

Do Context Files Help Coding Agents? A Two-Agent Ablation Study on Real Repositories

Prakhar Khatri

Independent Researcher

prakharkhatri123@gmail.com

Abstract

Persistent context files (`AGENTS.md`, `CLAUDE.md`) are standard practice for guiding AI coding agents, yet evidence for their effectiveness is contradictory. We present a controlled ablation of context-injection strategy across two frontier agents (Claude Code and Codex), 17 real tasks from 3 repositories (15 shared + 2 Codex-only), and 288 evaluated runs (gold tests on 13 tasks, diff overlap on 4). Context strategy does not measurably move correctness on either agent (bounded to ≤ 10 – 15 pp via *descriptive* equivalence testing, not a powered equivalence claim: our design’s minimum detectable effect exceeds 30pp). A manipulation check confirms the real `AGENTS.md` never converts a near-miss to a pass on either agent, and a failure-mode triage suggests why: agents fail on *implementation skill*—feature design, pattern selection, exact wiring—not missing repository knowledge that a context file could supply. A fourth, as-deployed condition (the real file left in the workspace for Codex to find on its own) also shows no effect. We further show that borderline task difficulty is *agent-specific* (Spearman $\rho=0.75$), offering a candidate explanation for prior contradictions: single-agent studies draw tasks from different agents’ informative bands. We release all code, data, and analysis.

1 Introduction

Repository context files (`AGENTS.md`, `CLAUDE.md`, and similar project-level guides) have become ubiquitous infrastructure for AI coding agents. Agent platforms auto-load their own such file into every session (`CLAUDE.md` for Claude Code, `AGENTS.md` for Codex CLI), and practitioners invest significant effort authoring coding conventions, architectural constraints, and workflow guidance, expecting agents to produce better code as a result.

Yet empirical evidence is contradictory. Lulla et al. (2026) report that `AGENTS.md` files improve agent *efficiency* (lower wall-clock runtime and fewer output tokens), while Gloaguen et al. (2026) find no significant effect on task completion when context files are present versus absent. The two studies differ in agent (Codex-family vs. Claude-family), evaluation method, and experimental control, making reconciliation impossible without a study that varies injection strategy under controlled conditions across both agent families.

This paper presents such a study. We conduct a controlled ablation of context-injection strategy on real merged pull requests, evaluated mainly with hidden gold tests (SWE-bench Tier-C style), across two frontier agents from different providers. Our three pre-registered strategies—NONE (no context), ALWAYS_ON (full `AGENTS.md` in every system prompt), and SELECTIVE (topic-organized wiki files the agent retrieves on demand)—vary primarily *how* context is delivered (see §3.3 for a caveat on the SELECTIVE files).

Our contributions are:

1. A **controlled, two-agent ablation** (288 evaluated runs: 15 Claude tasks + 17 Codex tasks, each $\times 3$ strategies $\times 3$ repeats) with SWE-bench-style evaluation on real repositories, yielding a **bounded null**: the correctness effect is ≤ 10 pp (Claude) / ≤ 15 pp (Codex).
2. A **manipulation check and failure-mode triage** demonstrating that the real `AGENTS.md` cannot convert a near-miss failure to a pass on either agent, and that real tasks instead fail on implementation skill, not missing repository knowledge, the mechanism behind the null.
3. Evidence that **borderline task difficulty** is

agent-specific: the same task may be trivially solvable for one agent yet challenging for another, explaining why single-agent studies reach contradictory conclusions.

4. A **cross-agent methodological pitfall:** a naive turn-count effort classifier silently misclassifies every Codex task as trivial, because Codex emits exactly one `turn.completed` event per session regardless of work performed; portable metrics (tool calls) are required for cross-agent classification.

2 Related Work

Coding agents and benchmarks. SWE-bench (Jimenez et al., 2024) established the issue-to-PR evaluation paradigm for coding agents, and subsequent work on agent-computer interfaces (Yang et al., 2024) shows that harness design (tool ordering, context budgets, retry logic) materially affects performance independent of the underlying model. Our work builds on this paradigm but studies a different variable: persistent repository context rather than harness mechanics.

Context and memory for agents. Persistent memory systems for LLM agents range from operating-system-style memory management (Packer et al., 2023) to session-accumulated skill libraries (Wang et al., 2023), and agent loops that interleave reasoning, acting, and self-reflection (Yao et al., 2023; Shinn et al., 2023). A separate line of work shows that *how* long contexts are arranged matters: models use information unevenly across a long prompt, often underusing material placed mid-context (Liu et al., 2024a); directly relevant to whether an always-on `AGENTS.md` in the system prompt is actually attended to. `AGENTS.md` files represent the simplest form: a static document loaded at session start. Lulla et al. (2026) study the impact of these files on Codex-family agents, measuring wall-clock time and token usage; they report efficiency improvements but do not control for injection strategy. Gloaguen et al. (2026) run a larger naturalistic study across Claude Code, Codex, and Qwen Code and find no significant effect on correctness. Neither study isolates injection strategy as an independent variable, nor do they test across agent families.

