Filesystem Is All You Need?

How Coding Agents Should Find Things: Grep, Embeddings, Git History and the Traces of Earlier Agents

Benchmark report

November 2026

Abstract

Agent memory is usually sold as a storage problem: add a vector database, add a knowledge graph. We ask a narrower question. An agent already lives in a filesystem. Which way of *finding* things there makes it right more often and cheaper? We test three claims: that keyword search is enough, that git history is a world model, and that the trails earlier agents leave behind can rank files the way links rank web pages. The main study uses only real, public data with deterministic scoring: 260 GitHub issues opened after the models' training cutoff, 100 questions about past states of real code mined from git history, and 204 real duplicate issues. It covers three agents (Claude Code with Claude Haiku 5.5 and Claude Sonnet 5.5, and Codex CLI with gpt-5-mini) in 4,076 scored runs in isolated E2B sandboxes, under a protocol written before the main run. **(1) Git history turns guesses into facts:** on questions about the past, read-only git lifts every agent from 52–66% to 92–100% (+34 to +40 points, all 95% CIs far from zero); without git, agents give today's value instead of the old one 30–46% of the time. **(2) The traces of earlier agents are a real ranking signal:** on recurring problems, pushing the files an earlier agent cited raises accuracy for all three agents (+3.9 to +9.8 points; significant for Sonnet and Codex) and beats commit-history hints, whose top-5 contained the right file 2.5 times less often. Wrong traces did not mislead. **(3) On fresh issues, retrieval add-ons do not matter:** grep, a semantic search tool and pushed embedding hints all land at 82–84% for Codex, and Codex never called the semantic search tool in 260 runs although it was available every time. A pilot on a synthetic company workspace (4,756 runs) motivated the design and agrees on every point it could test.

1 Motivation

The most successful agents today, coding agents, work on a plain filesystem with `grep`, `glob` and `read`. The thesis behind this report is that a filesystem plus its version history already is a world model: it holds the current state, every past state, and (through who opened and cited what) a signal of what matters. We test three claims:

1. **Retrieval.** Semantic (embedding) search adds little to an agent that can grep.
2. **Time.** Git history lets an agent answer questions about earlier states of the world.
3. **Ranking.** The traces of earlier agents work as a ranking signal, like links did for web search.

2 Pilot: a synthetic company workspace

We first built "Kiln", a fictional 25-person startup (223 files, grown to 1,757) with a backdated git history, facts that change over time, stale documents and near-duplicates, written by Sonnet from

a known fact timeline, and ran 4,756 Claude Code sessions under ten retrieval setups. In short: in a small tidy workspace every setup scored 96–100%; on 40 buried, reworded questions embeddings alone were the *worst* setup (57–62% vs. 74–84% for grep), and five embedding models ranked the right file first only 20–28% of the time, no better than BM25 (30%); read-only git answered every question about the past (100% vs. 58–83%); agents almost never called an optional tool they did not know (1–4% of runs); and pushing earlier agents' traces paid off only when questions recurred.

The pilot had a serious weakness: the same model family wrote the world, the questions and the paraphrases, and graded the answers; the questions were designed by us; parameters were chosen by hand; and samples were small. Every claim below is therefore re-tested on real data.

3 Main study: setup

3.1 Protocol

A protocol was written before any main-study agent run (`v2/PROTOCOL.md`) and every later change is logged there with its reason. Key rules: **no parameter is tuned**; all retrieval parameters are copied unchanged from the pilot; task samples are drawn once with seed 0; one run per (task, setup, agent), with power coming from many tasks; scoring is deterministic (no LLM judge); comparisons are paired within an agent, with a bootstrap over tasks (10,000 resamples) for 95% confidence intervals.

3.2 Agents and sandbox

Every run is a fresh non-interactive session inside an E2B sandbox (4 vCPU, 8 GB):

- **Claude Code** 2.1.331, `claude -p`, Claude Haiku 5.5 or Claude Sonnet 5.5, isolated from any personal configuration (`--setting-sources ""`, `--strict-mcp-config`, a fresh config directory per run).
- **Codex CLI** 0.172.0, `codex exec -s read-only`, gpt-5-mini, with its native shell.

The prompt is fixed per track; only the retrieval capability changes (Table 1).

