Can Graph-Based Context Engineering Substitute for Model Scale? Task Resolution and Cost for Cheap and Frontier Models on Repository-Level Code Tasks

Akhil Katakam
KoraGraph
github.com/agilkatakam

August 2026

Abstract

A cheap model given structured, graph-derived context solved nine of eleven repository-level code tasks at $0.027 per solved task. A frontier agentic coding system searching the same repositories with grep solved nine as well, at $2.82. The ratio is 105.9×, and it is a floor rather than a point estimate. Frontier spend is modelled at published list rates, and the executing engine's own reported figures exceed that model by a fifth to a quarter.

The gap is not a general effect of retrieval. In a 2×2 factorial of model tier against context source, switching the cheap model from grep to graph retrieval moved it from 2/11 tasks to 9/11, winning seven tasks and losing none. The same switch moved the frontier tier not at all, which records 9/11 either way. Structured context substitutes for model scale in one tier and is redundant in the other, and that interaction, rather than the cost figure, is the result measured without confounds.

Every condition received three independent draws on every task, so the full matrix is $11 \times 4 \times 3 = 132$ cells, and it was executed three times with the same tasks-solved outcome each time. Tasks are drawn from two open-source repositories at pinned commits and graded by executable oracles hidden from every condition. The suite is released with this paper.

We do not present the cost comparison as an equivalence test. Eleven tasks and four discordant pairs cannot establish parity, and the two sets of nine are not the same nine: each condition misses two tasks the other solves, and neither misses more. What the corpus supports is that two configurations priced two orders of magnitude apart solved the same number of tasks on it, and that the cheap condition's failures concentrate in tasks requiring wide, uniform edits.

1 Hypothesis

The claim under test is narrow and falsifiable:

> *On bounded repository-level code tasks, a cheap model given structured context derived from a code knowledge graph solves as many tasks as a frontier agentic coding system given ordinary grep search over the same repository.*

Each part of that sentence is load-bearing. The outcome is *tasks solved*, not output quality, latency, or developer preference. The scope is *bounded* tasks, meaning ones with a statement, a pinned commit, and a mechanical oracle, and not open-ended engineering. And the subject is a

deployed configuration rather than a set of model weights: the cheap condition includes its retrieval layer, and the frontier condition includes its agentic search loop.

The reason to ask is economic. If the hypothesis holds, the cost of the retrieval layer is paid once and amortised, while the cost of the frontier tier is paid per task forever. If it fails, it should fail visibly, and the failure should say which tasks resist cheap models.

A second question falls out of the same design, and it turns out to carry the result: does structured context help both tiers equally? A retrieval layer that lifts every model is a convenience. One that lifts a cheap model to the level of an expensive one while doing nothing for the expensive one is a substitution, and substitution is the economically interesting claim.

2 The system under test

The graph condition supplies the model with context assembled by **KoraGraph**, a persistent and queryable representation of the repository, rather than from a text search over it. The representation records the entities a codebase contains and the relationships between them. It is built once per repository revision and reused across every task run against that revision. Retrieval against it combines semantic similarity with traversal of those relationships, so the context assembled for a task can include code related to the task that shares little vocabulary with its description. That is the property grep cannot offer, and the one this experiment prices.

We describe it at this level on purpose. What the experiment needs in order to be reproducible is the *interface*: each condition receives a context block, the blocks are held to a shared byte budget, and what differs between conditions is how the block was chosen. Nothing in the analysis depends on the internal construction of the representation, and a reader who substitutes a different graph retriever behind the same interface is running the same experiment. Section 8 describes what is released so that substitution can be performed.

The grep condition is the comparator a working engineer would actually reach for: search the repository for terms drawn from the task statement, rank the hits, and pass the top-ranked file contents forward under the same byte budget.

3 Experimental design

3.1 The four conditions

The design is a 2×2 factorial. One factor is model tier: a frontier agentic coding system with an unrestricted tool loop, against a cheap model invoked in a single-shot loop with bounded retries. The other factor is context source, grep against graph retrieval.

	Grep	Graph retrieval
Frontier model	A frontier + grep	C frontier + graph
Cheap model	D cheap + grep	B cheap + graph

The two models stand in for the ends of a deployment decision rather than for their tiers as a whole. The frontier condition is `claude-opus-5` driven through Claude Code [2], which is what supplies the unrestricted tool loop: within a single cell the system can search the repository, read files, execute the repository's own tests, and revise before it emits a patch. The cheap condition is `deepseek/deepseek-v4-flash-0731` [4], called through an inference gateway in a single-shot loop with bounded retries and no tool access, so everything it learns about the repository arrives

in the context block it is handed. That difference in loop architecture is deliberate, because it is the difference a buyer is actually choosing between, and it is also why the diagonal is confounded.