Repository-level context for code generation. Retrieval-augmented generation (RAG) (Lewis et al., 2020) has been applied to code tasks by retrieving relevant files, functions, or documentation before generation, and repository-level benchmarks and retrieval methods (REPOCODER (Zhang et al., 2023), CROSSCODEEVAL (Ding et al., 2023), and REPOBENCH (Liu et al., 2024b)) specifically target cross-file and repository-scale context. Recent work on SWE-bench Verified (OpenAI, 2024) refines the benchmark with human-validated instances, raising the evaluation bar. However, these efforts focus on *what* context to retrieve from the codebase at query time, whereas `AGENTS.md` files represent a *static, author-curated* context layer orthogonal to retrieval. Our study isolates whether this layer helps, and how.

Equivalence testing in empirical SE. Traditional null-hypothesis testing cannot confirm the absence of an effect (Schuirmann, 1987). Two one-sided tests (TOST) provide bounded equivalence claims, but remain rare in agent evaluation. We adopt TOST alongside permutation tests and power analysis to characterize our null, following recommendations from empirical software engineering methodology (Wohlin et al., 2012).

3 Method

3.1 Overview

We measure whether the *strategy* by which repository context is injected into a coding agent affects task correctness or efficiency. The independent variable has three levels (NONE, ALWAYS_ON, SELECTIVE); the dependent variables are binary correctness (gold tests pass/fail) and continuous efficiency metrics (tool calls, wall-clock time, output tokens, cache usage). Each task is run under all three pre-registered strategies with 3 repeats on two agents (within-task paired design); §4.2 adds a fourth, as-deployed arm for Codex. In total the design comprises 291 completed agent runs; 3 Claude runs failed to produce a valid gold-test evaluation (crash/timeout) and are excluded from correctness, leaving 288 evaluated cells (efficiency metrics use all completed runs, so their per-cell n varies slightly).¹

¹All 3 exclusions fall on Claude: ALWAYS_ON 46→45, NONE 47→45, SELECTIVE unchanged; Codex is unaffected (51/51/51). Table 1’s Claude denominators already reflect them.

Strategy	File in workspace	In system prompt	Retrievable wiki files
NONE	—	—	—
ALWAYS_ON	—	full AGENTS.md	—
SELECTIVE	—	retrieval hint	wiki/* .md

Figure 1: The three context-injection strategies. In every condition the workspace AGENTS.md is removed, so the injection channel is the *only* context the agent receives. NONE provides no context; ALWAYS_ON injects the full file each turn; SELECTIVE places topic-organized wiki files in the workspace that the agent reads on demand, cued by a system-prompt hint. The SELECTIVE wiki is a section-by-section split of the AGENTS.md (§3.3).

3.2 Task Selection and Evaluation

Repositories. We surveyed ~ 40 repositories drawn from three sources: our own manual scan, and the repository sets of Lulla et al. (2026) and Gloaguen et al. (2026). We excluded SWE-bench repositories (none maintain AGENTS.md files) and filtered candidates on four criteria: (1) exactly one root AGENTS.md with no competing instruction stack; (2) file quality rated Good or Excellent on a structured rubric (coverage of build/test/style conventions, architectural guidance, specificity); (3) feasible pilot setup (no oversized monorepos); (4) Python-only, to avoid confounding context effects with build-system or language differences. Six repositories survived; we narrowed to three for budget feasibility: **pdm** (package manager, 477-word context file), **firebase-admin-python** (cloud SDK, 1236-word file rated “Excellent”), and **opshin** (domain-specific compiler, 255-word file). These span distinct domains (tooling, cloud infrastructure, compilers) and AGENTS.md quality levels (255–1236 words). The other three surviving repositories were dropped on budget grounds alone.

Tasks. We mine tasks from merged pull requests via the GitHub API: the PR description becomes the agent prompt, the base commit the starting state, and the PR’s own test files the gold evaluation. We prefer PRs that (1) add or modify test files (required for gold-test scoring; four firebase tasks without tests (906, 926, 932, 939) fall back to fuzzy diff overlap, which inflates apparent variance), (2) are non-trivial (multiple files or >50 LOC), and (3) need no external credentials or services (for the egress-locked pod). This

yields an initial batch of 12 tasks (4 per repository, spanning medium and complex difficulty). To avoid a floor/ceiling design (where every task is too easy or too hard for any manipulation to register), we ran a Codex screening sweep over 28 candidate tasks (repeats=3, strategy NONE, 84 screening runs, a substantial hidden cost beyond the main study) to locate the *borderline* band ($0 < \text{pass} < 1$), adding 5 borderline tasks ($12+5=17$). Screening also surfaced a portability bug in our effort classifier (§4.5). The final set is 17 tasks for Codex and 15 for Claude; two high-effort opshin tasks from the borderline sweep are Codex-only, dropped from the Claude arm under an Anthropic budget cap, leaving 15 tasks shared across both agents. Because the borderline band was calibrated on Codex, the screened tasks need not be borderline for Claude, an instance of the agent-specific difficulty we analyze in §4.4.

Evaluation (Tier-C). After the agent completes, we extract test files from the gold PR diff, apply them onto the agent’s workspace, and run them against the agent’s code. The task passes if and only if all gold tests pass; the four test-less tasks instead pass on $\geq 50\%$ recall of the gold diff’s changed lines. This follows the SWE-bench evaluation protocol: the agent never sees the test files; they serve as an oracle.