Setup	What the agent gets
`grep`	Claude: Glob, Grep, Read. Codex: its shell. Repository without `.git`.
`hybrid`	grep + a `semantic_search` tool (MCP; bge-small-en-v1.5, 40-line chunks)
`grep+push-embed`	grep; the 5 files most similar to the task are appended to the prompt as hints
`grep+push-hist`	grep; 5 files changed by similar past commits (BM25 over commit messages) as hints
`grep+push-crowd`	grep; 5 files that *earlier agents* cited on similar tasks, as hints
`grep+git`	grep + read-only `git log/show/diff/blame` (repository with `.git`)

Table 1: Setups. Hints are marked "may be wrong; verify". Pushed hints are computed outside the agent.

3.3 The crowd signal ("PageRank for the filesystem")

For a new task we find earlier tasks whose text is similar (embedding cosine ≥ 0.55) and score each file as $\sum s \cdot (+1$ if the earlier answer was correct, -0.7 if wrong) over the files the earlier agent cited, plus $0.05\,s$ for files it only opened. The top 5 are pushed, each annotated with its evidence ("cited in 1 correct / 0 wrong past answers"). These weights are the pilot's, unchanged.

3.4 Track A: fresh GitHub issues (localization)

We mined 1,264 issues from 30 popular repositories that were opened on or after 2026-08-01 (after the models' training cutoff, so they cannot be memorized) and closed by a merged pull request touching 1–4 non-test source files. We sampled up to 10 per repository: **260 issues in 26 repositories** across Python, Rust, Go and TypeScript/JavaScript (Django, FastAPI, httpx and Sphinx yielded none). The repository is checked out at the pull request's base commit without `.git`; the agent must name the file to change. Metric: **hit@1**, the first file named is one the real fix changed.

Embedding indexes are expensive on CPU (about 10 chunks/s), so each repository gets *one* index (459,579 chunks in total) built at the base commit of its earliest sampled issue. No fix leaks in, and later issues see an index that is days to weeks behind the code, which is how indexes are maintained in practice.

3.5 Track B: time travel in real git history

From commits since 2023 in 18 Python repositories we mined every change to the default value of a public keyword parameter (case-only renames excluded) and asked: "*As of March 3, 2025, what was the default value of the parameter* `p` *of* `f` *in* `path`*?*" The date is before the change, and the answer differs from today's value; the ground truth is read with `git show` at that date. Each question has a matched control ("…today?"). The repository is at HEAD. Metric: exact match of the normalized literal. **100 past questions and 100 controls from 14 repositories** (three repositories, including black, had no qualifying change; sympy was dropped, see Limitations).

3.6 Track C: recurring problems (real duplicates)

Maintainers close many issues as duplicates of an earlier one. We mined 465 such pairs where the original issue was fixed by a merged change and the duplicate was filed after the original, and sampled up to 12 per repository: **204 duplicates in 26 repositories** (vite and transformers excluded, see Limitations). Ground truth is the fix's source files; the repository is at the fix's base commit.

- **Phase 1 (earlier agents).** Claude Haiku 5.5 with grep tries every *original* issue in the repository once (189 runs, 139 correct). Its cited files and whether it was right form the trace store.
- **Phase 2.** Each duplicate is attempted under grep, grep+push-hist and grep+push-crowd, where the crowd signal sees only traces of originals filed before that duplicate.

4 Results

4.1 Git history turns guesses into facts

Agent	grep	grep+git	difference (95% CI)	gave today's value instead (grep → git)
Claude Haiku 5.5	58%	**98%**	+40 [+30, +49]	35% → 2%
Claude Sonnet 5.5	66%	**100%**	+34 [+25, +43]	30% → 0%
Codex (gpt-5-mini)	52%	**92%**	+40 [+28, +51]	46% → 8%

Table 2: Track B, 100 questions about past states of real code. On the 100 matched "today" controls every agent scores 98–100% with and without git (no loss).

