Memglide: Measuring the Cost and Latency of Bounded Repository Context for Coding Agents

Sree Charan Reddy Kailasam
srcharan014@gmail.com

***Abstract*—Memglide prepares bounded repository context using lexical and static syntax evidence and can supply verified public commands when source selection abstains. We compare its installed commands-first workflow with pass-through Codex on an amended fixed panel of 28 author-designed feature tasks, with two repeats per method: 112 native sessions. Both methods pass 54 of 56 frozen strict assessments. Mean token-tariff-equivalent cost is $0.16094 for Codex and $0.15316 for Memglide, an observed 4.84% reduction. Mean measured workflow time is 59.59 and 60.29 seconds, an observed 1.17% increase. Paired task-level 95% intervals for Memglide minus Codex are [-$0.02524, +$0.00968] and [-3.02, +4.42] seconds; neither superiority test passes. Native execution is 2.06 seconds shorter on average, offset by 2.75 seconds of added overhead. All 128 pooled comparison sessions have valid measurement and grading records; 16 exposed-task sessions remain descriptive. Evaluation amendments followed partial collection, so inference is nominal rather than fully prospective. The study identifies an overhead bottleneck without establishing general savings, speed superiority or quality equivalence. Token-price estimates are not invoices.**

***Index Terms*—coding agents, repository context, source retrieval, reproducibility, software engineering**

I. Introduction

Coding agents must locate source and test commands before making a change. Advance context may reduce this work, but selection, verification and added prompt text also have costs. We measure the complete workflow, retaining failed attempts and checks on existing behavior.

Memglide prepares bounded repository context around an installed coding agent. Lexical and static syntax evidence select complete current files; the opt-in commands-first workflow can supply verified public commands and baseline counts even when source selection abstains. Preparation makes no auxiliary model or embedding request. The external coding model interprets the task, edits code and runs checks.

Development included 30 tuning rounds on three tasks and subsequent exploratory comparisons. The latest study compares pass-through Codex with the installed commands-first candidate, including demand-driven public verification. Its primary cohort has 28 tasks, two repeats per arm and 112 sessions. Another 16 sessions on previously exposed tasks remain descriptive. Private grading follows collection; references, oracles and revealing plaintext logs are sequestered during collection.

The tasks are AI-assisted feature additions to small public libraries, unlike SWE-bench's real GitHub issues [1]. The primary panel was fixed before its coding sessions, but evaluation infrastructure was corrected after partial collection. Original measurements, failures and amendments remain in the evidence. This is an amended fixed-panel comparison with nominal inference, not an untouched prospective confirmation.

We ask when source, verified commands, both or neither are delivered; how the policies compare under feature, public-regression and source-integrity checks; and how token-tariff cost and workflow time change when fresh indexing and public verification are counted.

The contributions are an implementation separating bounded source selection from verified command assistance, an evaluation method recording activation and measurement provenance, and a development record retaining unsuccessful approaches and collection repairs. The study changes context preparation, not model weights; it establishes neither memory-reuse benefits nor benchmark-leading coding performance.

II. Related work

RepoCoder combines similarity retrieval with a pretrained code model in an iterative retrieval-generation pipeline [2]. RepoBench evaluates retrieval, completion and their combination, including cross-file dependencies [3]. These studies distinguish retrieved evidence from its use. Memglide evaluates functional patches and preserved public regressions on small feature tasks rather than completion similarity.

Liu et al. found that performance often declined when relevant information appeared in the middle of long contexts in multi-document question answering and key-value retrieval [4]. LLMLingua uses coarse-to-fine prompt compression and model alignment [5]. Memglide selects complete files from lexical and static syntax evidence and can abstain without an auxiliary model call. We do not compare performance with these methods.

SWE-agent studies an agent-computer interface for software engineering [6]; Agentless separates localization, repair and patch validation [7]. Memglide prepares context before the installed agent investigates and edits. Our author-designed panel differs from SWE-bench [1] and supplies neither a SWE-bench score nor a direct comparison with these systems.

III. Architecture and evidence boundaries

Memglide records the source and policy used to prepare bounded context. Lexical evidence, static syntax and limited dependency expansion select current files. The coding agent remains responsible for interpreting and implementing the task.

Interactive clients use a command-line interface, loopback HTTP API or MCP STDIO server backed by indexing, retrieval, SQLite and explicit session memory. The native memglide-codex launcher calls preparation libraries directly. This study evaluates its opt-in commands-first candidate at commit 3c3f82b with the setup planner below (Fig. 1). Interactive retrieval, the dashboard and memory reuse are outside the measured path.

Repository registration resolves the Git root and records project state. The SQLite FTS5 index stores searchable source, file hashes and metadata, symbol locations and signatures, and conservative dependency edges. Python structure comes from the standard AST; JavaScript and TypeScript use an isolated Tree-sitter worker. Incremental refresh checks current files before using indexed evidence. Dynamic imports, generated code and runtime dispatch can leave static dependencies unresolved, limiting what the selected context represents.