Safety and isolation. All runs execute on an egress-locked pod with GitHub DNS blackholed. Git remotes are scrubbed, credentials are stripped, and git push/commit commands are denied via PATH shims. Future commit history is pruned so the agent cannot read gold solutions from git log.

3.3 Independent Variable: Injection Strategy

NONE: The AGENTS.md file is removed from the workspace. No repository context is provided in the system prompt or accessible to the agent. The agent works from the code-base alone.

ALWAYS_ON: The full AGENTS.md content is injected into the system prompt every turn, preceded by a framing sentence (“The following is the repository’s AGENTS.md guide...”). The file is removed from the workspace to prevent double-reading.

SELECTIVE: Topic-organized wiki files (e.g., wiki/architecture.md,

wiki/error-handling.md) are placed in the workspace, and a system-prompt hint tells the agent to consult wiki/*.md; the agent retrieves relevant files on demand via its Read tool. **Caveat on the SELECTIVE files.** The wiki is a verbatim split of the AGENTS.md at its section headings (a single overview.md for opshin), so its content matches the file in all three repositories. Our splitter, however, never cleared earlier output, so on pdm and firebase later runs found the same sections repeated under suffixed filenames (up to roughly 10× and 18× copies in our local wiki directory). The content is unchanged but the directory is larger and redundant; in the stream logs we retain, 1 of 45 agent references to wiki files hit a duplicate.

In all three conditions above, the AGENTS.md file itself is removed from the workspace, so each strategy delivers only what it explicitly provides. The within-strategy contrasts are clean; the duplication caveat above qualifies the SELECTIVE arm and is revisited in the Limitations.

A fourth, as-deployed condition. NATURAL leaves the repository’s AGENTS.md in the workspace and injects nothing, making the agent’s own ingestion the treatment, the condition practitioners actually run (§4.2). It applies to Codex only, for a reason that is itself a finding: Codex CLI reads AGENTS.md natively, whereas Claude Code reads CLAUDE.md and never auto-loads AGENTS.md, so for Claude the file would be discoverable but never auto-loaded.

3.4 Agents

We run two frontier coding agents from different providers:

- **Claude Code** (claude-sonnet-4-6, Anthropic). Multi-turn, tool-use agent invoked via `claude --print --bare --output-format stream-json`. Context injected via `--append-system-prompt`. Cost-capped at \$6/run.
- **Codex CLI** (gpt-5.5, OpenAI, ChatGPT-plan auth). Single-session agent invoked via `codex exec --json`. Context injected by prepending to the user prompt (Codex lacks a system-prompt flag). Turn-capped at 120 via watchdog.

Both agents use the same workspace, task prompt, and evaluation pipeline. The injection-channel asymmetry (system prompt vs. user-turn prepend) is a noted confound (see Limitations): Claude’s per-turn system-prompt re-presentation and Codex’s one-time prepend are materially different manipulations of ALWAYS_ON, not just different plumbing.

3.5 Efficiency Metrics

We distinguish *portable* metrics (comparable across agents) from *agent-specific* metrics:

- **Portable:** tool calls, wall-clock duration, output tokens.
- **Agent-specific:** cache-read tokens, cache-creation tokens (Claude only; Codex folds cached input into `input_tokens` with no separate accounting).

Turns are excluded as a cross-agent metric: Codex emits exactly 1 `turn.completed` event per session regardless of work performed (a classifier artifact we discovered during screening; §4.5).

3.6 Statistical Analysis

The unit of analysis is the *task* (not the repeat). For each (task, strategy) pair, the 3 repeats are averaged to yield one value per task per strategy. We then apply:

- **Omnibus permutation test:** strategy labels permuted within-task (10,000–20,000 iterations); test statistic = variance of marginal strategy pass-rates.
- **Paired Wilcoxon signed-rank tests** with Holm-Bonferroni correction for efficiency metrics (family of 12 tests across 3 pairs × 4 metrics).
- **TOST equivalence:** task-clustered bootstrap (10k) on paired strategy differences; equivalence declared if 95% CI falls within $\pm\delta$.
- **Monte Carlo power:** simulate binary outcomes under varying true Δ and n_{tasks} to determine minimum detectable effect (MDE) and sample size for 80% power.

Table 1: Pass-rate by strategy and agent. Omnibus permutation p -values test whether strategy explains correctness beyond task difficulty, and cover the three pre-registered arms. NATURAL is a later-added robustness condition (§4.2) and is excluded from the omnibus.

Strategy	Claude (15 tasks)	Codex (17 tasks)
NONE	53.3% (24/45)	58.8% (30/51)
ALWAYS_ON	55.6% (25/45)	56.9% (29/51)
SELECTIVE	55.6% (25/45)	52.9% (27/51)
NATURAL	n/a [†]	62.7% (32/51)
Omnibus p	1.000	0.66

[†]Not run: Claude Code does not auto-load AGENTS.md (§3.3).

4 Results

4.1 Correctness: No Strategy Effect (Both Agents)

Table 1 presents pass-rates by strategy and agent. Neither agent shows a statistically significant strategy effect.