Both run at temperature 0, and costs are computed against rate table version `2026-08-07.1`. One model per tier is a real limit on what "tier" can mean here: any tier effect reported below is confounded with the identity of the model standing in for it, and a different frontier or cheap model could move either row of the matrix.

The primary endpoint is the **diagonal**, B against A. It is the comparison the hypothesis is about, and it is also the most confounded comparison in the matrix, since model tier, context source and loop architecture all move at once. We treat it as quotable only because the off-diagonal cells are run as well, so the decomposition that would explain any difference is published beside it. The within-tier contrasts, B against D and C against A, vary retrieval alone and carry no such confound.

3.2 What each condition gets, and where the remaining asymmetry lies

The most common objection to a benchmark published by the builder of one of its conditions is that the comparator was hobbled. Stated so it can be checked:

- **One context budget, shared by every condition that receives a block.** The budget is 100,030 bytes, realised within 0.9% across the cells for which per-cell records exist. Whatever a structured block spends, the file block cannot, which is what makes "the graph condition simply receives more context" an answerable objection rather than a rhetorical one. Condition A is the exception and is not bound by it: it searches agentically and is handed no context block at all, so the budget constrains what it can be given and not what it can find.
- **The same retry allowance.** Three attempts per cell for both tiers.
- **The same feedback between attempts.** Both tiers see a compile gate and an executed run of the repository's own suite, so both are shown the consequence of breaking behaviour the task did not ask them to change. The feedback arrives as an executed command result and never as a proof command in the prompt, so no condition is told where the task's code lives.
- **The same oracles, hidden from both.** Neither the fail-to-pass nor the pass-to-pass bundle is visible to any condition.

One asymmetry survives, and it favours the frontier tier. The cheap conditions run under a 131,072-token output ceiling, while the frontier conditions carry no harness ceiling at all. That figure is the result of a repair made during validation. The ceiling was originally format-dependent and in one format sat at 4,096 tokens, close enough to a correct answer to matter: one task's reference diff is 3,643 tokens, which leaves 453 of headroom on a patch already known to be right. We raised the cheap ceiling rather than lower the frontier one, on the view that a finite ceiling imposed on one tier alone is a handicap and the honest repair is to remove it where we can. Every reported cell ran under the raised ceiling. The per-task metadata in the released suite records the earlier 32,000-token budget in force when each task's edit format was chosen, and one task, M1P, is recorded there as running as a diff for that reason.

3.3 Draws, and what counts as solving a task

Every condition receives three independent draws on every task. A task counts as solved by a condition when a majority of its three draws pass the oracle.

Three draws on all four conditions departs from the pre-registration, which specified a single draw on the frontier conditions. Equalising the draw count removes an asymmetry between tiers and costs only compute. It also lowers the frontier conditions' apparent per-draw success relative to a single-draw measurement, because three draws expose variance that one draw hides. We prefer the fairer number and record the deviation here rather than in a footnote.

The full matrix is 11 tasks × 4 conditions × 3 draws, or 132 cells. It was executed three times, on separate machines with separate credentials, and returned the same tasks-solved outcome on each execution.

3.4 Grading, and the difference between a failure and a void

Each task carries two executable oracles: a *fail-to-pass* check that the intended behaviour now holds, and a *pass-to-pass* check that previously-passing behaviour still does. A patch is graded against a fresh checkout at the pinned commit, in a fixed order. Apply the candidate diff alone, run the pass-to-pass gate, then write the fail-to-pass bundle and run it. A candidate diff that touches the fail-to-pass paths is a protocol violation and scores zero. A mechanical check rejects the corpus if any proof command names a fail-to-pass class.

Every cell is recorded with exactly one outcome class:

Scored	`success`, `apply_failure`, `compile_failure`, `p2p_regression`, `f2p_failure`
Void	`infra_error`, `timeout`, `output_truncated`, `arm_identity_violation`, `graph_condition_unmet`, `protocol_violation`

A scored class is evidence about the model and counts against its condition. A patch that does not apply, does not compile, fails the fail-to-pass check, or breaks a previously-passing test is a measurement. A void class means the instrument failed to deliver the treatment, which makes it missing data rather than a result. Voids are excluded from every rate and retried under a fixed allowance. Scored cells are final and are never re-run, because re-running scored cells until they pass is how a harness manufactures a result. The rule this enforces, stated so it can be held against us: *a task may be rejected only for a cited mechanical defect, never because a model failed it.* In the reported matrix, two arm-D cells voided and both scored on retry, so no condition carries a surviving void and no contrast is withheld.

