KEEPING UP WITH THE CODE: FILE CONTEXT SYNCHRONIZATION FOR CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents change a repository while working, but their prompts usually keep every earlier file read. The prompt therefore accumulates duplicate code and code made stale by later edits. Existing compression methods ask which history is useful; they do not check whether that history is still true. We introduce LiveState, a training-free context layer that keeps file observations synchronized with the repository. It replaces old read outputs with stable pointers to a bounded panel containing only the regions the agent inspected. After an edit, LiveState finds each region in the current file and refreshes its text; if the match is uncertain, it hides the region rather than show stale code. On SWE-bench Verified, LiveState reduces mean turns for all eight models from three providers, while accuracy and token use vary by model. Claude 5 Fable matches baseline pass@1 ($+0.1$ points) with 17.0% fewer turns, and Gemini 3 Flash gains 6.7 points (paired 95% CI $[+4.3, +9.2]$). The other Claude models trade 1.4–3.3 points for 24–35% fewer turns and 23–48% fewer tokens. On the longer-horizon DeepSWE benchmark, Fable keeps pass@1 within 0.4 points while using 37.7% fewer turns and 33.3% fewer tokens, and within an 8M-token budget it resolves 43.4% of tasks versus 24.5% for the baseline. Keeping file context current is therefore not just prompt shortening: it also changes how agents act.

1 INTRODUCTION

Coding agents solve repository-level tasks through repeated cycles of reading files, editing code, and running tests (Yang et al., 2024; Wang et al., 2025). Although the repository changes during this process, the model usually conditions on an append-only transcript. A file read early in a trajectory is therefore retransmitted on every later turn, even after a subsequent read duplicates it or an edit makes it stale. The transcript records what the agent once observed, not what is true now.

This mismatch wastes tokens and can mislead the model. Repeated reads consume prompt space on every later call. Stale reads are worse: they may still look relevant even though they contradict the repository. In other words, whether an observation is useful is different from whether it is still true. Existing agent-memory and prompt-compression methods select history by recency, importance, relevance, or learned policies (Park et al., 2023; Packer et al., 2024; Jiang et al., 2024; Kang et al., 2026), but they do not check whether a retained observation still matches the environment. Coding agents need both relevance and *validity*. Figure 1 contrasts this requirement with append-only history and relevance-only compression.

Centralizing file state removes repeated historical copies without paraphrasing source code. Keeping one current copy of each file might then seem sufficient, but agents rarely read whole files. They usually inspect a function, a search result, or a short line range. In 8,999 baseline SWE-bench Verified trajectories, 89.5% of file reads are partial, and 73.3% of files are never read in full (Figure 3). Replacing that small observation with the full file can add thousands of lines the agent never requested. The challenge is therefore to keep *only the inspected regions* current.

This is harder than tracking line numbers. An insertion can move a region, an edit can change it, and a rewrite can remove it. Applying a diff is not enough because the agent may have seen only a small part of the old file. Loose text matching is also unsafe because similar code may appear in several places. The system must update a partial view without silently turning it into a different region.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

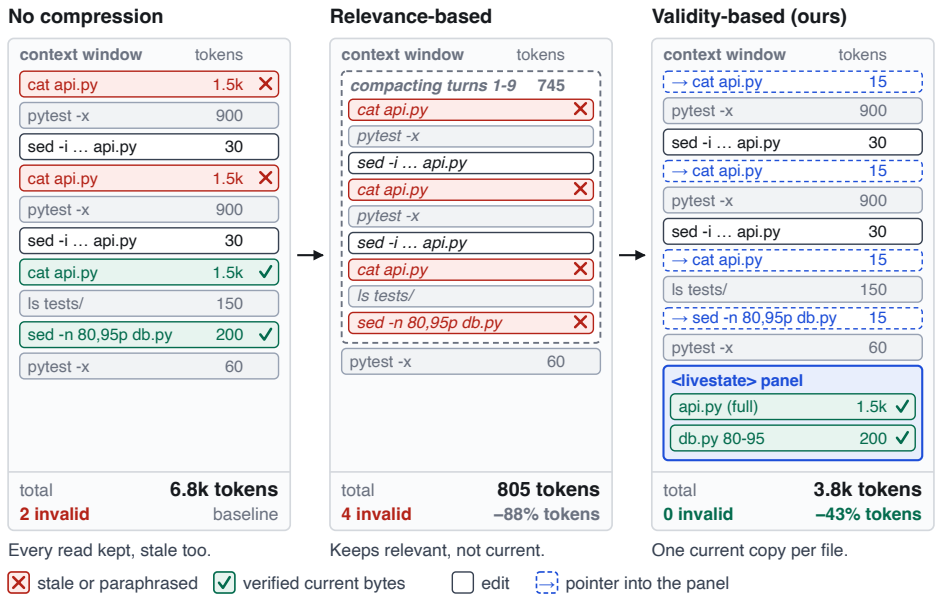

Figure 1: Three ways to keep file reads in context (schematic example; commands and counts are illustrative). Append-only history re-sends duplicate and stale reads. Relevance-based compaction shrinks the prompt but can keep stale code. LiveState replaces each tracked read with a pointer into one panel holding verified current bytes of only the lines the agent read.