Interactive retrieval combines identifier and path matches, FTS5 evidence and graph expansion. Session memory stores caller-submitted observations with provenance, timestamps, source versions and expiry. Context passports identify hashes and omissions. The native selector uses a separate deterministic algorithm and does not automatically inject session observations; cross-session reuse is unmeasured.

Prelaunch gates check enablement, existing supplied context, project approval and task eligibility. Failed initial gates preserve the task. The v4 policy securely refreshes the index and screens production files, excluding known tests, examples and demonstrations. Unsupported relevant files, partial indexing or an overfull frontier cause source abstention. A completed content-selection decline may still permit verified commands. Source and command activation are recorded separately.

Supported candidates are parsed into runtime or declaration syntax units. Query weights are $w(t) = \log\Big(1 + \frac{N+0.5}{\mathrm{df}(t)+0.5}\Big)$, where N is the bounded file candidate count and df(t) counts files whose eligible unit body or header contains t. Sufficient evidence requires an explicit current anchor or two distinct clean matches, including one body match. Explicit and sufficient evidence receive priority; remaining body, header, path and documentation weights have coefficients 3, 2, 0.25 and 0.05. Scores rank evidence rather than calibrated confidence. A competitive alternative causes abstention unless the leader meets body-evidence dominance.

After accepting an anchor, Memglide reads its complete file and at most three resolved direct production helpers, checking current hashes and evidence. Full source means these selected files, not the entire repository or transitive closure. Final handoff revalidates competitors, source, approval and ignore policy; a changed proof prevents reuse. Source text remains project data for the agent to interpret.

Default indexing allows 20,000 files, 100 MiB total and 512,000 bytes per file. V4 queries allow 4,000 UTF-8 bytes and 64 normalized terms of at most 128 characters. Screening allows 100 files/2 MiB and syntax analysis 24 files/2 MiB. Worker limits are five seconds, 200,000 nodes and 1,000,000 output bytes. Added context allows 12,000 bytes and a ceil(bytes/3) planning proxy of 4,000, not measured tokens. Repeated freshness checks mean these are phase bounds, not whole-workflow I/O or memory bounds.

Public-command assistance requires separate authorization. The caller supplies atomic arguments and allowed environment updates. A verifier binds a bounded public-text tree and declared executables, executes checks, records exit status, positive behavioral counts and log hashes, then rechecks tree, policy and tools before storing a receipt outside the project. The caller retains its trusted hash. Tree limits are 20,000 entries, depth 12 and 16 MiB; receipts allow 4 MiB and 16 commands, with 256 MiB per executable fingerprint. These distinct limits do not form a combined resource budget.

The final delivery modes are preloaded (complete selected source, optionally with a command card), commands_only (a card with no selected source and a retained decline proof), and passthrough (neither source nor commands, preserving original task bytes).

Command-only assistance requires a completed v4 content-selection decline, a refreshed index and no selected files. Eligible declines include ambiguity, weak production evidence, unsupported competitors and bounded candidate, dependency or context limits. Caller skip gates, over-limit queries, missing approval, and freshness or policy failures do not permit a card. Source thresholds remain unchanged.

The launcher checks the receipt against the caller's trusted hash and revalidates public-tree, approval, policy and tool bindings. The card lists atomic commands, allowed environment settings and observed baseline results. Its instruction asks the agent to use that baseline without rerunning it before editing, then execute the listed checks after changes and report full results. This records baseline facts, not agent compliance or future correctness.

Stale or unavailable optional facts, or a card exceeding unchanged whole-context caps, leave eligible source preparation available; a command-only route returns to the original task. Invalid explicitly pinned receipt input is rejected on an active path. Before execution, the launcher repeats selection, validates the source or decline proof, reconstructs the exact payload and rechecks provider bindings. Empty source lists do not exempt public-tree or tool freshness.

After indexing, a demand-driven setup planner calls installed preparation and validation functions and checks command-only eligibility. It skips public checks and provider creation only when no card could attach. Eligible routes retain fresh binding, verification, cleanup and manifest validation. Typed initial binding unavailability may omit the optional provider; failed checks or later integrity failures remain errors. The launcher repeats routing and final validation. The planner is not a reusable receipt, and the shipping launcher does not silently run project checks. All measured planning, indexing and verification time is included.

Local-first refers to index and state placement; selected source reaches an external coding model. Readers enforce