3.5 Execution order

Cells are executed in a seeded shuffle rather than in task-major order. A fixed arm-minor order gives every condition the same position in every task's sequence, so anything position-correlated lands on the same condition every time and becomes indistinguishable from a treatment effect. A cold build cache absorbed by whichever condition runs first would do it; so would a rate limit that bites late, or a retrieval service that degrades over a long run. The shuffle seed is recorded, so order is random with respect to condition and still exactly reproducible. One useful side effect is that a run halted early is an unbiased random subset of the design rather than a biased prefix.

3.6 Guards against endpoint drift

Four standing assertions run before any result is reported. Exactly one contrast may hold the `primary` role, and a test names which one, so the endpoint cannot drift by accretion in code after being fixed in prose. Every confounded contrast must carry an explicit confound string, and a

tampered set throws. A contrast whose two conditions differ in void rate by more than a pre-registered margin has every delta withheld rather than explained. Costs are recomputed from token counts, and a record whose stored total disagrees with the recomputed one throws rather than being reported.

4 The task suite

Eleven tasks are drawn from two open-source repositories at pinned commits: a Java web application (Spring PetClinic [11], Apache 2.0, at `f182358`) and a Python forum library (django-machina [7], BSD 3-Clause, at `5ecfad3`). Both are real projects with real test suites, and every task is derived from a defect or a missing capability genuinely present at the pinned commit rather than injected for the benchmark.

Statements are written from observable behaviour, meaning what a user sees go wrong, without reference to the patch that fixes it. Mechanical checks reject any statement naming a target file or type. This matters for a retrieval experiment: a statement that names its own gold file hands the answer to every condition and leaves the retrieval treatment nothing to measure.

4.1 The tasks

J1, boundary fault (Java, 1 file). An over-long animal name is not refused by the form. It reaches the persistence layer, the page breaks, and the operator gets an exception. The equivalent field for a person is already guarded, so the fix must match an existing convention rather than invent one.

J4, configuration (Java, 2 files). Every listing in the application shows five entries per page, fixed in code. Make the page size operator-configurable under a given property name, falling back to five when unset.

J5, boundary fault (Java, 10 files). A visit note longer than its storage column is passed to the database, which rejects it, producing a broken page and a stack trace instead of a form error. As with J1 the codebase already guards comparable fields elsewhere.

M1P, localisation (Python, 26 files). Two user-visible validation messages are written as plain literals, so they are never collected for translation and appear in English regardless of the active language. The correct patch touches every shipped translation catalogue, which makes it the widest task in the suite.

M2J, shared contract (Java, 3 files). Values pasted from another system arrive with surrounding whitespace. Records saved with padding cannot afterwards be found by searching for the name as normally written, and a padded species value is mishandled on submission. The fix changes a contract shared by more than one caller.

M2P, shared contract (Python, 2 files). A discussion held for moderation is correctly kept off the boards, but replies inside it are not: a user profile lists them and links into content that is supposed to be invisible.

M3J, ordering, negative control (Java, 1 file). A specialities list is sorted by capitalisation rather than alphabetically.

M3P, error contract, negative control (Python, 1 file). A lookup helper exists to return nothing when a record is absent. It is too permissive: it also returns nothing when the lookup itself was invalid, hiding genuine errors.

Y1, input validation (Python, 2 files). Attachments have a count ceiling but no size ceiling, so a single upload is unbounded and nothing stops it before it reaches disk. Refuse an oversize file at the point it is offered, and say what the limit is.

Y2, permission leak (Python, 2 files). A profile page shows two totals above a list of recent activity. The list respects the viewer's read permissions; the totals do not, and silently disclose counts from boards the viewer cannot see.

Y3, error contract (Python, 1 file). Records of attached files can outlive the files themselves. When a reader follows a link to one that is gone, the request fails with a server error instead of reporting absence.

M3J and M3P are **negative controls**, single-file tasks with no ripple beyond the file that carries the defect. They are included because a retrieval treatment should show little advantage where there is nothing to traverse, and a corpus that reports an advantage everywhere is more likely measuring something other than retrieval.

