-indexing before answering. `cgraphy view` renders the graph locally for humans

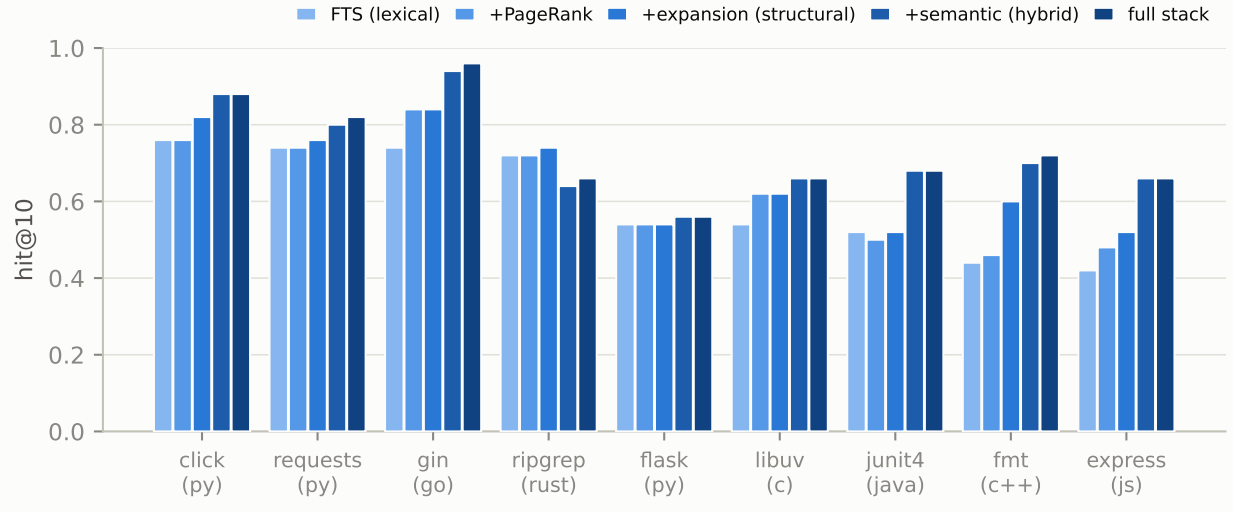


Figure 1: hit@10 by repository and ablation level (darker = more signal), sorted by lexical-baseline strength. The stack helps most at the weak end; ripgrep is the honest counterexample.

(Cytoscape.js, no CDN).

4 Evaluation Design

Both benchmarks are objective (no LLM judging, no human grading) and reproducible from public data.

Commit-derived localization. From each repository’s history we select recent fix-like commits (bug-fix lexicon in the subject; 1–5 code files changed; files still present at HEAD). The subject is the query; the touched files are ground truth. Co-change mining *excludes the evaluated commits*. 50 tasks per repository; 9 repositories spanning Python, C++, Go, Rust, JavaScript, Java, and C (450 tasks).

Issue-driven localization (SWE-bench Lite). For instances from seven tractable SWE-bench Lite projects we check out the exact `base_commit`, index that snapshot, query with the real GitHub issue text (code blocks stripped, first 120 words), and score against the gold patch’s files. Because the snapshot predates the fix, history-based signals cannot leak by construction.

Methods. An ablation ladder: FTS (FTS5 bm25); RANK (+PageRank blending); EXPAND-NC (+one-hop weighted expansion, co-change removed); EXPAND (full graph). **Metrics:** hit@ k (first ground-truth file in top k), MRR, and mean retrieval payload in tokens.

5 Results

Commit-derived suite (Table 1, Figures 1–2). Across 450 tasks, mean hit@10 rises monotonically along the ablation ladder: 0.602 (FTS) $\rightarrow$ 0.629 (+PageRank, free) $\rightarrow$ 0.662 (+expansion/co-change) $\rightarrow$ 0.724 (+semantic fusion) $\rightarrow$ **0.733** (full stack) — a +13.1-point total gain, with +8.6 hit@5 and +0.076 MRR. Four patterns stand out. *First*, semantic fusion is the single largest signal and costs nothing extra: the HYBRID rung uses the same payload as FTS (364 tokens mean) yet adds +12.2 points hit@10, with the biggest jumps exactly where identifiers and prose diverge (junit4 +18 hit@5; express +24 hit@10 over FTS; fmt +26 with expansion). *Second*, gains concentrate