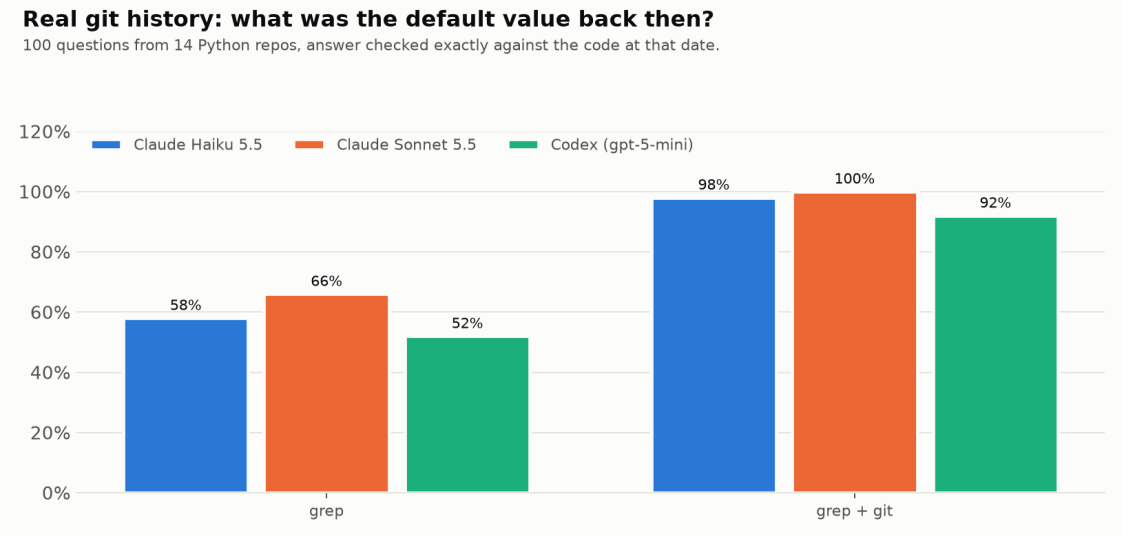

Figure 1: Track B accuracy on past questions.

Without history an agent can only see the present, so its dominant error is to report today's value as if it had always been true. With read-only git, every agent recovers the past almost perfectly, and it is cheaper for Claude (Haiku \$0.0032 → \$0.0019 per question, 8.0 → 4.7 tool calls). This replicates the pilot's time-travel result on real repositories, real changes and a second agent family.

4.2 The traces of earlier agents are a ranking signal

Agent	grep	+ commit hints	+ traces	traces − grep (95% CI)	tokens
Claude Haiku 5.5	65.2%	67.6%	**69.1%**	+3.9 [−0.5, +8.3]	62.6k → 55.2k
Claude Sonnet 5.5	69.6%	70.6%	**76.0%**	+6.4 [+2.0, +10.8]	30.4k → 26.7k
Codex (gpt-5-mini)	58.8%	58.8%	**68.6%**	+9.8 [+4.4, +15.2]	280k → 335k

Table 3: Track C, 204 real duplicate issues, hit@1. "Commit hints": files changed by similar past commits. "Traces": files earlier agents cited. Commit hints are never significantly different from grep.

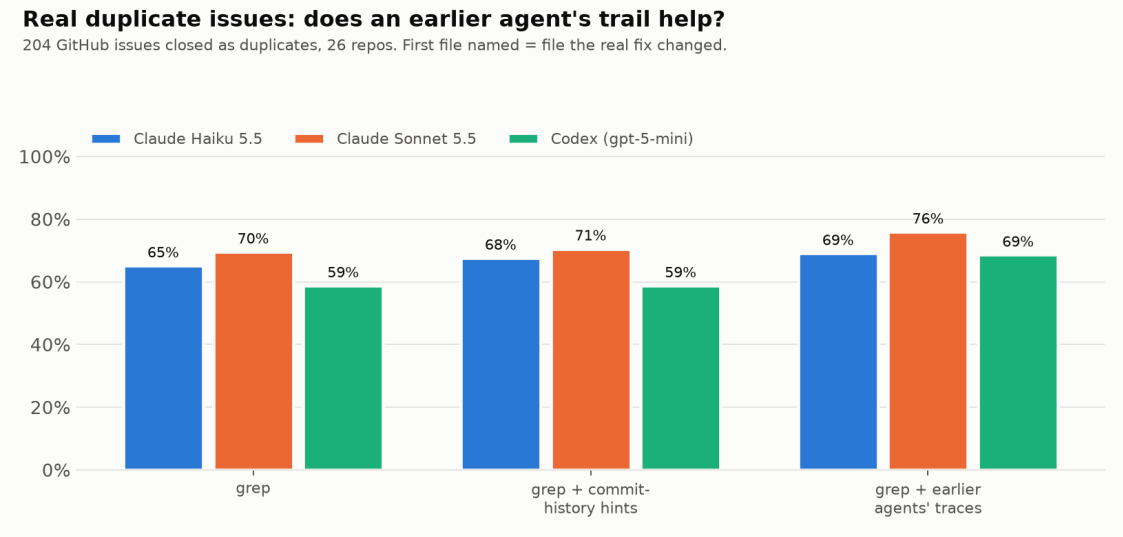

Figure 2: Track C accuracy.