4.2 What the construction protocol caught

A benchmark's credibility rests on the gates it survived, and gates that are declared rather than executed are worth nothing. Each task passed nine gates in sequence. Every one of them found something, and several found defects in work an earlier gate had blessed.

Table 1: The 2×2. Tasks solved by majority of three draws, out of 11, with total inference spend for all 33 draws of that condition. Frontier costs are modelled at published list rates; cheap-tier costs are metered.

	Grep		**Graph retrieval**	
Frontier model	9/11	\$25.3742	9/11	\$25.2953
Cheap model	2/11	\$0.4317	9/11	\$0.2396

Gate	**Defends against**	**What it caught**
Premise audit	The behaviour already works, so any patch that compiles scores	Tasks whose complained-of behaviour reproduced only under conditions the statement did not describe; those tasks were rewritten or dropped before authoring continued
Blind statements, mechanical leak checks	The statement names its own gold file	The comparator tension in §6
Two reference solutions, both executed	A completeness rule satisfied by an alternate nobody ran	One alternate violated its own task's statement; on another, the two accepted solutions disagreed
Script-derived metadata	Hand-typed metadata drifting from the patch	Nothing
Origin-only ripple probes	Over-crediting blast radius	Nothing
Adversarial degenerate rounds	An oracle satisfiable by a hard-coded constant	Three rounds, each still finding holes; two survive on one task, both recorded with reasons
Output-budget check	The correct answer cannot fit the ceiling	One task's whole-file answer is 132k tokens against the then-current ceiling, so it runs as a diff
Substrate decontamination	The index contains files not present at the pinned commit	The coverage caveat in §6
Per-condition identity assertions	The cell did not run as the condition it claims	A process-inheritance defect that had every cell executing as one condition while each record stamped a distinct one

Twelve tasks were authored and one was withdrawn for a cited mechanical reason: its patches rewrite an error code the repository's own suite asserts on, so a behaviourally correct candidate records a regression in every condition. We report the withdrawal rather than only the surviving count.

5 Results

5.1 The interaction

The clearest result in the matrix is not the diagonal but the difference between the two within-tier contrasts, and it is clean in a way the diagonal is not. Within a tier, the model, the loop and the draw count are held fixed, and only the context source changes.

Within the cheap tier, replacing grep with graph retrieval takes the model from 2/11 tasks to 9/11. The direction is unanimous, with seven tasks won and none lost. Within the frontier tier the

Table 2: Per-condition outcome over all 33 draws. Success rate is over draws; tasks solved is majority-of-three. No condition recorded a surviving void. Cells recorded without an explicit per-draw count are entered as 3/3 for a solve and 0/3 for a failure; this affects the success column only, and the tasks-solved endpoint is unchanged under either reading.

	Condition	Draws	Wins	Success	Total $	$/solved
A	Frontier + grep	33	25	75.8%	25.3742	2.81936
C	Frontier + graph	33	25	75.8%	25.2953	2.81059
B	Cheap + graph	33	23	69.7%	0.2396	0.02662
D	Cheap + grep	33	8	24.2%	0.4317	0.21586

Table 3: Per-task outcome. Each entry is draws won out of three; **bold** marks a task solved on a majority.

Task	Class	A	C	B	D
J1	Boundary fault	0/3	0/3	**3/3**	1/3
J4	Configuration	**3/3**	**2/3**	**2/3**	1/3
J5	Boundary fault	**2/3**	**3/3**	**2/3**	0/3
M1P	Localisation	**3/3**	**3/3**	1/3	1/3
M2J	Shared contract	0/3	**3/3**	**2/3**	**2/3**
M2P	Shared contract	**3/3**	**3/3**	0/3	0/3
M3J	Ordering	**3/3**	**2/3**	**3/3**	0/3
M3P	Error contract	**3/3**	**3/3**	**3/3**	1/3
Y1	Input validation	**3/3**	**3/3**	**2/3**	**2/3**
Y2	Permission leak	**3/3**	**3/3**	**2/3**	0/3
Y3	Error contract	**2/3**	0/3	**3/3**	0/3
Solved		**9**	**9**	**9**	**2**

same substitution produces 9/11 against 9/11, one task traded in each direction. The interaction is seven tasks (Figure 1).

The two negative controls qualify this. On M3J and M3P, single-file tasks with no ripple, the cheap condition still solves 3/3 with graph context against 0/3 and 1/3 with grep. If retrieval only helped where traversal helps, these tasks should have shown little difference. They did not, so the effect is not attributable to traversal alone, and part of the cheap tier's advantage plausibly comes from being handed the right file at all.