We introduce **LiveState**, a training-free layer that maintains a *panel* of the file regions (*windows*) the agent has read. Before each model call, old read outputs become short pointers and the panel is shown once at the end of the prompt. When a file changes, *content anchoring* searches for each old region near its previous location, checks that enough of it still matches, and limits how far the region may expand. LiveState refreshes safe matches from current bytes and hides uncertain ones. The model and the agent's shell interface are unchanged.

We evaluate LiveState on all 500 SWE-bench Verified tasks (Jimenez et al., 2024) with eight model deployments from Anthropic, OpenAI, and Google, and on 113 longer-horizon DeepSWE tasks (Huang et al., 2026). LiveState reduces turns in every deployment, while its effect on accuracy depends on the model (Table 1). Claude 5 Fable matches baseline pass@1 ($+0.1$ points) with 17.0% fewer turns; Opus, Sonnet, and Haiku trade 1.4–3.3 points for 23–48% fewer tokens; and Gemini 3 Flash gains 6.7 points (paired 95% CI $[+4.3, +9.2]$) with roughly unchanged token use. On DeepSWE, Fable uses 37.7% fewer turns and 33.3% fewer tokens with pass@1 within 0.4 points, and resolves 43.4% versus 24.5% of task–seed pairs within an 8M-token budget. Behavioral analysis shows how the regimes differ: Opus, Sonnet, and Haiku stop earlier and test less, whereas Gemini spends less on routine successes and more on the hard tasks it newly solves (Section 4.6). An offline replay confirms the mechanism itself: LiveState relocates 97.9% of $146{,}381$ previously read windows, and all $67{,}530$ displayed views of changed files match the current file bytes (Section 4.7).

Our contributions are:

- We separate the *validity* of an observation (whether it still matches the repository) from its relevance, and formulate context management as keeping every file region the agent read valid, without widening it to lines the agent never saw.
- We introduce LiveState and its content-anchoring procedure, which reconciles independently observed file windows after edits while bounding how far a window can grow and withholding uncertain regions.
- We evaluate LiveState across eight deployments from three providers and on longer-horizon DeepSWE tasks. It reduces turns in all eleven model benchmark comparisons, and its accuracy and token effects fall into three regimes: preserved accuracy with fewer turns, an accuracy efficiency trade-off, and higher accuracy without token savings.

- We validate the mechanism directly with exact offline replay, which checks every displayed view against the current bytes and localizes rare defects that task accuracy cannot reveal.

2 Related Work

Context synchronization. Concurrent work, CORVUS (Zheng et al., 2026), also replaces stale file observations, using a synchronized registry that holds at most one current copy of each file. LiveState differs in granularity and interface: it keeps current only the regions exposed by partial reads, and it requires no agent-invoked synchronization tool. Work on memory updates also recognizes that retained information can become invalid. TEPA revokes superseded memories (Zhou et al., 2026a), while Supersede trains agents to handle memory updates (Patel, 2026). LiveState specializes this problem to file observations, for which the current repository supplies an external source of bytes against which a stored view can be reconciled.

Agent memory and prompt compression. Agent-memory systems retrieve or consolidate experience according to recency, importance, relevance, or learned policies (Packer et al., 2024; Park et al., 2023; Chhikara et al., 2025; Xu et al., 2025). Context can also be managed through explicit agent actions (Liu et al., 2026) or learned compact representations (Zhou et al., 2026b; Kang et al., 2026; Gao et al., 2026). Prompt compressors prune or encode tokens according to information content or query relevance (Jiang et al., 2023; 2024; Li, 2023; Mu et al., 2023; Ge et al., 2024). These approaches primarily decide *what is useful*; LiveState asks the orthogonal question of whether a concrete observation still corresponds to the mutable state from which it was read. The two can therefore be composed: a synchronization layer can maintain current observations before a memory or compression policy selects among them.