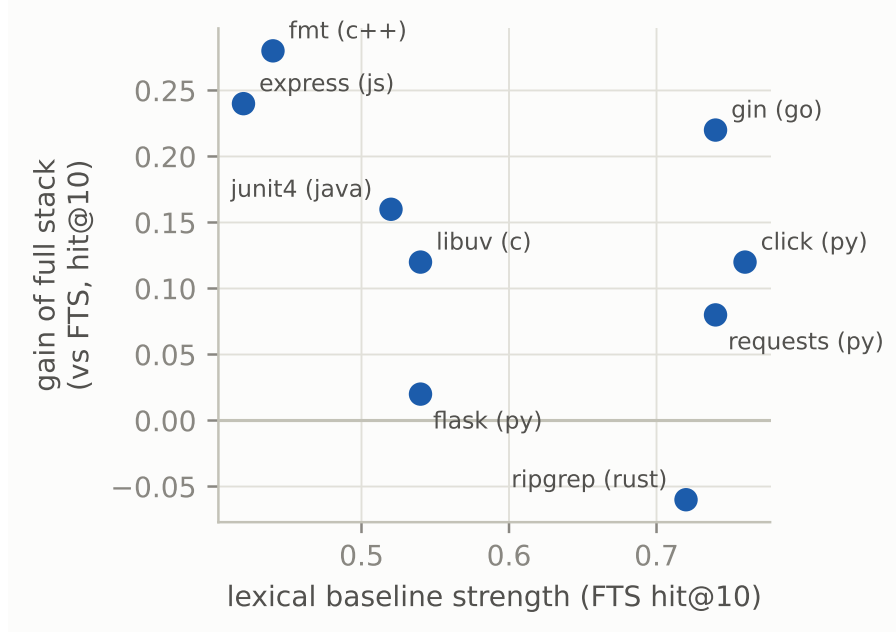


Figure 2: Full-stack gain versus lexical baseline strength. The largest gains (fmt +28, express +24, gin +22 points hit@10) occur where bm25 is weakest or naming is terse; on ripgrep the stack *loses* to lexical — signal utility is repository-dependent and worth measuring.

where the lexical baseline is weak (Figure 2): C++’s header/implementation split and JavaScript’s terse naming defeat bm25, and the graph supplies what naming does not. *Third*, PageRank is free and never harmful on hit@10 (up to +10 on gin); co-change contributes where history is dense (click +4). *Fourth*, the signals are not universally additive: on ripgrep, fusion *hurts* (0.72 → 0.66 hit@10) — Rust’s already-precise identifiers make lexical rank the better prior, and RRF dilutes it. Signal utility is measurable, repo-dependent, and worth measuring per codebase — precisely the argument for shipping the benchmark with the tool.

Issue-driven suite (SWE-bench Lite, Table 2). Real issue texts are markedly harder than commit subjects: the lexical baseline drops to hit@10 0.456 (from 0.602 on the commit suite), because user-written reports describe symptoms in prose that shares little vocabulary with code identifiers. This is exactly the regime where semantic fusion earns its keep: HYBRID adds **+15.8 points hit@5** (0.333 → 0.491) and +12.3 hit@10 at the same payload as FTS, and the full stack reaches **0.614 hit@10** (+15.8 over lexical, +12.3 over the best structural rung). PageRank and co-change contribute little here — pre-fix histories at often-years-old base commits offer sparse support. We emphasize the operating point: these are single-shot, sub-second, LLM-free retrievals costing 450–1,300 tokens that seed an agent’s *first* tool call; LLM-in-the-loop localization agents [3] report higher file accuracy at orders of magnitude more tokens, and the two are complementary layers, not competitors.

Token economy. Retrieval payloads range 148–1,239 tokens per query across all repositories and methods. On three private codebases (852, 360, and 592 files; Python and C++), full orientation via **overview+search+context** cost 2,623–4,431 tokens versus 0.59M–1.25M tokens for reading the code: **134–420×** reductions, growing with repository size since the graph payload is budget-capped while code volume is not.

6 Deployment Practice

Two findings that research prototypes rarely report. *Steering is part of the system*: an MCP server cannot force an agent to prefer it; cgraphy ships instruction-file blocks (`CLAUDE.md`, `AGENTS.md`) and imperative tool descriptions, which in our experience determine whether the graph is consulted at all. *Real directories are hostile*: a user project containing saved webpages (73,794 JavaScript files of minified code without `.min` names) made a filename-based indexer run for hours; an 8 KB head-sniff for kilobyte-long lines reduced indexing to seconds. Robustness of this kind is a precondition for any of the retrieval gains to matter.

7 Limitations and Threats to Validity

Commit subjects are terser than real prompts (the SWE-bench suite mitigates this with genuine issue texts). Ground truth equals the fixing patch’s files — necessary but not always sufficient context. Nine repositories and seven languages remain a sample; effects are modest against a strong lexical baseline and repository-dependent, and we frame them accordingly. The SWE-bench-derived suite measures localization, not end-to-end issue resolution; running full SWE-bench with an agent steered by cgraphy is the next experiment and requires substantial compute. Token counts use a 4-chars/token estimate. Agent-written summaries were empty during all benchmarks: embeddings covered only names, qualified names, and signatures, and expansion used only structural text — so the reported numbers are a lower bound on what an enriched graph could achieve. Usage-aware reweighting is implemented but unevaluated (it requires longitudinal agent traces); the RRF fusion weights and the ripgrep regression suggest per-repository signal calibration as concrete future work.