For Claude, all pairwise strategy differences are ≤ 2.3 pp. For Codex, the largest difference is 5.9pp (NONE vs. SELECTIVE), driven by per-task noise at $n=3$ repeats. The omnibus permutation test shows no detectable strategy effect ($p=1.00$ Claude; $p=0.66$ Codex). This test is intrinsically low-power here: the floor/ceiling structure (§4.4) leaves little variance to permute, so $p=1.00$ is close to mechanical.

The null holds with dynamic range. A potential objection is that the null reflects floor/ceiling effects (tasks too easy or too hard for context to matter). We address this with borderline-task expansion: on 4 Codex-borderline tasks (17–67% baseline pass rate, 12 runs/strategy), NONE passes 7/12 (58%) vs. ALWAYS_ON 5/12 (42%) and SELECTIVE 5/12 (42%): context injection does *not* help even where the design has power to detect an effect.

Equivalence bounds. Descriptive equivalence testing (TOST on the task-clustered bootstrap) bounds every pairwise strategy difference to <10 pp for Claude and <15 pp for Codex. This is compatible with the >30 pp MDE (§4.6) because the observed point estimates are ≈ 0 , not a powered equivalence claim given $n=15$ –17 clusters.

4.2 The As-Deployed Condition Also Shows No Effect

Under NATURAL (the repository’s own AGENTS.md left in the workspace, nothing injected), Codex passes 32/51 (62.7%, Wilson CI 49.0–74.7%) against 58.8% for NONE; every paired contrast is non-significant and inside the ≤ 15 pp bound of §4.1 (Appendix C). More tellingly, NATURAL matches NONE on 15 of 17 tasks and the floor/ceiling structure reproduces exactly, so ordinary discovery of a real context file behaves like no context file at all: exactly what Limitation 4 previously had to assume. This arm supports the correctness comparison only; it ran later on a different agent build, so we draw no efficiency conclusions from it.

4.3 Efficiency: Two Narrow Process Signals, Otherwise Null

Claude. SELECTIVE uses significantly less cache-creation tokens than NONE (13/15 tasks lower; $p=0.002$, $p_{\text{Holm}}=0.024$); the two exceptions are both firebase tasks (926, 932). Directional but non-significant trends appear for cache-read (11/15 lower, $p=0.064$) and ALWAYS_ON vs. SELECTIVE duration (11/15 lower, $p=0.064$). No significant effect on turns or tool calls after correction. All efficiency statistics use the full 15-task Claude set; Codex is excluded from cache metrics, which it records as zero rather than null. We read this cache result mechanically rather than as a strategy benefit: SELECTIVE keeps only a short retrieval hint in the system prompt and reads a few files on demand, whereas ALWAYS_ON re-presents the whole AGENTS.md each turn, so the footprint difference follows from the delivery mechanics, not from context making the agent more capable. The retrieval channel is demonstrably live: Claude reads a wiki file in 30/45 SELECTIVE runs (66.7%, 58 reads, 11/15 tasks), against 0/93 in the NONE/ALWAYS_ON controls where no such file exists. Codex emits no structured file-read events at all (§4.5), so we cannot report its wiki-read rate and do not claim one.

Codex. All efficiency metrics are flat across strategies: tool calls 32/32/32, output tokens $\pm 3.8\%$, duration $\pm 3.8\%$ (all $|d_z| < 0.2$, negligible effect size).