Coding-agent interfaces and inference efficiency. Coding-agent interfaces already constrain observations through windowed file viewers and context condensation (Yang et al., 2024; Wang et al., 2025), but typically preserve the resulting observations in an append-only interaction history. KV-cache methods instead reduce the cost of processing a fixed token sequence by evicting, selecting, or reallocating cached entries (Zhang et al., 2023; Xiao et al., 2024; Li et al., 2024; Cai et al., 2025). LiveState changes the sequence itself by replacing superseded observations with a current partial view. Because this transformation can also disrupt reusable prompt prefixes, submitted-token savings need not translate directly into serving savings; TokenPilot studies this cache-continuity trade-off explicitly (Xu et al., 2026).

3 Method

3.1 LiveState as a synchronized file panel

Coding agents normally keep every file read in an append-only transcript. This creates two problems: the same text may appear many times, and an edit can make earlier copies wrong. LiveState instead maintains a small *panel* with one *entry* per file the agent has read. Each entry stores a stable ID, the *windows*—line ranges the agent actually saw—and the latest verified text for those windows.

Before every model call, LiveState updates this panel from the current repository. It then replaces old read outputs with short pointers to their panel entries and appends the panel once at the end of the prompt. LiveState does not summarize code or decide what is important. It simply keeps one current copy of the code the agent has already inspected. Figure 2 shows the full path from a shell command to the next model context; the remaining subsections follow it in order.

Three safety rules guide the update. First, LiveState displays text only when it can verify that the text is current. Second, it never shows lines outside the windows the agent read, except edits made inside them. Third, when it cannot identify an old window after an edit, it hides that window and asks the agent to read it again.

3.2 Capturing repository state in the same execution

Git as the change oracle. A key design choice is to detect file changes from repository state rather than infer them from the agent's command. Shell commands can write through scripts, redirections,

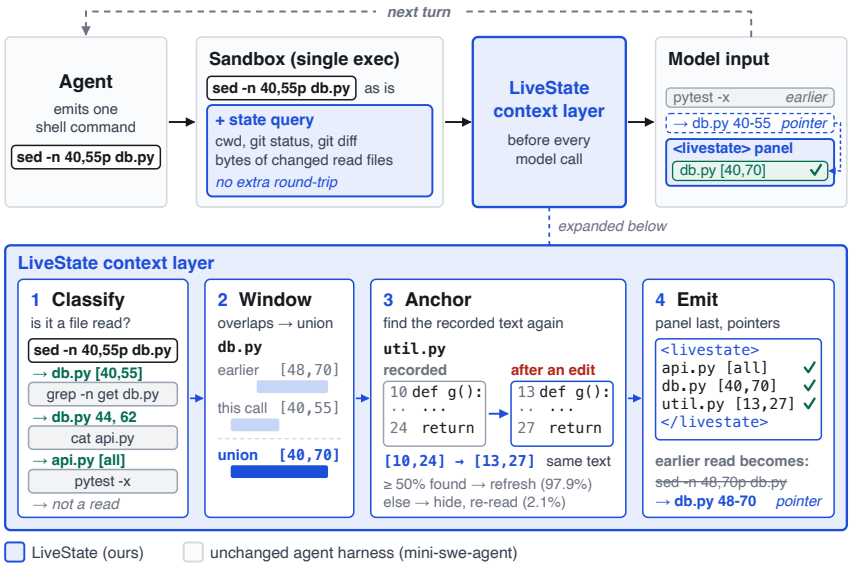

Figure 2: LiveState's per-turn pipeline. The agent's command runs unchanged; a state query appended to the same execution returns Git status and the current bytes of changed files the agent has read. Before the next model call, LiveState (1) classifies file reads, (2) merges ranges into windows, (3) re-anchors windows in changed files and hides failed matches, and (4) replaces old reads with pointers to one panel at the end of the prompt. The 97.9%/2.1% split is from the offline audit (Section 4.7).

formatters, or tests, making command-level write detection incomplete. After each command, the harness records the resolved working directory and runs `git status`, `git diff HEAD`, and `git rev-parse HEAD`. These queries identify the changed paths and repository revision; the harness then captures the current bytes of changed files that already have a LiveState entry. Git therefore determines *which files may be stale*, while the captured bytes provide the content against which their windows are refreshed.

No additional executor round trip. The Git queries are appended to the agent's shell invocation rather than issued as a second sandbox call. Their output is separated by sentinels and removed before the observation reaches the model, so the model sees the original command output and exit status. To bound the added transfer, LiveState returns content only for changed files the agent has read, and skips binary files, files over 200 KB, and files over 4000 lines. A deleted or unavailable file receives an explicit unavailable marker instead of retaining old text.