5.2 The diagonal

On the primary endpoint, B and A each solve 9 of 11 tasks, at $0.02662 and $2.81936 per solved task, a ratio of 105.9×. Across all eleven tasks, seven are solved by both, two by A alone (M1P, M2P), two by B alone (J1, M2J), and none by neither.

The two conditions therefore fail differently rather than one dominating. A loses J1, a single-file boundary fault, and M2J, a contract change with three dependent files. B loses M1P, whose correct patch spans twenty-six translation catalogues, and M2P, a permission-visibility change across a shared type. M1P looks like a capability limit rather than a retrieval limit: the cheap condition locates the relevant code and then fails to express an edit of that breadth.

Table 4: Paired contrasts across all 11 tasks. "Discordant" counts tasks solved by one condition and not the other, in each direction; it is the quantity that carries a paired comparison.

Contrast	What varies	Solved	Discordant
B vs D	Retrieval, within the cheap tier	9 vs 2	7 / 0
C vs A	Retrieval, within the frontier tier	9 vs 9	1 / 1
B vs A	Diagonal: tier, retrieval and loop	9 vs 9	2 / 2
C vs B	Model tier, within graph retrieval	9 vs 9	2 / 2
A vs D	Model tier, within grep	9 vs 2	8 / 1

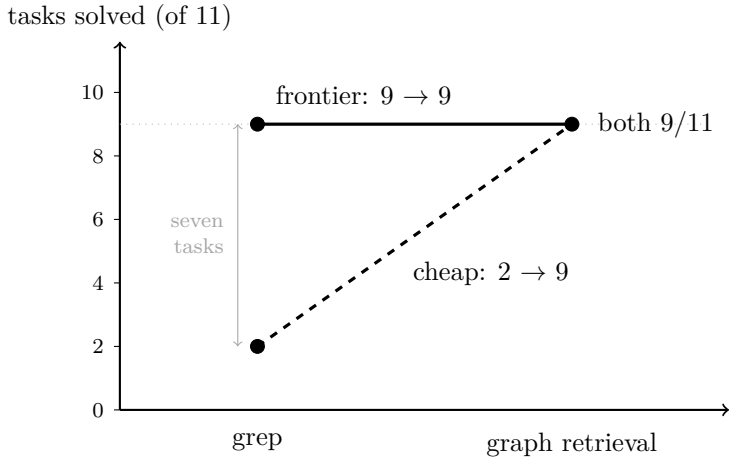

Figure 1: The interaction. Switching context source moves the cheap tier from 2/11 to 9/11 and the frontier tier not at all. Within each line the model, the loop architecture and the draw count are held fixed, and only the context source changes.

We state plainly what this does and does not support. Two discordant pairs in each direction is a balanced result, and a paired comparison at this sample size cannot resolve a difference of practical size in either direction. It establishes that no difference was detected, not that the conditions are equivalent, and by the same token it does not place the frontier condition ahead. We decline to round a null result to parity. The defensible sentence is that the two conditions solved the same number of tasks on this corpus, at costs differing by two orders of magnitude, and that the cheap condition's failures are concentrated in tasks requiring wide, uniform edits.

5.3 How each condition fails

Outcome classes separate the two cheap conditions further. Condition D's dominant scored failure is a pass-to-pass regression, 13 of its 28 cells, against 7 of 29 for condition B on the same model and the same loop. A cheap model given poor context does not only miss the fix; it edits code it should not have touched and breaks behaviour the task did not ask it to change. The frontier conditions record one such regression apiece. This is the failure mode a reader deploying a cheap tier should price.

5.4 Cost, and why the ratio is a floor

Frontier costs are modelled from token counts at published list rates; cheap-tier costs are metered. The distinction is material and we do not blur it. The frontier system runs against a subscription that emits no per-call price, so a metered figure for it does not exist, and the modelled figure is the honest substitute. Indexing is a separate line item and is not folded into any per-task cost.

The direction of the residual error is measurable, and it runs against our own headline. Across the 22 frontier cells carrying per-cell records, the executing engine's own reported cost exceeds the modelled figure by a median of 1.22× for condition A (range 1.10 to 1.32) and 1.26× for condition C (range 1.17 to 1.36). The modelled basis therefore understates frontier spend by roughly a fifth to a quarter against the engine's own accounting, and 105.9× is a lower bound on the cost ratio. We report the modelled figure anyway, because it is the one a reader can derive from a published rate table.