Why the traces work: their top-5 contained a file the real fix changed for **132 of 204** duplicates (82 times in first place), against **52 of 204** (21 first) for commit-history hints. And they are safe: when the traces were wrong (61 duplicates, mostly because the earlier agent itself was wrong), accuracy stayed at or above grep's on the same issues (Haiku 46% vs. 48%, Sonnet 59% vs. 48%, Codex 51% vs. 46%). When they were right, every agent gained (Haiku 75% → 82%, Sonnet 80% → 85%, Codex 65% → 77%). Claude also read fewer tokens with traces; Codex read more, because it opened the hinted files in addition to its own search.

4.3 The ranking itself: quality grows with traffic

Agent accuracy measures the ranking only indirectly. We also scored the rankers directly, offline, against the files the real fix changed (no agent runs). Table 4 scores the hint lists the agents actually saw.

Ranker	Track	right file first	in top 5	MRR
Embedding similarity	A (fresh issues)	23%	55%	0.34
Commit history	A (fresh issues)	22%	46%	0.31
Commit history	C (duplicates)	10%	26%	0.16
Earlier agents' traces	C (duplicates)	**40%**	**65%**	**0.51**

Table 4: Ranking quality of the pushed hint lists.

We then replayed the frozen crowd formula on Track C with only a share of the earlier agents' traces available (5 random subsets per share, 203 duplicates). Replayed on all traces the ranker is somewhat stronger than in the live runs (50% vs. 40% first), because the trace store is rebuilt from the saved runs.

Traces available	0%	10%	25%	50%	75%	100%
Right file ranked first	0%	10%	20%	31%	40%	50%
Right file in top 5	0%	12%	26%	44%	60%	76%

Table 5: The crowd ranker learns from use: quality rises steadily with the amount of earlier work.

Two ablations sharpen what the signal is. **Without the trace of the duplicate's own original**, the ranker puts the right file first only 5% of the time: the signal is a memory of problems seen before, not general knowledge transferred from different problems. **The correctness label adds little.** A simplified replay that keeps the same candidate traces and changes only the per-file weighting puts the right file first 57% of the time with the success-weighted formula, 57% when every cited file counts the same, 57% when every opened file counts the same, and 61% when only files cited in correct earlier answers count (top-5: 74%, 86%, 87%, 75%). The earlier agent was right for 74% of the originals, so wrong traces were rare; what drives the ranking is the presence of a trace of the same problem, not whether it was a success. (Ranking from only wrong traces is near zero by construction, since a wrong earlier answer means its first cited file was not the fix.)

4.4 On fresh issues, retrieval add-ons do not matter

Setup (Codex, gpt-5-mini)	hit@1	difference vs. grep (95% CI)	input tokens	tool calls
grep	81.9%		202.5k	7.4
grep + semantic search tool	82.7%	+0.8 [−2.3, +3.8]	202.6k	7.1
grep + 5 similar files pushed	84.2%	+2.3 [−0.4, +5.0]	183.8k	6.9
grep + commit-history hints	82.3%	+0.4 [−3.5, +4.2]	209.3k	7.5

Table 6: Track A, 260 issues opened after the training cutoff. No setup differs detectably from grep.