3.3 Classifying Reads into Windows

Partial reads are the common case. The sparse representation follows how coding agents actually inspect repositories. Across 8,999 baseline trajectories, we identify 73,398 successful file-read turns; the structural classifier assigns 97.6% of them unambiguously to partial or whole-file reads. Among these classifiable turns, 89.5% are partial, with a range of 87.1–99.4% across all eight model deployments. Partial reads also account for an estimated 78.2% of the cumulative prompt exposure attributable to classifiable read bodies (Figure 3).

The file lifecycle in Figure 3b makes whole-file replacement a poor default. Among classifiable file–trajectory pairs, 77.0% begin with a partial read. Of these partial-first files, 95.2% remain partial-only (73.3% of all files), while only 4.8% are later read in full. Among the partial-only group, 60.2% receive multiple partial reads, directly motivating an interval union that accumulates newly observed ranges. Separately, 23.0% of apparent whole-file read outputs are truncated by the harness; this statistic is not the 23.0% whole-first branch in the figure. Appendix Table 6 reports the complete accounting.

LiveState tracks only reads for which it can identify one file and the returned line range. Ambiguous, empty, failed, or harness-truncated observations remain unchanged in the transcript. An untruncated

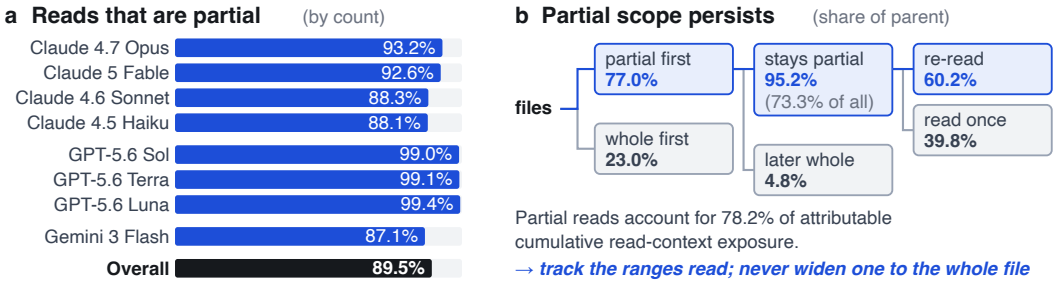

Figure 3: Partial reads dominate file access in 8,999 SWE-bench Verified baseline trajectories. **A:** Share of classified read turns that are partial, overall and per deployment. **B:** File-level access lifecycle; each percentage is relative to its parent box. The 78.2% exposure share is estimated from prompt-level token counts.

whole-file read records its complete returned body, a bounded read records that range, and a single-file `grep -n` records only the numbered lines it printed; searches without line numbers or across several files stay inline. Overlapping or adjacent ranges are merged, but unseen surrounding lines are never added. Reading the same window again therefore adds only another pointer, not another copy of the code.

3.4 Re-anchoring windows in changed files

After an edit, old line numbers may point to the wrong code. LiveState therefore uses the previously read text as an anchor, comparing lines with leading and trailing whitespace stripped so that re-indentation does not lose the window. For a window of n lines, three rules govern the match: (i) search first within $\max(60, 3n)$ lines on each side of the old location, then the whole file; (ii) accept only if at least half of the old non-blank lines remain; and (iii) reject if the matched span exceeds $3n + 30$ lines. The refreshed window runs from the first surviving line to the last, so edits inside it appear in the new view, while rule (iii) keeps a few common lines from pulling a large unrelated section into the prompt. Appendix A gives the pseudocode.

Each window is updated independently. If LiveState finds it, the panel displays the raw current bytes at the new location. If it cannot find it safely, only that window is hidden; other windows from the same file remain visible. An unchanged file needs no relocation. A changed file is shown only from a captured current-file payload. Repeated identical code can still make a location ambiguous, which is why the search favors the previous position and rejects weak or overly wide matches.

3.5 Emitting pointers and the panel

LiveState renders the surviving ranges with line numbers and labels unread gaps explicitly. Each tracked historical read is replaced by a stable pointer, for example `<output (lines 40--60 of path) is in <livestate> entry 3>`, and the corresponding entry is shown once in the final user message. A fixed prompt explains how to use the panel. An approximate 24,000-character budget bounds source bodies; when exceeded, least-recently-read entries become labelled header-only stubs, while the most recently read file is always kept.