The cost comparison is an accounting consequence of that rate table and the two loop architectures. It is not a hypothesis this experiment could falsify. The quantities a reader can use are the ratio per solved task and the observation that the frontier conditions spend their advantage on tokens consumed by search the graph condition does not need to perform.

5.5 An exploratory note on retrieval quality

This subsection is exploratory and should be read that way. It was not part of the pre-registered design, the study is not powered to settle it, and nothing in the results above depends on it. We include it because it points at a plausible account of the interaction, and because leaving a suggestive record unpublished is its own kind of selection.

Cells carry a record of whether the context block the model received contained the file or files the reference patch modifies. Aggregating that record gives Table 5.

Table 5: Exploratory. "Gold served" counts cells in which at least one file modified by the reference patch was inside the delivered context block; recall is over all gold files across all cells of that condition. Cell counts are unequal and small, no correction is applied, and the right-hand column for condition C rests on three cells. Condition A searches agentically and is handed no context block, so it has no served-context record by construction.

	Condition	Cells	Gold served	Solved | served	Solved | not served
B	Cheap + graph	29	24 (40.2% recall)	18 / 24	0 / 5
C	Frontier + graph	11	8 (25.5% recall)	6 / 8	3 / 3
D	Cheap + grep	28	5 (14.3% recall)	2 / 5	2 / 23

The two right-hand columns point in opposite directions between tiers. Where the retrieval missed, the cheap condition solved none of those cells and the frontier condition solved all three such cells of its own. If that pattern held at scale it would suggest that what model scale buys on these tasks is partly the ability to recover from being handed the wrong context, paid for in tokens rather than in tasks, and that a cheap single-shot model has no such recourse. It would also explain why the negative controls behaved as they did, since the operative variable would be whether the right file is inside the budget at all rather than whether traversal put it there.

The design cannot settle any of that. Three cells cannot carry a claim, the cell counts across conditions are unequal, and recall is in any case not sufficient on its own: J5 had a gold file served in 3/3 cells and was still lost 0/3 by the cheap condition, a ten-file task where locating the code

was never the constraint. Testing this properly needs a design built for it, which this one was not.

6 Threats to validity

Threat	**Status**
Corpus size	Eleven tasks. Running the matrix three times establishes that the outcome is stable under repetition; it does not widen the corpus. Three replications of eleven tasks is still eleven tasks, and the interval on the diagonal is set by the four discordant pairs rather than by the 132 cells. No significance test on this design is decisive.
Confounded primary endpoint	Tier, context source and loop architecture all differ across the diagonal. This is why the off-diagonal cells are run and published, and why we lead with the within-tier interaction.
The grep comparator and the statement protocol are in tension	Because statements are written so as not to name their target files, the vocabulary shared between a statement and its gold file is small, and ranking by term match is correspondingly poor. This is a consequence of blind statement authoring rather than a concession made to favour the graph conditions, and it is a finding about how a corpus of this kind must be built. Measured before any experimental cell ran, condition D is served no gold file on 8 of 11 tasks and never the origin file; its slots go to translation catalogues, build wrappers, a licence file and a stylesheet. We strengthened it first, adding a second retrieval round with IDF-weighted expansion, the shape the strongest published grep-style baseline for code takes, and the composition did not move, because round two can only expand from what round one returned. The B against D contrast therefore partly measures the statement-writing protocol and not only the retriever, and the diagonal against A, which searches agentically and can re-query, is the primary endpoint for that reason.
Oracle looseness	Adversarial rounds found oracles that accept a patch not doing the work. Three rounds were run and each still found something, so the rounds had not converged when the corpus was sealed. We hardened what a fixture or assertion change could close and recorded the rest. The direction matters. A loose oracle inflates every condition's success rate equally, making it condition-independent measurement error, and where it touches a contrast it biases *toward* the null, because a task solvable without its dependent files is a task a weaker retriever can also solve.
Output pressure is not equalised by equalising input	Conditions share a byte budget. An effect we did not anticipate is that the fix moves pressure downstream: a condition handed more *files* within one budget is likelier to answer by rewriting several of them and to run past the output ceiling while doing so. Those cells are recorded as voids rather than failures, and they fall disproportionately on the cheap grep condition.