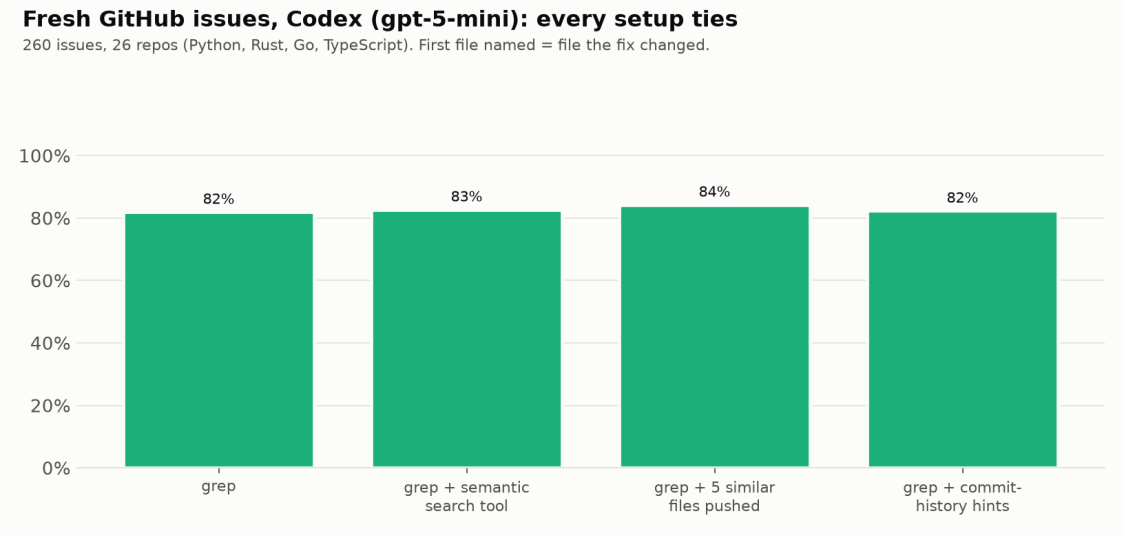

Figure 3: Track A accuracy (Codex).

Real bug reports name the things that are broken (error messages, function names, flags), which is exactly what grep needs. Pushed embedding hints save about 9% of tokens but do not change accuracy. Strikingly, Codex **never called the semantic search tool in 260 runs**, although a marker file confirmed that the tool server had started in every one of them, and the same agent calls it immediately when told to. Together with the pilot (Claude called an optional trace tool in 1–4% of runs), this says: *agents use the tools they already know; anything new has to be pushed into the context.*

5 Discussion

- **History is the cheapest world model.** Git already stores every past state; agents only need permission to read it. This is the largest effect we measured, it holds for every agent, and it costs nothing to build.
- **Usage traces are a real ranking signal, and they generalize.** The pilot's recurring-question result replicates on real duplicate issues, across two agent families, with parameters frozen in advance. Like links on the web, the signal needs prior traffic: it helps when work recurs, which in real projects it does (at least 465 fixed-and-duplicated issues in our 28 repositories since 2023).
- **Embeddings are not where the gains are.** On real issues they neither help nor hurt; in the pilot they lost when near-duplicates crowded the results.
- **The filesystem remembers, it does not generalize.** The trace ranker improves steadily with traffic, and its power comes from repeats of the same problem; weighting traces by success barely changes it.
- **Push, don't offer.** An optional tool is ignored; the same information in the prompt is used.

6 Limitations

- **Track A has one agent.** The Anthropic API credit ran out during Track A; the few Claude runs on A are not reported. Claude results come from B and C only.
- **Coverage.** Track B dropped sympy (its question mining had not finished when the credit ran out), so all three agents answer the same 100 questions. Track C dropped vite and transformers, whose phase-1 runs could not be completed. Comparisons are paired within agents, so this changes coverage, not the comparisons.
- **Crowd traces come from one earlier agent per original.** Real deployments would accumulate many traces; our signal is the sparse case.
- **Codex cost** is reported as tokens only. Codex token counts include its cached context and are not comparable to Claude's.
- **Track C originals may be memorized** (filed 2023–2026); this affects the earlier agent's accuracy, not the paired comparison on duplicates.
- Each task runs once per setup; confidence intervals are over tasks.

7 Reproducibility

Everything is in `experiments/fs-retrieval/`. Main study (`v2/`): task mining (`mine_github.py`, `mine_git.py`), frozen samples (`sample.py`, `data/tasks_*.json`), protocol and change log (`PROTOCOL.md`), sandbox template (`template.py`), sharded index build (`build_index.py`), sandbox worker and orchestrator (`worker.py`, `orchestrate.py`), analysis and charts (`analyze.py`, `charts.py`). Every run's full record (tool calls, hints shown, answer, tokens, cost) is in `v2/results/`, and the tables in `v2/analysis/results.md`. The pilot's code and results are in the parent directory (`RESULTS.md`).