Evaluated configuration. Our experiments evaluate the complete LiveState system: sparse-view construction, hidden state capture, content anchoring, panel instructions, and an inherited repeated-command backstop. The backstop warns after three identical command outcomes within six commands; forced submission is disabled. Because the baseline contains neither this warning nor the panel instruction, observed turn and re-read differences are system-level effects rather than isolated effects of anchoring. Two rare edge cases, both involving edited files, are described in Section 4.8.

4 Experiments

We first measure how LiveState changes task success, turns, and tokens across model families and on longer-horizon tasks, then use paired outcomes, agent behavior, and exact state replay to explain when it helps or hurts.

Table 1: SWE-bench Verified results. Each model's baseline is followed by the same agent with LiveState. Adjacent Δ columns report changes relative to the corresponding baseline; pass@1 uses percentage points and negative resource changes denote savings; favorable changes are bold green and others bold black. Multi-seed entries are means over completed matched seeds; full mean ± standard deviation results appear in Appendix Table 5.

Agent	Seeds	Pass@1 (%)		Turns/traj.		Tokens/traj.	
		Value	Δ pp	Value	Δ (%)	Value	Δ (%)
Claude 5 Fable	3	82.5		19.48		110.6K	
+ LiveState		82.7	**+0.1**	16.14	**−17.0**	109.0K	**−1.0**
Claude 4.7 Opus	3	79.8		20.87		248.8K	
+ LiveState		77.2	**−2.6**	15.75	**−24.5**	190.1K	**−23.5**
Claude 4.6 Sonnet	3	71.7		30.96		431K	
+ LiveState		70.3	**−1.4**	22.79	**−26.4**	333K	**−22.6**
Claude 4.5 Haiku	3	60.5		56.81		1.0M	
+ LiveState		57.2	**−3.3**	37.01	**−34.8**	522K	**−47.9**
GPT-5.6 Sol	1	84.0		17.46		168.0K	
+ LiveState		84.0	**0.0**	17.08	**−2.2**	163.2K	**−2.9**
GPT-5.6 Terra	1	75.4		17.09		163.3K	
+ LiveState		76.6	**+1.2**	14.85	**−13.1**	136.5K	**−16.4**
GPT-5.6 Luna	1	69.8		13.56		129.4K	
+ LiveState		69.8	**0.0**	12.38	**−8.7**	111.6K	**−13.7**
Gemini 3 Flash	3	54.9		27.05		271.7K	
+ LiveState		61.6	**+6.7**	25.55	**−5.6**	275.1K	**+1.2**

4.1 Experimental setup

Benchmarks. Our main evaluation uses all 500 tasks in SWE-bench Verified (Jimenez et al., 2024). Each task presents a real repository and an issue description; a task is resolved only when the submitted patch passes its task-specific tests. Unless noted otherwise, missing trajectories, infrastructure failures, and empty submissions count as unresolved in the 500-task denominator. We also report evaluations on DeepSWE v1.1 (Huang et al., 2026), which contains 113 longer-horizon tasks spanning 91 repositories and five programming languages.

Models. We evaluate Claude 4.7 Opus, 4.6 Sonnet, 4.5 Haiku, and 5 Fable; GPT-5.6 Sol, Terra, and Luna; and Gemini 3 Flash on SWE-bench Verified, and Claude 5 Fable, Claude 4.6 Sonnet, and GPT-6 Astra on DeepSWE. All comparisons are made within a model and deployment, never across models. SWE-bench Verified runs use a controlled `mini-swe-agent` configuration, and Gemini 3 Flash additionally uses a fixed short-reasoning budget, so absolute pass@1 is not directly comparable to leaderboard systems with different scaffolds or inference-time budgets.

Systems. The baseline retains the ordinary append-only transcript. LiveState applies the transformation in Section 3: modeled file-read outputs become stable pointers to a synchronized, bounded panel, in the evaluated configuration of Section 3.5.

Metrics and aggregation. We report pass@1, model calls per trajectory (*turns*), and total prompt-plus-completion tokens from the saved trajectories, excluding retries and restarts. For multi-seed experiments, changes are computed within each matched seed and then averaged across seeds. Accuracy differences are descriptive unless tested with a task-clustered sign randomization test (all seeds of a task resampled together) with Holm correction across deployments.

4.2 Main result: three response regimes

Table 1 consolidates the SWE-bench Verified comparisons. The evaluated system uses fewer turns in every row, but this common direction produces three distinct regimes.

Table 2: DeepSWE v1.1 results (113 tasks, three seeds per configuration). Pass@1 pools resolved over graded task–seed pairs; resource deltas are computed within seed on the shared graded set and then averaged. Grade coverage and per-seed variability appear in Appendix Table 12.