Threat	Status
Per-cell artifacts	Cost figures for the reported matrix are per-condition totals rather than per-cell records, because the per-cell artifacts did not survive a storage failure on the executing machines and the totals were recovered from a surviving artifact. We cross-checked them against per-cell costs measured independently on separate hardware; implied per-cell cost agrees within 1.5%, 0.9%, 1.6% and 5.6% for the four conditions. Per-cell wall-clock and token counts did not survive, and this paper reports none rather than estimating them. The per-cell figures in §5.3 and §5.5 are drawn from the execution for which per-cell records exist; their cell counts are stated there, and the tasks-solved endpoint does not rest on them.
Imputed per-draw counts	Task cells recorded without an explicit per-draw count are entered as 3/3 for a solve and 0/3 for a failure. This affects the per-draw success column of Table 2 and the corresponding entries in Table 3. It does not affect the tasks-solved endpoint, which is unchanged under either reading, and entries showing a split draw (2/3, 1/3) are measured.
No no-context floor	No condition runs with no retrieval at all, so nothing is anchored to zero and we cannot bound how much of any condition's success comes from prior familiarity with these repositories.
Index coverage	One index of the Python repository, at 74.58% file coverage, with repeat-index divergence unmeasured rather than zero. Incomplete coverage can only have cost the graph conditions retrieval they would otherwise have had, so the reported graph results are obtained under an index that is not complete.
Scope	Two repositories, two languages, one application domain. Nothing here speaks to codebases substantially larger than these, to languages with different module conventions, or to repositories whose test suites are weaker than the ones we relied on for grading.
Single author, and the author builds one condition	Reduced by upstream-derived defects, blind statements, derived metadata, oracles proven in both directions, adversarial rounds, and by publishing the corpus (§8). Reduced, not eliminated. Independent replication is the only real answer and we invite it.

7 Related work

Benchmarks for repository-level code work are easy to build wrong, and the errors are systematically flattering. An audit of SWE-bench found that 32.67% of successful patches had their solution present in the issue text and 31.08% passed tests too weak to distinguish a correct patch from an incorrect one; removing those instances dropped one agent's resolution rate from 12.47% to 3.97% [1, 5]. Leaked solutions and permissive oracles are the two failure modes our blind statements with mechanical leak checks, and our adversarial degenerate rounds, are built to target. They are also why we report that the degenerate rounds have not converged rather than declaring the oracles sound.

Graph-structured retrieval for code has been explored as a way to supply models with dependency context rather than whatever a text search happens to return [12, 8]. A more recent line questions how much of that benefit survives against well-tuned grep baselines, finding that grep generally outperforms vector retrieval and that the result depends heavily on the surrounding agent

harness [10, 14]. Our result is consistent with both positions once the tier interaction is taken into account, and it suggests why the harness dependence they report should be expected. An agentic harness can re-query, so it converts a retrieval miss into token spend, while a single-shot cheap model converts the same miss into a failed task. A grep baseline is competitive for a sufficiently capable agentic searcher and is not competitive for a cheap single-shot one, so any evaluation that fixes one tier will report one half of that.

Two findings bear directly on ours. Work on retrieval utilisation across model scale [9] reports that models of 7B parameters and below fail to extract the correct answer from oracle retrieval on the large majority of questions they cannot already answer, which is a utilisation bottleneck rather than a delivery one. The two bound each other. Our cheap condition is far larger than 7B and clears the utilisation bar on nine of eleven tasks, and its two losses (M1P, M2P) look like utilisation failures rather than retrieval failures, which is what a utilisation ceiling predicts. Separately, work on a structural codebase index inside a coding agent reports that the index lands at a lower cost per solved task than agentic grep, and frames the deployment question as whether the workload contains multi-file changes where structural ranking pays [3]. We reach a compatible cost conclusion by a different route, and would qualify the framing, since on our corpus the structural advantage appears even on single-file negative controls. Work on persistent codebase memory delivered through tool interfaces points the same way [13].

8 Artifact availability

The task suite is released as `Koragraph_test_suite_1` [6]. For each of the eleven tasks it contains the statement as given to every condition, the pinned commit, both oracle commands, the fail-to-pass test files with recorded checksums, two independently authored reference solutions, and the executable degenerate patches an oracle must reject. The grading protocol and the failure-versus-void taxonomy are included, so a task can be graded exactly as it was graded here. Both upstream repositories carry permissive licences (Apache 2.0 and BSD 3-Clause) and the released patches are redistributed under them with attribution. Neither upstream project endorses this suite.

What is released is the corpus and its oracles, not the retrieval implementation. This is the same boundary drawn in §2. The reproducible unit of this experiment is the interface, meaning a task, a pinned repository, a byte budget, and a context block, and any retriever placed behind that interface is running the same experiment. A reader who disputes the result has what is needed to re-measure it against a retriever of their choosing.