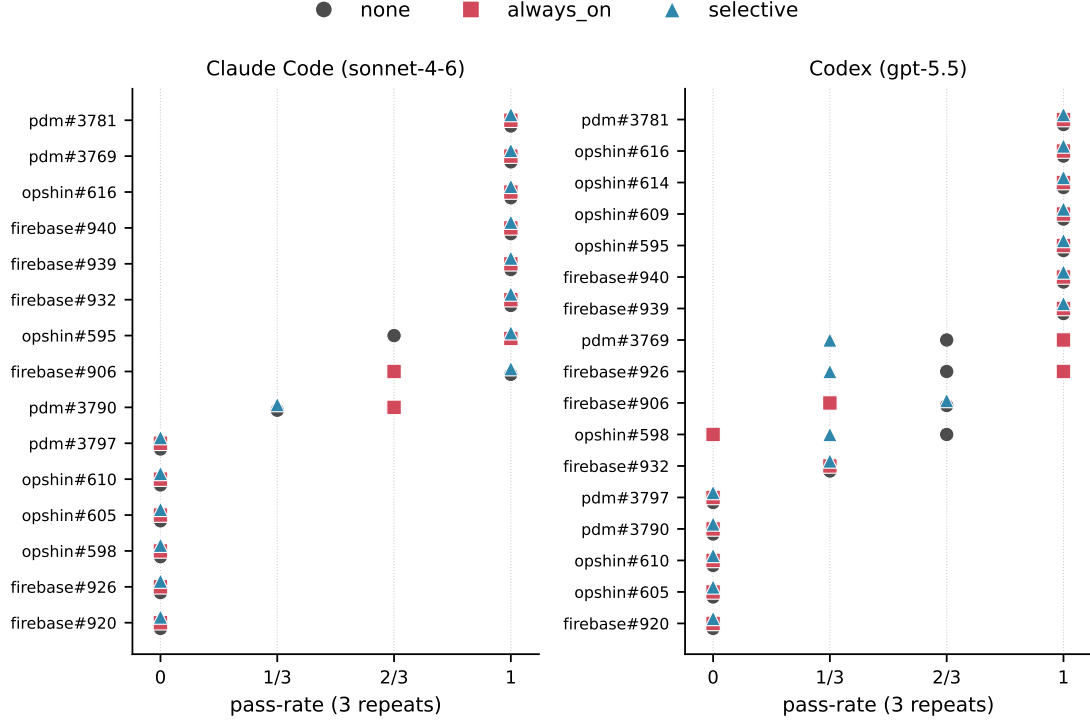


Figure 2: Per-task pass-rate (3 repeats) by injection strategy, one panel per agent, tasks ordered by mean pass-rate. For the large majority of tasks the NONE/ALWAYS_ON/SELECTIVE markers coincide (strategy does not move correctness), and tasks sit at the floor (0) or ceiling (1). The markers separate only on the handful of borderline tasks, which differ by agent (e.g. pdm#3790 for Claude; pdm#3769, firebase#926 for Codex), illustrating the agent-specific difficulty of §4.4.

A within-agent process effect (Claude $\times$ opshin). A second, narrow efficiency signal survives. On opshin, the one repository whose `AGENTS.md` carries explicit *runtime/compile-time* warnings (“the full test suite takes >20 minutes”), Claude’s within-task wall-clock time is $\sim 24\%$ lower under context (NONE 2689s vs. ALWAYS_ON 2066s vs. SELECTIVE 2032s; faster-with-context on 4/5 tasks, sign-flip $p=0.125$, underpowered at $n=5$). What lifts this above noise is a *dose-dependent* mechanism: counting the agent’s `pytest` invocations, the number of *blind full-suite* runs per cell falls monotonically NONE 3.67 $\rightarrow$ ALWAYS_ON 2.44 $\rightarrow$ SELECTIVE 1.67 (Table 2), directional under a within-task paired test (fewer on 3/4 tasks, $p=0.25$, $n=4$; per-task counts in Appendix B). Stripped of the `AGENTS.md` warning, the agent repeatedly runs the slow full suite; given the warning, it runs targeted tests instead. The dose ordering (SELECTIVE $<$ ALWAYS_ON $<$ NONE) matches both the full-suite count and the duration. The scope is deliberately narrow: the effect appears only for Claude (Codex duration

Table 2: Claude on opshin: context reduces blind full-suite test runs and wall-clock time. Full-suite count = `pytest` invocations targeting the whole suite per cell.

Strategy	Full-suite runs/cell	Duration (s)
NONE	3.67	2689
ALWAYS_ON	2.44	2066
SELECTIVE	1.67	2032

is flat) and only on opshin (firebase shows the opposite $+82s$ direction, a fast repository with little to warn about). Like the cache signal, it is a *process* effect (how the agent works), not an *outcome* effect (correctness, still null). We report it as *exploratory*: it emerged from a post-hoc mechanism analysis rather than the pre-registered efficiency-metric family, and we therefore do not fold it into the Holm-corrected confirmatory tests below.

Cross-agent comparison (portable metrics, qualitative). Codex is consistently leaner (fewer tool calls, less wall-clock time, fewer output tokens) at a slightly lower pass-rate. We report this only qualitatively: the agents differ in injection

Table 3: Agent-specific difficulty. Pass counts none/always/selective; the borderline outcome is marked **bold***. pdm#3790 is borderline for Claude only; pdm#3769 for Codex only.

Task	Claude	Codex
pdm#3790	1/3 · 2/3* · 1/3	0/3 · 0/3 · 0/3
pdm#3769	3/3 · 3/3 · 3/3	2/3 · 3/3* · 1/3

channel (§3.3), tokenizer, and token accounting (Codex folds cached input into `input_tokens`), so any between-agent gap is confounded. Our actual contrast (the *within*-agent strategy comparison) is unaffected.

4.4 Agent-Specific Borderline Tasks

A striking cross-agent finding: the *same task* occupies different difficulty bands depending on the agent.

This is not merely anecdotal. Across the 15 tasks both agents attempted, per-task pass rates are positively but imperfectly correlated (Spearman $\rho=0.75$, 95% task-clustered bootstrap CI [0.42, 0.93], $n=15$, 10k resamples; $p=0.001$; Pearson $r=0.77$): difficulty largely transfers, yet the *informative band* does not. Of the 15 shared tasks, 6 are borderline ($0 < \text{pass} < 1$) for exactly one agent, and 6 differ in floor/ceiling status across agents, so for roughly 40% of tasks, the agent that would reveal a context effect is not the same agent.

This offers a candidate explanation for prior contradictions, a hypothesis we cannot test directly without their exact task sets: if Lulla et al. (2026) (Codex-family) and Gloaguen et al. (2026) (Claude Code, Codex, Qwen Code) drew tasks that landed in different agents’ borderline bands, they would observe different context effects, not because context inherently helps or doesn’t, but because the informative range is agent-specific. A practical consequence is that ablation studies must screen tasks *per agent*; a borderline set calibrated on one agent gives another agent mostly floor/ceiling tasks, where no manipulation can show an effect.