Agent	Pass@1 (%)		Turns/traj.		Tokens/traj.	
	Value	Δ pp	Value	Δ (%)	Value	Δ (%)
Claude 5 Fable	70.6		82.63		11.99M	
+ LiveState	70.2	**−0.4**	51.51	**−37.7**	8.00M	**−33.3**
Claude 4.6 Sonnet	35.8		117.57		11.99M	
+ LiveState	36.4	**+0.6**	68.35	**−41.8**	9.31M	**−22.0**
GPT-6 Astra	71.2		26.30		1.84M	
+ LiveState	71.7	**+0.5**	25.24	**−4.1**	2.07M	**+12.6**

Efficiency without observed accuracy loss. Fable and the three GPT-5.6 deployments occupy the most favorable regime: they keep observed pass@1 unchanged or higher while using fewer turns and tokens, although Fable's token saving is small and variable across seeds. These results indicate that synchronized context need not impose an accuracy degradation, although additional GPT seeds are required to estimate run-to-run variation.

An explicit efficiency–quality trade-off. Opus, Sonnet, and Haiku trade 1.4–3.3 points of observed pass@1 for 24–35% fewer turns and 23–48% fewer tokens. The ordering is not monotonic in model capability: Fable is both the strongest Claude deployment and the one with the smallest observed accuracy change.

Behavioral improvement without token compression. Gemini defines a third regime. Pass@1 increases by 6.7 points and turns decrease by 5.6%, but total tokens increase by 1.2%. Thus neither turn count nor task success can be inferred from aggregate token compression alone.

Paired inference distinguishes replicated effects from sampling noise. Task-clustered 95% confidence intervals separate the Gemini gain ($+6.7$ points, $[+4.3, +9.2]$), Opus loss (-2.6, $[-4.3, -0.9]$), and Haiku loss (-3.3, $[-5.5, -1.4]$) from zero. The Fable and Sonnet intervals cross zero, as do all three one-seed GPT intervals. We therefore treat the GPT accuracy changes as descriptive rather than established improvements. Full paired counts and multiplicity-corrected tests appear in Appendix D.

Outcome changes are broad rather than task-specific. Aggregate pass@1 also hides turnover in which tasks are solved. Across the 18 model–seed comparisons, LiveState solves a task more often than the baseline for 165 tasks and less often for 161, and most of these net differences come from a single comparison (Appendix Figure 5). The changes therefore reflect broad sensitivity to the representation rather than a single class of consistently broken tasks.

4.3 Transfer to longer-horizon tasks

DeepSWE is our primary transfer evaluation: its 113 tasks use longer trajectories, multiple programming languages, and a native tool interface rather than the SWE-bench Verified scaffold. We compare only within each DeepSWE model and configuration. Table 2 shows that the main efficiency result becomes stronger for Claude 5 Fable: pooled accuracy changes by -0.4 points while turns fall by 37.7% and tokens by 33.3%, with savings that are stable across seeds. The other two models differ in token use. Sonnet changes pooled accuracy by $+0.6$ points while turns fall by 41.8% and tokens by 22.0%. GPT-6 Astra changes pooled accuracy by $+0.5$ points and reduces turns by 4.1%, but uses 12.6% more tokens. Thus the turn reduction transfers to all three DeepSWE models, but token effects remain model dependent.

4.4 Resolution under finite inference budgets

A large context window does not remove the computational constraint of a long agent trajectory. Even with a 1M-token context window, the model can process tens of millions of cumulative tokens because each turn submits another prompt. We therefore measure *budgeted resolution*: at budget

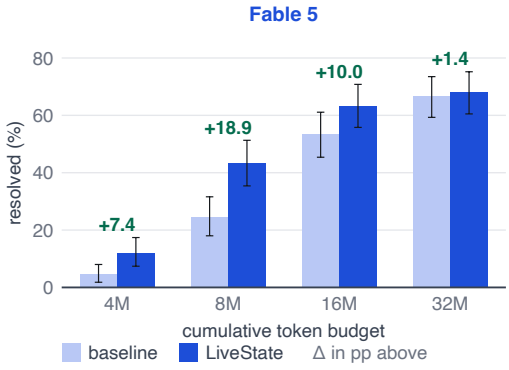

Figure 4: DeepSWE resolution under a cumulative token budget (solid: LiveState; dashed: baseline; three seeds pooled; missing grades count as unresolved; shading: 95% task-clustered bootstrap CI). LiveState's advantage is largest at tight budgets (+18.9 points for Fable at 8M tokens).