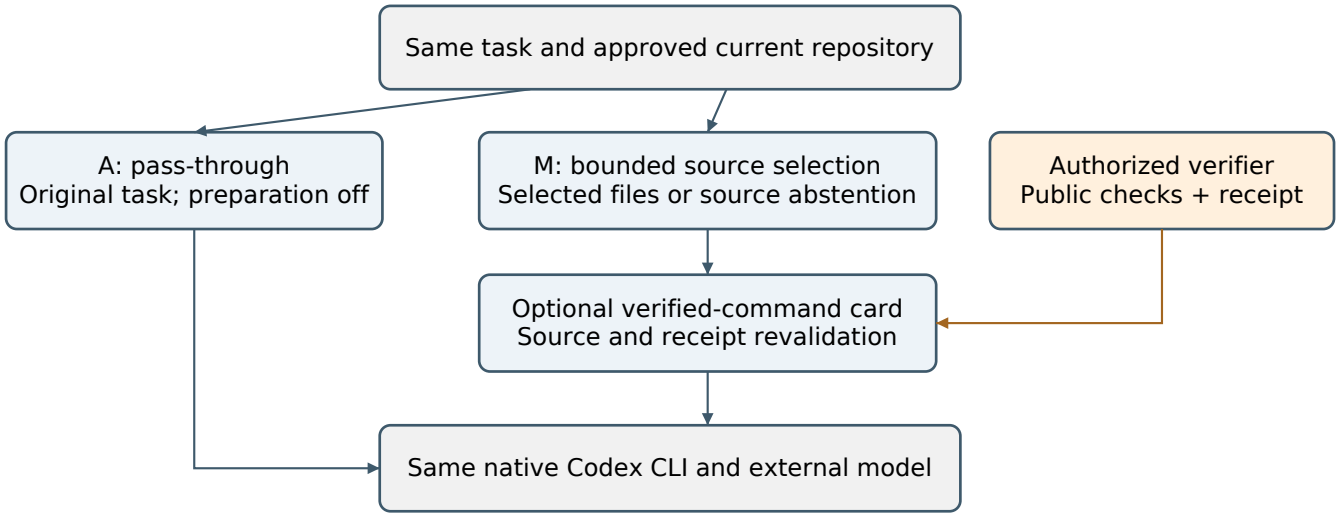


Fig. 1. Evaluated A/M workflow. A disables preparation. M can supply source, verified commands alone, or neither. Public verification is separately authorized and included when eligible; interactive memory and dashboard services are unmeasured.

containment and bounded regular UTF-8 reads, reject symlink traversal and heuristically filter sensitive names/content. Hashes prove equality at checked reads, not semantic safety or universal secret detection. Receipts cover eligible text and declared executables, not all dependencies or environment state; runner bounds are not operating-system isolation.

IV. Evaluation Design

A. Task panel and conditions

The v7 experiment started with 36 author-designed feature tasks in 36 pinned public repositories: 18 Python projects and 18 JavaScript/TypeScript projects. Tasks were selected for executable upstream checks, not sampled from historical issues. Several repositories share maintainers or development conventions. The model's prior familiarity with their source is unknown. AI agents assisted with task writing, feature oracles, reference implementations, deliberately incorrect controls and review. Separate authoring roles and control outcomes provide checks on the evaluation; they are not independent human replication. Public task inputs, dependency profiles, commands and original protected-file hashes were fixed before collection. Each task had baseline, reference and incorrect-implementation controls. Private tests remain finite specifications, and the wording ambiguity discussed in the results limits one task's rubric.

Both arms invoke the installed candidate at commit 3c3f82b. A uses explicit off/pass-through mode. M uses commands-first with source, command-only or pass-through delivery. All assigned tasks, including abstentions, remain in the comparison. Delivery modes are policy outcomes rather than separately randomized arms.

B. Allocation and the fixed primary cohort

The original schedule contained two arms and two repeats per task, or 144 unique assignments. Seed 20261101 determined a sequential, balanced first-repeat schedule with 18 A–M and 18 M–A task orders. Repeat two reversed the full first-repeat task/arm sequence. Every coding session used a fresh project copy, local native state and thread.

The first 15 assignments exposed eight tasks before an infrastructure stop. Assignment 0016, on an already exposed task, then failed locally before the native bridge because its copied executable lacked execute permission. Its recorded usage remains unknown. Before any of the remaining tasks had a model session, the primary cohort was defined as the 28 tasks outside that exposure prefix: 15 Python and 13 JavaScript/TypeScript projects. Each contributes both arms twice, giving 112 sessions and 28 analysis clusters. This membership rule used exposure, not observed savings, speed or grading outcomes.

The same 112 primary assignments occupy slots 0017–0128. The final 16 slots are the remaining second-repeat sessions on the eight exposed tasks; these are auxiliary descriptive evidence. The completed 128-record comparison pool therefore spans 36 tasks with unequal repeat coverage. Statistical inference uses the 28 complete task clusters; the other 16 records are descriptive. No primary task was removed because retrieval abstained, coding failed or a measurement appeared unfavorable. The primary cohort retains 13 A–M and 15 M–A first-repeat orders.

C. Collection, amendments and isolation

All coding sessions requested gpt-6.1-sol, low reasoning effort and the default service tier through Codex CLI 0.159.2. This fixes the requested alias, not a provider backend or model-weight snapshot. Both arms received the same task and common developer contract, response schema, tools and network-disabled workspace-write configuration. Project-document injection was disabled in both. Actual prompt bytes and native arguments were checked against installed zero-model handoff

receipts. These are coding-agent codex exec sessions; local sandbox processes used for controls or grading do not themselves make additional coding-model requests. Neither process counts nor agent commands identify the provider's internal inference count.