4.5 Classifier Portability Lesson

During borderline screening, our effort classifier (split on turns ≥ 30) silently misclassified all Codex tasks as “trivial” because Codex emits exactly 1 `turn.completed` event per session. Eight genuinely high-effort tasks (18–51 tool calls, 410–1220s) were incorrectly dropped.

Reclassifying on *tool calls* (the pre-registered portable metric) recovered them. This demonstrates that turn-based metrics are non-portable across agent architectures.

4.6 Power Analysis and Detection Limits

Our Monte Carlo simulation (§3.6) reveals:

- MDE at $n=17$, reps=3: even a large $\Delta=30\text{pp}$ effect is caught only 57% of the time; smaller effects fare worse.
- Detecting $\Delta=10\text{pp}$ at 80% power requires $\sim 120\text{--}200$ tasks.
- Adding repeats barely helps ($n=17$, reps $3 \rightarrow 10$: power increases 13% $\rightarrow$ 58% for $\Delta=15\text{pp}$).

Task-level variance dominates: scaling requires more *tasks*, not more repeats. This is a methodological finding relevant to future ablation study designs.

5 Discussion

5.1 Manipulation Check

Before interpreting the correctness null, we first ask a prior question: did our manipulation do anything at all, or did we simply inject text that had no measurable effect (an *inert manipulation*)? We address this directly with a check we designed and committed to before running it. First, we rated our three context files against a structured quality rubric we applied across a 40-repository `AGENTS.md` corpus (scoring coverage of build/test/style conventions, architectural guidance, and specificity); on this rubric *firebase* rates “Excellent,” *pdm* and *opshin* “Good”; they are not low-quality files. This is our own rubric assessment, not a third-party audit. Second, we reran the two near-miss tasks whose failures looked most convention-related on inspection, a judgment call by the authors and not a formal selection rule (*opshin* 554, *firebase* 907), under all three strategies, 3 repeats, on *both* agents (36 cells), to test whether the real, unmodified `AGENTS.md` can flip a near-miss to a pass (Table 4). *Opshin* 554’s base commit predates its repository’s `AGENTS.md`, so its `ALWAYS_ON` cells injected nothing; the evidence about the file itself rests mainly on 907.

Table 4: Manipulation check: pass-rate (none/always/selective) on the two near-miss tasks selected for this check. The real `AGENTS.md` never converts a failure to a pass on either agent; in the single task with cross-agent range (Claude, 907), more context does not improve correctness.

Task	Codex	Claude
907 (firebase)	0/3 · 0/3 · 0/3	2/3 · 1/3 · 0/3
554 (opshin)	0/3 · 0/3 · 0/3	0/3 · 0/3 · 0/3

The result confirms the manipulation is live and sharpens the picture. (1) *No helpful flip anywhere*: the real `AGENTS.md` never rescues a near-miss (Codex fails 18/18 regardless of strategy; task 907’s NONE near-miss of 109 pass / 1 fail never crosses). (2) *In the one task with cross-agent range, more context did not help*: task 907 is Codex-hard but Claude-borderline, and Claude passes it 2/3 under NONE, 1/3 under ALWAYS_ON, 0/3 under SELECTIVE. We do not claim a general downward effect (this is a single task at $n=3$ repeats), but the trend is clearly non-positive, and no condition on either agent produced a flip toward passing. (3) Task 554 is Claude-floored, another instance of agent-specific difficulty (§4.4). This check is a stronger answer to “did your manipulation do anything?” than a bare null: across both agents and all 36 cells, no condition converts a near-miss to a pass, including the 18 cells of task 907, where the real `AGENTS.md` was delivered. The manipulation is not inert (it can perturb behavior, and our files rate Good/Excellent on our quality rubric); it simply does not supply the implementation skill that gates these tasks.

5.2 Failure-Mode Triage: Skill, Not Knowledge

The manipulation check (§5.1) establishes that the null is not an artifact of inert context; a qualitative triage of the near-miss failures suggests *why* it holds. We inspected the near-miss failures (1–4 failing gold tests), the tasks most likely to flip with one additional piece of information:

- **Task 510 (opshin)**: requires a subtle union-expansion optimization. The agent built the entire optimization pass but introduced a correctness bug. Gap: engineering precision, not a missing fact.
- **Task 907 (firebase)**: requires proactive auth-token refresh. The agent implemented reac-

tive retry instead. Gap: architectural pattern choice, not secret knowledge.

- **Task 554 (opshin)**: requires rejecting V2 validator arguments. The agent knew the V3 rule (from the code) but miswired the check. Gap: exact behavioral specification.
- **Task 593 (opshin)**: requires type-narrowing through `isinstance + assert`. Gap: deep type-system reasoning.

None of these four cases isolate a knowable-fact gap that an `AGENTS.md` could fill; they instead read as *implementation skill*—feature design, pattern selection, exact wiring—rather than missing repository-private knowledge. We report this triage as a qualitative reading of four cases, not a quantified claim: it is illustrative of a mechanism the 36-cell manipulation check supports directly, not independent statistical evidence on its own.