Table 3: Efficiency on tasks solved by both systems. Positive values denote mean LiveState savings across seeds. This post-treatment analysis is descriptive; Appendix Table 8 reports full variability.

Model	Shared	Turn Δ	Token Δ
Claude 4.7 Opus	1,102	**24.6%**	**23.0%**
Claude 5 Fable	1,128	**15.2%**	**2.6%**
Claude 4.6 Sonnet	972	**22.6%**	**25.7%**
Claude 4.5 Haiku	761	**33.6%**	**45.8%**
Gemini 3 Flash	699	**14.5%**	**11.9%**

B, a task counts as solved only if its trajectory is grader-resolved and uses at most B total prompt-plus-completion tokens. Figure 4 shows that LiveState shifts the DeepSWE resolution curves toward lower budgets. At 8M cumulative tokens, Fable resolves 43.4% with LiveState versus 24.5% with the baseline, an 18.9-point advantage. The gaps narrow at 16M tokens to 10.0 points, respectively, and nearly disappear by 32M tokens. This convergence is consistent with the similar final pass@1 values in Table 2: LiveState reaches a similar final resolution rate at substantially lower inference budgets, rather than raising the final ceiling.

4.5 Efficiency on jointly solved tasks

Differences in success can distort average computational usage: one arm may terminate early on tasks that the other solves, or may spend additional calls reaching a successful patch. Therefore, we recompute efficiency on the intersection of tasks resolved by both arms. Table 3 reports total shared task–seed pairs and averages the savings computed separately in each seed. On shared successes, the Claude savings largely survive: Opus, Sonnet, and Haiku still use substantially fewer turns and tokens, and Fable keeps its turn reduction with a small token change. Gemini, whose all-task token use is roughly unchanged, becomes more efficient on jointly solved tasks, although its token saving varies across seeds. LiveState therefore does not reduce computation for every model, but the Claude savings are robust to outcome matching: they are not explained by failed trajectories alone.

4.6 Two sources of efficiency: state reuse and trajectory adaptation

LiveState can reduce inference use through two channels. *State reuse* replaces repeated reads with one current file view, partly offset by panel instructions, pointers, and line-number metadata. *Trajectory adaptation* changes the agent's policy, for example by reducing re-reads, tests, or follow-up actions. Only the first is mechanical compression; the second can save more but can also change which tasks are solved. An offline accounting decomposition (Appendix H) shows that on the action sequences LiveState induces, the panel's metadata slightly outweighs the raw reads it replaces (0.3–7.0% of baseline tokens), whereas trajectory shortening contributes 3.7–48.8% and accounts for every positive net reduction in Table 1; this is an accounting identity, not a causal split. Paired replay further shows that most accuracy flips diverge before any panel is displayed, ruling out corrupted panel content as a general explanation (Appendix K).

Behavioral savings depend on the model. Table 4 compares each LiveState trajectory with its baseline partner, separately for tasks both systems solve and tasks only one solves. It asks whether LiveState merely removes redundant reads or changes how long the agent keeps exploring and testing.

The Claude trade-off is associated with broad contraction. Opus, Sonnet, and Haiku perform 41–57% fewer tests even on shared successes, and use fewer actions on both regressions and gains. LiveState

Table 4: Change in LiveState behavior relative to the paired baseline. **Shared success**: solved by both; **Regression**: solved only by the baseline; **Gain**: solved only by LiveState. Negative values mean that LiveState performs fewer actions. Outcome groups are post-treatment diagnostics, not causal estimands. Full counts and metrics appear in Appendix G.

	Shared success		Regression	Gain
Model	**Re-reads Δ**	**Tests Δ**	**Agent actions Δ**	**Agent actions Δ**
Claude 4.7 Opus	−27.6%	−57.4%	−33.5%	−33.2%
Claude 5 Fable	−7.5%	+0.4%	−40.1%	+33.3%
Claude 4.6 Sonnet	−32.2%	−41.1%	−30.5%	−44.6%
Claude 4.5 Haiku	−37.5%	−47.1%	−34.6%	−42.1%
Gemini 3 Flash	−33.4%	+4.5%	−23.0%	+31.7%

therefore does more than eliminate duplicate reads for these models: it encourages earlier closure, which saves substantial computation but can skip useful exploration or validation.

Gemini allocates effort selectively instead. On shared successes it performs 33.4% fewer re-reads, but on the tasks it newly solves it performs 31.7% more agent actions than the failed baseline. Gemini thus spends less on routine successes and more on branches that become additional solves, which explains why accuracy improves without an overall token reduction. Fable shows a weaker version of this split, while GPT-5.6 responses are small and heterogeneous (Appendix G).