8 Conclusion

cgraphy packages code knowledge graphs as a zero-infrastructure, any-agent, token-budgeted context layer spanning both the reading loop (overview, search, context, symbol-level reads) and the editing loop (impact analysis, diff review), and uses two objective benchmarks to decompose which signals earn their tokens: semantic fusion (largest, free at query time, occasionally harmful), PageRank (small, free, consistent), structural expansion (moderate cost, biggest where lexical retrieval fails), and co-change (history-quality dependent). The token economy — hundreds of tokens per query against 134–420× more for reading code — is the unconditional win. Remaining roadmap: evaluating usage-feedback reweighting on longitudinal traces, summary-quality evaluation after agent enrichment, per-repository signal calibration, and full SWE-bench agent runs.

References

- [1] Ouyang, S., et al. RepoGraph: Enhancing AI Software Engineering with Repository-level Code Graph. arXiv:2410.14684, 2024.
- [2] Liu, X., et al. CodexGraph: Bridging Large Language Models and Code Repositories via Code Graph Databases. arXiv:2408.03910, 2024.
- [3] Chen, Z., et al. LocAgent: Graph-Guided LLM Agents for Code Localization. arXiv:2503.09089, 2025.

- [4] Codebase-Memory: Tree-Sitter-Based Knowledge Graphs for LLM Code Exploration via MCP. arXiv:2603.27277, 2026.
- [5] ARISE: A Repository-level Graph Representation and Toolset for Agentic Fault Localization and Program Repair. arXiv:2605.03117, 2026.
- [6] Gauthier, P. Building a better repository map with tree-sitter. aider.chat, 2023. <https://aider.chat/2023/10/22/repomap.html>
- [7] Zimmermann, T., et al. Mining Version Histories to Guide Software Changes. ICSE 2004.
- [8] Gall, H., Hajek, K., Jazayeri, M. Detection of Logical Coupling Based on Product Release History. ICSM 1998.
- [9] Xia, C., et al. Agentless: Demystifying LLM-based Software Engineering Agents. arXiv:2407.01489, 2024.
- [10] Edge, D., et al. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. arXiv:2404.16130, 2024.
- [11] Jimenez, C., et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? ICLR 2024.

Table 1: Commit-derived localization: 50 fix-commits per repository (July 2026 snapshots). FULL = hybrid semantic fusion + graph expansion. Best per repo in bold.

Repo (lang)	Method	hit@5	hit@10	MRR	Tokens
click (py)	FTS	0.76	0.76	0.589	406
	EXPAND	0.80	0.82	0.609	1,264
	FULL	0.80	0.88	0.671	1,281
requests (py)	FTS	0.70	0.74	0.476	415
	EXPAND	0.70	0.76	0.478	1,233
	FULL	0.76	0.82	0.582	1,260
flask (py)	FTS	0.46	0.54	0.288	354
	EXPAND	0.46	0.54	0.297	724
	FULL	0.48	0.56	0.382	958
fmt (c++)	FTS	0.32	0.44	0.191	349
	EXPAND	0.48	0.60	0.251	1,035
	FULL	0.54	0.72	0.296	1,043
gin (go)	FTS	0.70	0.74	0.588	291
	EXPAND	0.78	0.84	0.622	1,094
	FULL	0.86	0.96	0.718	1,090
ripgrep (rust)	FTS	0.66	0.72	0.476	363
	EXPAND	0.66	0.74	0.447	1,245
	FULL	0.58	0.66	0.455	1,118
express (js)	FTS	0.42	0.42	0.332	207
	EXPAND	0.50	0.52	0.342	662
	FULL	0.52	0.66	0.417	720
junit4 (java)	FTS	0.48	0.52	0.410	574
	EXPAND	0.46	0.52	0.424	1,159
	FULL	0.66	0.68	0.516	1,171
libuv (c)	FTS	0.52	0.54	0.460	319
	EXPAND	0.50	0.62	0.451	1,216
	FULL	0.60	0.66	0.450	1,232
mean (9 repos)	FTS	0.558	0.602	0.423	364
	RANK	0.571	0.629	0.426	364
	EXPAND-NC	0.584	0.653	0.433	1,070
	EXPAND	0.593	0.662	0.436	1,070
	HYBRID	0.642	0.724	0.495	364
	FULL	0.644	0.733	0.499	1,097

Table 2: Issue-driven localization: 57 SWE-bench Lite instances across seven projects, each evaluated at its pre-fix `base_commit` snapshot with real GitHub issue text as the query.

Method	hit@5	hit@10	MRR	Tokens
FTS	0.333	0.456	0.239	450
RANK	0.333	0.456	0.239	450
EXPAND-NC	0.351	0.491	0.246	1,308
EXPAND	0.351	0.491	0.246	1,308
HYBRID	0.491	0.579	0.301	450
FULL	0.491	0.614	0.306	1,298