Eleven tasks is a small corpus, and we prefer a small one that carries executable oracles, adversarial degenerate patches and a stated construction protocol to a larger one that carries none of them. The suite is built to grow: contributions clearing the same nine gates are welcome, and independent replication is the reason it is public.

9 Conclusion

Structured context substitutes for model scale in one tier and not the other. Given graph-derived context, a cheap model went from 2/11 to 9/11 tasks on this corpus, winning seven tasks and losing none, while the same substitution moved a frontier agentic system not at all. On the pre-registered diagonal the cheap-plus-graph condition matched the frontier-plus-grep condition at 9/11 tasks each, for one hundredth of the cost per solved task, while failing on a different pair of tasks.

The result we would defend is the interaction, because it is the one measured without confounds. The diagonal we report with its cost, its decomposition and its statistical limits attached, and not

as a claim of equivalence. Whether the substitution holds at a larger corpus, on larger repositories, and across more languages is the obvious next question, and this design, a factorial with the off-diagonal cells actually run and a published corpus any retriever can be measured against, is what makes that question answerable rather than rhetorical.

References

[1] Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. SWE-Bench+: Enhanced Coding Benchmark for LLMs. *arXiv preprint arXiv:2410.06992*, 2024.

[2] Anthropic. Claude Code. `https://claude.com/claude-code`, 2026. Agentic coding CLI; the harness for the frontier conditions. Accessed August 2026.

[3] Ishaan Bhola, Adithyan Krishnan, Sravanth Kurmala, and Mukunda NS. Code Isn't Memory: A Structural Codebase Index Inside a Coding Agent. *arXiv preprint arXiv:2606.22417*, 2026.

[4] DeepSeek. DeepSeek V4 Flash (`deepseek/deepseek-v4-flash-0731`). `https://openrouter.ai/deepseek/deepseek-v4-flash-0731`, 2026. Served through the OpenRouter gateway. Accessed August 2026.

[5] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.06770.

[6] Akhil Katakam. Koragraph_test_suite_1: eleven repository-level code tasks with executable oracles. `https://github.com/agilkatakam/Koragraph_test_suite_1`, 2026. The task suite released with this paper.

[7] Morgan Aubert and contributors. django-machina: a Django forum engine. `https://github.com/ellmetha/django-machina`. BSD-3-Clause. Pinned at commit `5ecfad3`.

[8] Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. RepoGraph: Enhancing AI Software Engineering with Repository-level Code Graph. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.14684.

[9] Sanchit Pandey et al. Can Small Language Models Use What They Retrieve? An Empirical Study of Retrieval Utilization Across Model Scale. *arXiv preprint arXiv:2603.11513*, 2026.

[10] Sahil Sen, Akhil Kasturi, Elias Lumer, Anmol Gulati, and Vamse Kumar Subbiah. Is Grep All You Need? How Agent Harnesses Reshape Agentic Search. *arXiv preprint arXiv:2605.15184*, 2026.

[11] Spring Projects. Spring PetClinic. `https://github.com/spring-projects/spring-petclinic`. Apache-2.0. Pinned at commit `f182358`.

[12] Hongyuan Tao, Ying Zhang, Zhenhao Tang, Hongen Peng, Xukun Zhu, Bingchang Liu, Yingguang Yang, Ziyin Zhang, Zhaogui Xu, Haipeng Zhang, Linchao Zhu, Rui Wang, Hang Yu, Jianguo Li, and Peng Di. Code Graph Model (CGM): A Graph-Integrated Large Language Model for Repository-Level Software Engineering Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2505.16901.

[13] Martin Vogel, Falk Meyer-Eschenbach, Severin Kohler, Elias Grünewald, and Felix Balzer. Codebase-Memory: Tree-Sitter-Based Knowledge Graphs for LLM Code Exploration via MCP. *arXiv preprint arXiv:2603.27277*, 2026.

[14] Baoyi Wang, Xingliang Wang, Guochang Li, Chen Zhi, Junxiao Han, Xinkui Zhao, Nan Wang, Shuiguang Deng, and Jianwei Yin. Better Call Grep: Evaluating and Improving Grep-Like Lexical Retrieval for Repository-Level Code Completion. In *Proceedings of the 35th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2026. arXiv:2601.23254; earlier versions titled "GrepRAG: An Empirical Study and Optimization of Grep-Like Retrieval for Code Completion".