5.3 Reconciling Prior Work

Our results suggest a reconciliation of the Lulla et al. (2026) / Gloaguen et al. (2026) contradiction (offered as a hypothesis, since we do not have access to their exact task sets):

1. Context files *do not* improve correctness on either agent family (confirming Gloaguen et al., 2026).
2. The efficiency gain Lulla et al. (2026) report likely reflects the *mechanical* cost of context injection (larger prompts → more tokens), not a task-performance improvement. Notably, Gloaguen et al. (2026) report the opposite sign on their own cost metric (context files *increase* inference cost by over 20%), consistent with injection mechanics, not agent capability, driving these numbers in either direction depending on setup. Our SELECTIVE strategy achieves a real cache-efficiency reduction ($p_{\text{Holm}}=0.024$), but this is likewise an artifact of injection mechanics, not agent capability.
3. Prior disagreements may reflect agent-specific borderline effects (§4.4): a task in one agent’s informative band appears in another’s floor/ceiling.

5.4 Practitioner Implications

The `AGENTS.md` files in our sample are real-world context files representative of practitioner effort: style guides, architectural notes, error-handling conventions. Scoped to our setting (3 Python repositories, naturalistic style-guide-type context, and the injection *channel* rather than content), we detect no correctness benefit: **in this sample, generic context files do not measurably improve coding-agent correctness** (bounded to <10–15pp; see Limitations). We cannot rule out benefits for other languages, larger repositories, or purpose-built task-specific context; the safe reading is that effort spent on *generic* context documents may pay off less than effort on task decomposition, tooling, or example-driven prompting. One nuance is genuinely actionable, however: even when context does not change pass/fail, it can change *how* the agent works: on opshin, our SELECTIVE strategy measurably reduces wasted full-suite test runs (Table 2), so context may still earn its place on cost and latency grounds. This is a single-repository, exploratory finding at $n=4$ tasks (§4.1); a hypothesis for practitioners to test, not a validated recommendation.

6 Conclusion

We presented a controlled ablation of context-injection strategy for coding agents, spanning 291 agent runs (288 evaluated) across two frontier agents and 17 real-world tasks, plus a 36-cell manipulation check and a 51-run as-deployed arm. Context-injection strategy does not measurably move correctness (bounded to <10–15pp via descriptive TOST equivalence; not a powered claim, §4.6), and leaving the real `AGENTS.md` in the workspace for Codex to find on its own does not move it either (§4.2). The two surviving efficiency effects are narrow and *process-level*: a cache-footprint reduction for Claude’s SELECTIVE strategy (mechanical; §3.3), and a dose-dependent, single-repository reduction in blind full-suite test runs; neither changes correctness.

Our manipulation check and failure-mode analysis point to the mechanism: real coding tasks fail on implementation skill, not missing repository knowledge a context file could supply; across both agents the real file never converts a near-miss to a pass. The “borderline is agent-specific” finding explains prior contradictions and implies future ablations must screen tasks per-agent. We

release the harness, all datasets, and the power-analysis code to support scaling toward the ~120 tasks needed for powered equivalence at 10pp.

Limitations

1. **Sample size.** 15/17 tasks with 3 repeats. MDE >30pp; a 10pp effect is undetectable. Our TOST bounds the effect to <10–15pp but cannot achieve narrower equivalence without ~120 tasks.
2. **Repository diversity.** Three Python repositories. Results may not generalize to other languages, larger codebases, or repositories with exceptionally detailed context files.
3. **Injection-channel asymmetry.** Claude receives context via system prompt; Codex via user-turn prepend (no system-prompt flag). This confounds the between-agent comparison; within-agent strategy contrasts remain clean.
4. **Ecological validity.** ALWAYS_ON is stronger than ordinary use. The NATURAL arm (§4.2) measures the as-deployed case for Codex only, and ran three months later on a newer agent build, so we use it for correctness alone. Claude Code does not auto-load `AGENTS.md` (documented, and confirmed by direct test on `claude-code 2.1.267`); a `CLAUDE.md` that imports the file, as pdm ships, is an untested condition.
5. **Selective is our construction.** Other retrieval designs (e.g., semantic search) might behave differently. Our splitter also left duplicated section files in the pdm and firebase wikis (same content; §3.3). Claude read a wiki file in 66.7% of SELECTIVE runs; Codex logs no structured file reads, so its usage is unmeasured, not zero.
6. **Inert-manipulation concern.** Our files are naturalistic, not task-specific. The manipulation check (§5.1) shows the real `AGENTS.md` does not rescue near-misses, but it rests mainly on one task (907); purpose-built context may still help.
7. **Model-version snapshot.** Results are specific to `claude-sonnet-4-6` and `gpt-5.5`; the null may shift as these models are updated.

8. **Mixed provenance.** Claude repeat-0 ran locally and repeats 1–2 on the pod; dropping repeat-0 keeps Claude’s pass rates at 53–55% under every strategy and every paired contrast within $\pm 3.3\text{pp}$.