Collection required several versioned amendments. After 25 primary measurements, a broad protected-file rule incorrectly classified published production files as tests or build material. A baseline-only audit of all 36 repositories certified five exact production code/type paths. It used original package metadata and public module documentation, not candidate patches, private grades or resource contrasts. The correction was applied uniformly to all 128 records while preserving the original raw flags. Feature assertions and reference cases were unchanged. Ordinary edits to owned protected tests, documentation or configuration still imply strict coding failure; unsafe paths, Git/shared-dependency changes and evidence corruption remain hard integrity failures.

After 52 measurements, the usage parser stopped on an intermediate reconnect warning even though its turn completed. The pinned CLI source distinguishes retry notifications from fatal failures but renders both through the same JSONL error shape. A separate, source-verified adapter accepts only the narrow retry constructions within one fresh, completed thread/turn, with matching full token counters, telemetry, process identity and final-output evidence. It recomputes every pooled record and preserves the original parser verdict and events. Arbitrary errors or unfinished turns remain invalid. The resulting estimand is native-reported completed-thread usage, not proof that every underlying provider attempt is accounted for in an invoice.

A subsequent slot-0069 CLI startup failed on a missing local output-schema file. Pinned source established that this exit precedes model-turn creation. That extra process remains a separate operational failure with its original null usage; only its separate ledger records source-proven zero generation requests and zero model-generation tariff. The canonical model assignment proceeded after deployment repair. A declared sensitivity charges the full failed startup duration to that M assignment. Operator diagnosis and repair time was not measured. Overall accounting retains 144 unique master assignments, 145 allocation attempts, 143 coding-model turns and 144 native CLI processes. These counts exclude separate development and connectivity probes.

The final pool combines the first 52 measurements with the remaining 76 canonical sessions. There were no retries of measured coding trials, replacement tasks, additional outcome-driven repeats or significance-based stopping. Original stopped states and protocol decisions remain unchanged and cannot acquire a retrospective confirmatory pass. Because 25 measurements preceded the source-role correction and 52 preceded the parser amendment, the final study is an amended fixed-panel evaluation. Its local hashes and analysis freezes are audit commitments, not a public preregistration or a fully prospective confirmation.

Private grader, reference and control plaintext was absent during model collection. Authenticated AES-GCM ciphertext and per-file commitments could remain locally, while keys were held outside the model executor. A historical key-loss incident required reconstruction against original hash commitments before the primary cohort began; later versioning changed source-protection metadata and deployment paths, not feature assertions. The final private bundle was restored and byte-verified after collection closed. This is a documented plaintext-sequestration procedure, not proof of complete operating-system isolation.

D. Timing and strict assessment

Timing begins after identical public-source copying and Git-baseline creation, before registration, indexing, eligibility planning or verification. It ends after native process-group cleanup. All M preparation, fresh eligible public checks, generated-file cleanup and installed-launcher work are included. Grading, environment acquisition, source/Git copying and evidence packaging are excluded. Setup has a 90-second allowance; native execution has a separate 600-second allowance from the recorded bridge handoff. Native wall time is a local process measurement, not isolated remote inference latency.

Environment profiles and developer instructions were shared. Projects and run-local caches were fresh; OS page caches, provider prefix caches and backend load were uncontrolled. Allocation order was retained across amendments, but interruptions and clock-time effects cannot be removed by reversing order.

After closure, each final source tree was reconstructed without reusing model-generated caches. Public and private checks ran with candidate source read-only, network disabled and bounded artifact write roots. Raw hashes, source-role adjudication, native adjudication and version 3 grading metadata are bound to every result. Strict success requires valid execution and usage, correct treatment, intact hard protections, valid unchanged-source grading and passing original public plus feature checks. A valid public coding failure can conclude strict failure without running private tests. Failures retain their observed cost and time; unknown values are never filled with zero. All 128 pooled records must satisfy integrity and grading prerequisites before the primary inference is eligible.

E. Accounting and analysis

For each task and arm, the two repeats are averaged first. The analysis uses the 28 paired task differences M minus A for two primary endpoints: uniform API-equivalent dollars and full workflow seconds. The dollar convention is [(input minus cached input) $\times$ 5 + cached input $\times$ 0.20 + output $\times$ 15] / 1,000,000. These are native-reported cumulative token counters under a fixed historical tariff, not current prices, subscription charges or actual billing.

Each endpoint receives a paired Student t test with 27 degrees of freedom, one-sided alternative M minus A below zero and alpha 0.025, together with a pointwise 95% two-sided t interval. The joint efficiency claim requires both tests to pass, both upper interval endpoints below zero, complete

primary metrics, all 128 integrity/grade gates and M's strict-success count at least A's among 56 sessions per arm. This intersection-union rule tests a single claim of simultaneous cost and speed superiority; one favorable endpoint cannot replace the other. Equal observed pass counts are a quality-count guard, not a noninferiority or equivalence test.