Ethics Statement

This study involves no human subjects or personal data. The experimental subjects are AI coding agents operating on public open-source repositories. All agent runs execute in an egress-locked sandbox that prevents any modification of external state (no pushes, commits, or pull requests reach GitHub); see Appendix A. AI coding agents (Claude Code, Codex) are the *object of study*, not authoring tools. Separately, the authors used an AI assistant for drafting support and code scaffolding; all experimental results, statistics, and claims were verified by the authors against the released data.

References

- Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. 2023. Cross-CodeEval: A diverse and multilingual benchmark for cross-file code completion. In *Advances in Neural Information Processing Systems (NeurIPS)*. ArXiv:2310.11248.
- Thibaud Gloaguen, Niels Mündler, Mark Müller, Veselin Raychev, and Martin Vechev. 2026. [Evaluating AGENTS.md: Are repository-level context files helpful for coding agents?](#) *Preprint*, arXiv:2602.11988. ArXiv:2602.11988.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*. ArXiv:2310.06770.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*. ArXiv:2005.11401.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics (TACL)*. ArXiv:2307.03172.
- Tianyang Liu, Canwen Xu, and Julian McAuley. 2024b. RepoBench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations (ICLR)*. ArXiv:2306.03091.
- Jai Lal Lulla, Seyedmoein Mohsenimofidi, Matthias Galster, Jie M. Zhang, Sebastian Baltes, and Christoph Treude. 2026. [On the impact of AGENTS.md files on the efficiency of AI coding agents.](#) *Preprint*, arXiv:2601.20404. ArXiv:2601.20404.
- OpenAI. 2024. Introducing SWE-bench verified. OpenAI blog post and dataset release. <https://openai.com/index/introducing-swe-bench-verified/>.
- Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*.
- Donald J. Schuirmann. 1987. A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. *Journal of Pharmacokinetics and Biopharmaceutics*, 15(6):657–680.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. ArXiv:2303.11366.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*; arXiv:2305.16291.
- Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer.
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*. ArXiv:2405.15793.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*. ArXiv:2210.03629.
- Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-level code completion through iterative retrieval and

generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ArXiv:2303.12570.

A Experimental Harness Details

Safety layers. Our harness implements defense-in-depth to prevent the agent from modifying external state: (1) DNS blackhole for GitHub (egress lock); (2) `git remote remove` on all workspace remotes; (3) `git push/commit/remote` denied via PATH shims and Claude’s `--disallowedTools`; (4) `GH_TOKEN/GITHUB_TOKEN` stripped from environment; (5) future git history pruned (agent cannot `git show` the gold commit).

Evaluation pipeline. After agent completion: (1) capture diff against `base_sha`; (2) restore gold test files to base state; (3) apply gold test patch; (4) run `pytest` with per-repo configuration; (5) pass iff all gold tests pass with zero failures/errors (test-less tasks: $\geq 50\%$ changed-line recall against the gold diff).

B Per-Task Results

Figure 2 plots per-task pass-rates for both agents. The complete per-task pass/fail counts are available in the supplementary material (`results-summary.csv`) alongside the `experiment_full.db` SQLite database.

Opshin full-suite test runs (§4.1). Per-task blind full-suite `pytest` counts (NONE/mean-context) behind Table 2: task 595 2.50/0.75, task 605 1.20/1.25, task 610 2.50/0.67, task 616 9.33/6.25. Three of four tasks run fewer full-suite invocations under context; task 605 runs counter to the trend.

C As-Deployed (NATURAL) Arm Detail

Codex, 17 tasks $\times$ 3 repeats = 51 runs, September 2026. Paired per-task contrasts against the three pre-registered arms (task-clustered bootstrap, 10k resamples; exact two-sided sign test over non-tied tasks):

Contrast	Δ	95% CI	sign p
NATURAL — NONE	+3.9pp	[0.0, +9.8]	0.50
NATURAL — ALWAYS_ON	+5.9pp	[−5.9, +19.6]	1.00
NATURAL — SELECTIVE	+9.8pp	[+3.9, +17.6]	0.062

Against NONE, NATURAL is better on 2 tasks (firebase 906, 932, both scored by diff overlap;

906’s only gold change is the `AGENTS.md` itself, which scoring excludes), worse on none, and tied on 15. The arm ran on `codex-cli` 0.154, three months after the pre-registered arms, which ran on 0.137. Wall-clock (234s vs. ~ 900 s) and tool calls (44.9 vs. 32.0) differ accordingly, which is why we restrict this arm to correctness.

D Reproduction and Data Availability

We release the full experimental harness, the context-injection strategy implementations (NONE/ALWAYS_ON/SELECTIVE), the defense-in-depth safety layer, the task specifications, the statistical analysis code, the 291-run ablation dataset (288 evaluated cells), the 36-cell manipulation check (`experiment_full.db`, `probe_codex.db`, `probe_claude.db`), the 51-run NATURAL arm (`natural_arm.db`), and an aggregated per-cell results table (`results-summary.csv`). Raw agent transcripts are available from the authors on request. The repository is public: <https://github.com/codeprakhhar25/context-files-coding-agents>.

```
# Requires: experiment_full.db
python3 power_analysis.py \
    experiment_full.db codex
python3 power_analysis.py \
    experiment_full.db claude_code
python3 efficiency_stats_correct.py
```