Intervals and tests are nominal model-based summaries. Their finite-sample calibration assumes independent, normally distributed task differences; repeats do not increase the independent cluster count above 28. Pointwise 10,000-resample task-bootstrap intervals, distribution/influence diagnostics, workflow components and jointly successful timing are secondary. A further secondary summary omits the entire four-session task cluster containing any certified retry warning, symmetrically across arms and repeats. It cannot change the primary decision. Soft known-tariff allowances and admission reserves controlled further dispatch, not actual account charges. The old 0016 unknown usage prevents a complete 144-assignment cost claim even though all 112 primary and all 128 pooled comparison records have complete uniform accounting. The public result package contains the portable export and reproduction instructions.

V. Development history

Earlier studies motivated this comparison but are not pooled with it. Historical capsules preserve source, tasks, outcomes and analysis rules. Reanalyses are not new model sessions, and differing cohorts do not form a common benchmark.

Early native MCP retrieval increased resources: one matched cohort used 2.751 times the input and 1.465 times the Codex-plus-index time; an optimized cohort still increased input by 32.82% and time by 17.88%. An earlier v4 selector delivered no source on any of 24 assisted assignments, so arm differences could not demonstrate a preload benefit. Active pilots also exposed delivery failures and cheap unsuccessful attempts.

The v5 search used three reference sessions and 90 candidate sessions across 30 rounds on three tasks. Verified commands and baseline counts were promoted; 28 other variants were rejected. A separate eight-task comparison scheduled 32 attempts, retaining one provider failure with unknown usage. Seven complete task clusters showed 18.726% lower observed uniform cost with a bootstrap interval crossing zero. Prototype timing excluded public verification, and delivery differed from the installed policy.

The separate v6 study used twelve tasks, three arms and two repeats: 72 completed sessions, all passing finite strict checks. Source-plus-command C had 13.69% lower observed uniform cost than A and 10.04% lower than source-only B. Neither exact contrast passed the two-test Holm rule; timing showed no established speed improvement. Seven tasks preloaded and five abstained. These exploratory results motivated v7 and are not pooled with it.

V7 added eligible command-only delivery after source abstention and demand-driven provider creation. The ordinary source-selection guards remained intact. Its two-arm comparison evaluates this combined policy; it does not reproduce v6's component ablation.

TABLE I
PRIMARY PANEL: 28 TASKS, 112 ATTEMPTS. COSTS ARE MEAN FIXED TOKEN-TARIFF EQUIVALENTS, NOT INVOICES.

Arm	Strict pass	USD/run	Workflow (s)	Native (s)
A	54/56	0.1609423	59.5902	59.3713
M	54/56	0.1531605	60.2863	57.3127

The original v7 schedule stopped after 15 assignments because source-local Vitest caches interacted with capture/protection rules. Eight exposed tasks were excluded from the primary cohort; assignment 0016 then stopped on copied-bridge permissions, retaining unknown usage. Capture, deferred reconstruction and committed grader-byte recovery were repaired before primary collection. Section IV-C records later source-role, telemetry and schema-startup amendments; the supplement retains their detailed chronology and original stopped states.

VI. Results

All 128 pooled comparison records have valid effective telemetry, uniform accounting and deferred grading. The primary 28-task panel contributes 112 sessions, with 54/56 strict passes per arm (Table I). The full descriptive pool has 62/64 passes per arm; its eight previously exposed tasks contribute only remaining second-repeat sessions and do not enter primary inference.

A. Primary cost and latency

The primary uniform equivalents total USD 9.012769 for A and USD 8.5769854 for M. The observed mean difference is -0.00778185 dollars per attempt, or 4.84% lower for M. Mean workflow time is 59.5902 seconds for A and 60.2863 seconds for M: an observed increase of 0.6961 seconds, or 1.17%. These are averages over all assigned primary attempts, including the four strict-assessment failures.

Neither endpoint meets the declared superiority rule. Both nominal intervals include zero, and joint cost-and-speed superiority is not established. The cost estimate favors M, while the workflow point estimate favors A; the data do not establish either a population cost reduction or a population slowdown. The bootstrap sensitivity agrees about uncertainty: its cost interval is $[-0.023639, +0.009080]$ dollars and its workflow interval is $[-2.796425, +4.170258]$ seconds. It does not replace the primary t analysis.

Figure 2 shows the task-level variation and aggregate intervals.

B. Coding outcomes and delivered context

All four strict failures are the two repeats in each arm on task u21 (is-number). Original public checks passed in all four cases. The private feature oracle expected a TypeError

TABLE II

PRIMARY PAIRED TASK-MEAN CONTRASTS, M MINUS A: 28 CLUSTERS AND 27 DEGREES OF FREEDOM. JOINT SUPERIORITY REQUIRES BOTH ONE-SIDED TESTS AT $\alpha = 0.025$ AND BOTH UPPER INTERVAL ENDPOINTS BELOW ZERO.

Endpoint	Mean difference	Change	Nominal 95% CI	One-sided p
Uniform USD	-0.007782	-4.84%	$[-0.025242, +0.009678]$	0.1843
Workflow seconds	$+0.696101$	$+1.17\%$	$[-3.023997, +4.416199]$	0.6480

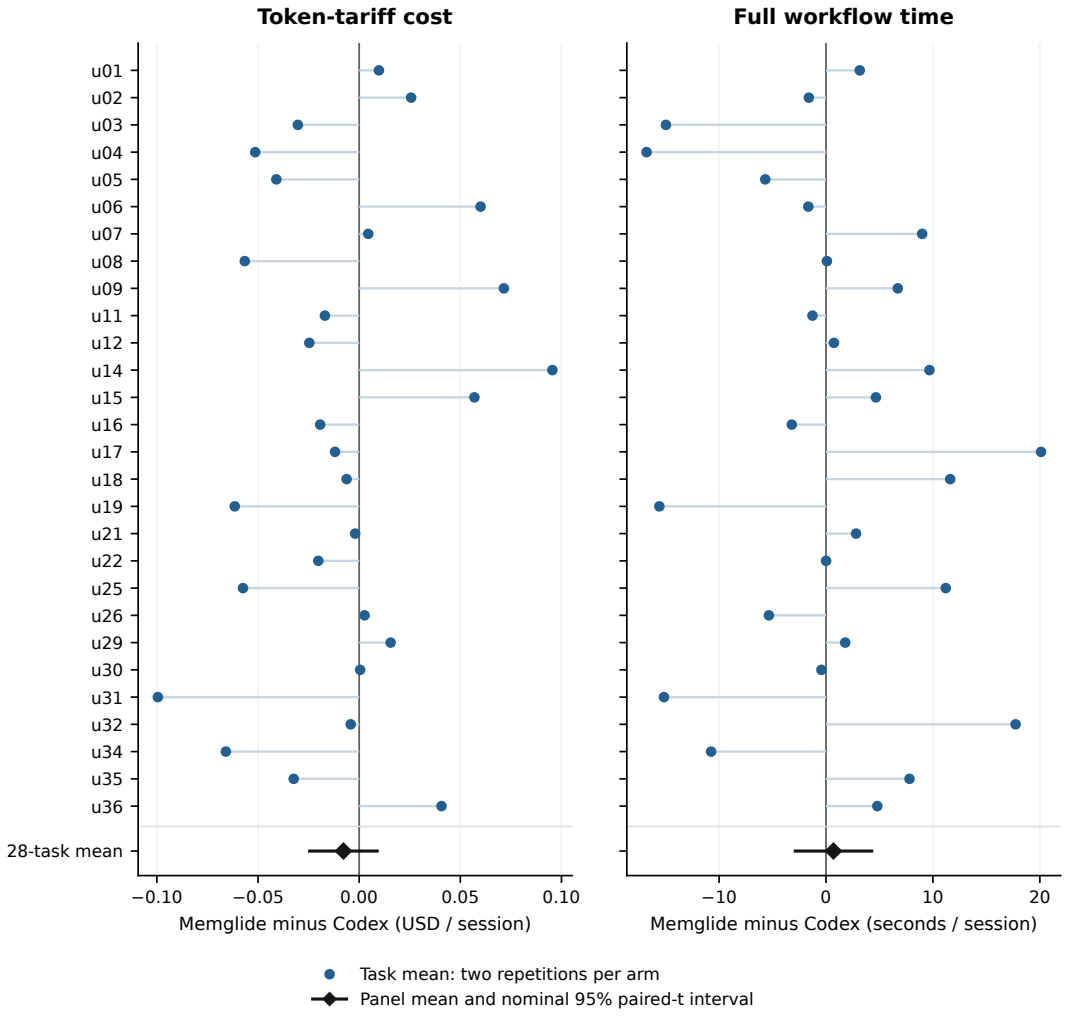


Fig. 2. Task-level M-minus-A contrasts for all 28 primary tasks, averaging two repeats per method. Negative values favor Memglide. The aggregate mean and nominal 95% paired-t interval use tasks as clusters; both resource intervals cross zero.

for four nonboolean safeInteger options; the submitted implementations returned false instead. The public request said "reject nonboolean safeInteger values" without specifying an exception. The frozen rubric therefore contains an ambiguity that limits interpretation of these failures. Each implementation passed 111 upstream tests and 26 of 30 private checks; only the four TypeError assertions failed. All four remain strict failures, with their measured costs and times retained. The u21 exception assertions and these four strict outcomes were not changed or rescored.

M actually preloaded source on 12 primary sessions covering six tasks, supplied command-only context on 18 sessions covering nine tasks, and passed through on 26 sessions covering thirteen tasks. A passed through on all 56. Captured handoffs matched the committed treatment receipts. This mixed policy exposure is part of the primary comparison. It does not identify a causal effect of source alone, command cards alone or active retrieval alone. Both-repeat strict success occurred on the same 27 of 28 tasks in each arm. Equal counts and zero harmful task discordances do not establish quality equivalence; the descriptive one-sided 97.5% harmful-discordance upper bound remains 12.34%.

C. Native time, preparation and robustness

M's observed mean native wall time is 2.0586 seconds shorter than A's. Its total workflow time nevertheless is longer. Workflow minus native time averages 2.9736 seconds for M and 0.2189 seconds for A. M's measured components include 1.4081 seconds of public verification, 0.2432 seconds of indexing and 0.8178 seconds of installed-launcher preparation per primary assignment; planning, cleanup and other local work also belong to the workflow clock. This decomposition describes where time was spent. It does not establish that any individual preparation component caused a change in agent behavior.

Figure 3 shows the corresponding workflow decomposition.

The recorded reconnect warning occurred in A on task u26. Omitting that whole four-session task cluster, as declared for descriptive robustness, leaves 27 tasks: observed uniform cost is 5.13% lower and workflow time 1.57% higher for M. Both nominal paired intervals still include zero. The primary retains u26. Charging the separate failed schema startup's full 2.6373 workflow seconds to canonical u08-M changes the observed workflow increase to 1.25%, with an interval of [−2.976951, +4.463340] seconds. Operator repair time was not measured. Neither sensitivity changes the primary decision. Restricting timing to the 27 tasks passing both repeats in both arms also remains secondary and outcome-conditioned.

Primary input counters average 131,105 tokens for A and 119,320 for M; output averages 2,257 and 2,188 tokens. Cached input averages 110,091 and 99,221 tokens. These are cumulative native counters, not unique source tokens or provider computation. The alternate native-price/write scenarios remain incomplete for one primary record in each arm and are not presented as complete comparative costs. A separately implemented automated audit reproduced the complete pool, primary estimates, intervals, grading counts and accounting from retained records. It is an internal consistency check, not external replication.

Reported values come from the frozen assessment; its full SHA-256 commitment and separate export hashes are retained in the reproducibility package.

VII. Interpretation and limitations

The panel was fixed before its coding sessions, but source-role and retry interpretation changed after 25 and 52 measurements. A pre-model schema failure also required an exception. Preserving raw records and avoiding resource contrasts or private candidate grades when choosing repairs limits favorable selection, but cannot restore prospective confirmation. Local hash freezes are not public preregistration; inference remains nominal under the stated chronology and assumptions.

Tasks are author-designed feature additions in small public repositories, selected for runnable checks. They are not a representative issue benchmark. Several projects share maintainers and conventions, so treating task differences as independent is an approximation. Tasks, references and graders were AI-assisted. Distinct authoring agents and negative controls do not replace independent human task review or external replication. "Unexecuted" means no earlier study coding session on that task; it does not imply source unseen by the model, a blinded retrieval algorithm or an unknown task to all engineering agents. Results cannot establish performance on large codebases, other languages, other model configurations or arbitrary historical issues.

The policy is heterogeneous: source preloading, verified command-only context and pass-through all occur in M. Its contrast with A measures their combined installed workflow. Natural activation is not randomized, and subgroup or cross-study differences cannot isolate the benefit of any component. The older v6 three-arm study used different tasks and delivery behavior; it cannot supply the missing causal ablation for v7. Demand-driven verification was implemented before the primary cohort, but its improvement over eager verification is not measured by a separate v7 arm.

Uniform dollars value native-reported completed-thread tokens at the fixed 5/0.20/15 tariff. They are not invoices, subscription charges, current model prices or proof that all unsuccessful underlying provider attempts are counted. The source-backed retry amendment establishes the meaning of the retained CLI counters, not the provider's billing behavior. Secondary price scenarios have missing records and cannot be promoted into a complete cost claim. The old 0016 usage remains unknown, preventing a complete 144-assignment accounting claim. Engineering, hosting, operator diagnosis and repair effort are not priced. Command-text diagnostics do not measure all file I/O or avoided model inference; reported zero cache writes may reflect upstream defaulting.

Fresh project/run caches do not control provider prefix caches, OS caches, backend revisions or cloud load. Sequential execution, reversed orders and retained pauses cannot remove temporal dependence. Native time is not isolated inference latency. Amortized verification and long-lived-project operation remain unmeasured.

VIII. Conclusion

Memglide combines bounded source discovery with optional verified commands. The amended 28-task comparison retained all 112 primary sessions: each arm passed 54/56 strict assessments. The observed 4.84% lower token-price equivalent and 1.17% longer workflow time both have nominal intervals crossing zero. Shorter native execution was offset by preparation and verification, and one task's ambiguous exception wording limits interpretation of all four failures. General savings, joint resource superiority and quality equivalence remain unestablished.

The contribution is an auditable implementation and an explicit measurement of its resource tradeoff with uncertainty. Subsequent work should use independently reviewed task contracts, a prospectively fixed panel and additional models, and should compare specific preparation choices in randomized arms before attributing benefits to source selection, command cards or deferred verification.

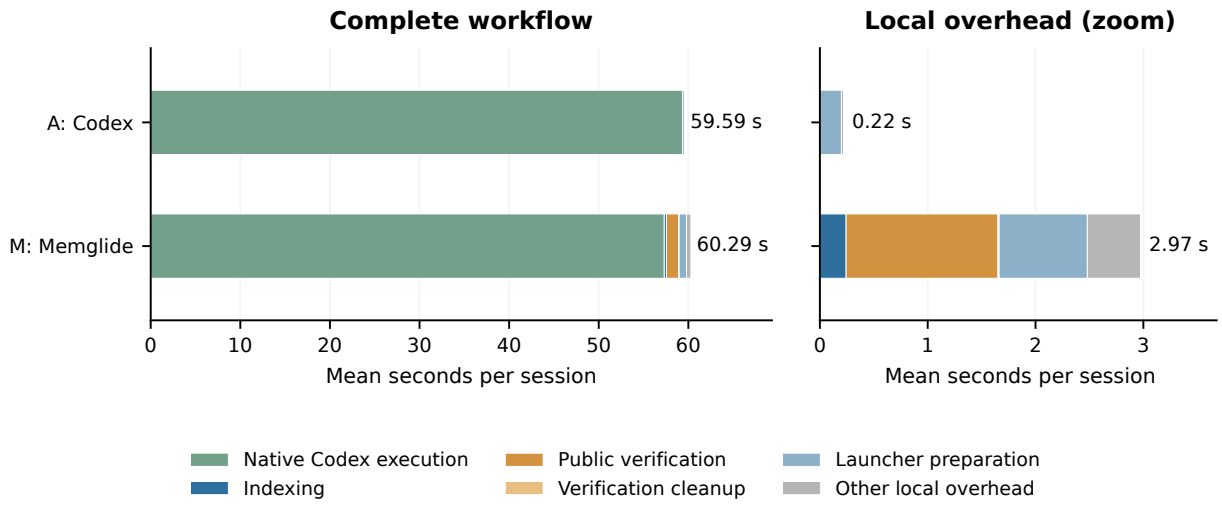


Fig. 3. Mean workflow components over 56 sessions per method, including fresh indexing, eligible public verification, cleanup and launcher preparation. Other overhead is the total less recorded components. Native execution includes process handling and external-model waiting.

IX. Artifact and Reproducibility

The public Memglide repository hosts the project. The updated release supplies the manuscript, supplement, editable sources, bibliography, figures, evaluated implementation and portable export at research/results/router-v7-20261021/. It includes 112 primary measurements, the 128-session pool, task-paired values, public task provenance, protocol information, checksums and an offline statistical reproduction script. Original evidence commitments are distinguished from export hashes. The original final assessment SHA-256 is `fcd5ded7f3c73799cad03a1fda0554294bc974ba34625b6680ee323ef5c94b26`.

The numeric export permits reproduction of the reported arithmetic; it is not an independent replication or a complete execution image. Authentication, installed dependency trees and platform state are excluded. Private grading material and complete native transcripts from the new comparison are not part of this public numeric export. Source commitments identify the preserved records without claiming that a checksum alone makes unpublished material publicly auditable. The complete download retains the earlier v6 evidence capsule in a clearly marked historical directory, separate from the new data. That capsule also remains available at its immutable historical GitHub revision.

The supplement documents accounting, amendments, timing, grading and reproduction boundaries. Recomputing tables or rebuilding PDFs makes no model calls. Repeating collection requires authenticated Codex access, the declared runtime and a new allocation; the sealed allocation is not reopened. This manuscript makes no claim of acceptance, completed peer review, an archive DOI or independent human replication.

X. Author Contributions and AI Assistance

The author, Sree Charan Reddy Kailasam, chose the research question, directed Memglide's development, made the experimental decisions, and interpreted the results.

OpenAI Codex assistants supported implementation, task and control generation, experiment orchestration, analysis code, literature-source checks and manuscript drafting. Separate agents reviewed task contracts and experimental records; these automated reviews do not constitute independent human replication. The experimental coding model is specified in the methods. No AI tool is listed as an author. Final manuscript verification and responsibility for any submission rest with the author.

References

[1] C. E. Jimenez et al., "SWE-bench: Can language models resolve real-world GitHub issues?" in Proc. ICLR, 2024. [Online]. Available: https://openreview.net/forum?id=VTF8yNQM66

[2] F. Zhang et al., "RepoCoder: Repository-level code completion through iterative retrieval and generation," in Proc. EMNLP, 2023, pp. 2471–2484. https://doi.org/10.18653/v1/2023.emnlp-main.151

[3] T. Liu, C. Xu, and J. McAuley, "RepoBench: Benchmarking repository-level code auto-completion systems," in Proc. ICLR, 2024. [Online]. Available: https://arxiv.org/abs/2306.03091

[4] N. F. Liu et al., "Lost in the middle: How language models use long contexts," Trans. Assoc. Comput. Linguistics, vol. 12, pp. 157–173, 2024. https://doi.org/10.1162/tacl_a_00638

[5] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing prompts for accelerated inference of large language models," in Proc. EMNLP, 2023, pp. 13358–13376. https://doi.org/10.18653/v1/2023.emnlp-main.825

[6] J. Yang et al., "SWE-agent: Agent-computer interfaces enable automated software engineering," in Proc. NeurIPS, 2024. [Online]. Available: https://arxiv.org/abs/2405.15793

[7] C. S. Xia, Y. Deng, S. Dunn, and L. Zhang, "Agentless: Demystifying LLM-based software engineering agents," arXiv:2407.01489, 2024. [Online]. Available: https://arxiv.org/abs/2407.01489