4.7 Does LiveState display the current file contents?

Task accuracy cannot show whether the synchronization itself is correct, so we replay every model call whose prompt contains a file panel in 17 SWE-bench Verified model–seed comparisons, and compare each displayed line with the file snapshot taken after the preceding command. Of 146,381 previously read windows, LiveState relocates 97.9% and withholds the other 2.1% rather than guessing. All 67,530 accepted views of changed files match the current bytes exactly: whenever LiveState retains a window of a changed file, it shows current rather than stale text. Appendix J reports the full audit and the rare exceptions it finds.

4.8 Limitations

Instruction prior and edge cases. LiveState's instruction prefix precedes its first file panel, so the intervention includes both a representation and an instruction prior. Offline replay also finds two rare edge cases: a reverted edited file can show its earlier edited text until re-read, and re-reading a large edited file can fail to restore its hidden view.

Logical tokens are not serving efficiency. Submitted-token reductions do not imply proportional savings in latency, computation, or cost. Updating the panel can invalidate a reusable prompt prefix, and cache-read share falls substantially in the Claude logs (Appendix Table 15); we therefore report tokens and turns as model-facing usage, not serving efficiency.

5 Conclusion

Append-only transcripts are a poor fit for agents acting on mutable repositories: they repeatedly carry file observations that may be duplicated or stale. LiveState replaces those observations with a synchronized view of only the file regions the agent inspected, refreshing identifiable windows from current bytes and hiding uncertain ones; it requires no training and leaves the model and the agent's shell interface unchanged. It reduces turns in all eight SWE-bench Verified deployments and all three DeepSWE models, and on DeepSWE Fable keeps pass@1 within 0.4 points with 37.7% fewer turns, but the effects on accuracy and tokens depend on the model. This heterogeneity shows that synchronized state is not interchangeable with generic token compression: its value must be measured through agent behavior, task outcomes, and, as our exact replay shows, direct state-level validation. More broadly, our results support treating mutable environment observations as state to be maintained rather than history to be replayed. Natural next steps include cache-aware panel updates, adaptive selection of which entries to show, and separate compression of the remaining non-file history (Appendix M).

AI USE STATEMENT

Large language models are the object of study in this work: all evaluated agents are hosted LLMs running in `mini-swe-agent`, as described in Section 4.1. Separately, the authors used LLM to edit and restructure the manuscript text. The authors reviewed all AI-assisted code and text, checked reported numbers against the saved experiment outputs, and take full responsibility for the content of the paper.

ETHICS STATEMENT

This work involves no human subjects or personal data. All experiments use public benchmarks (SWE-bench Verified and DeepSWE) built from open-source repositories, and every agent command runs in an isolated container. LiveState reduces the context submitted to coding agents; we do not foresee direct harms beyond those of the coding agents themselves. We note that the evaluation consumes substantial inference compute, and that token savings do not necessarily translate into proportional energy or cost savings (Section 4.8).

REPRODUCIBILITY STATEMENT

Section 3 specifies LiveState, and Appendix A gives pseudocode for content anchoring, including all thresholds. Section 4.1 describes the benchmarks, models, agent scaffold, and metrics. Appendices B and I report per-seed results and grade coverage, and Appendix D describes the paired statistical tests. We will release the LiveState implementation, the agent configuration including the panel instruction prompt, and the analysis scripts upon publication. Because the evaluated models are served through hosted APIs whose versions may change, exact reproduction of absolute scores may not be possible. However, every comparison in the paper is made between arms of the same deployment.

REFERENCES

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. In *Conference on Language Modeling*, 2025. URL https://openreview.net/pdf/40900f3b1b3442ab76c9934ef2acc1c2a3ebb3e5.pdf.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Shuzheng Gao, Wenhao Zeng, Zhaojian Yu, Jianqiao Wangni, Chaozheng Wang, Kai Cai, Shilin He, and Michael R. Lyu. Swe-mem: Learning adaptive memory management for long-horizon coding agents, 2026. URL https://arxiv.org/abs/2606.28434.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uREj4ZuGJE.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. Deepswe: Measuring frontier coding agents on original, long-horizon engineering tasks, 2026. URL https://arxiv.org/abs/2607.07946.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. doi: 10.18653/v1/2024.acl-long.91. URL https://aclanthology.org/2024.acl-long.91/.

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*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents, 2026. URL https://arxiv.org/abs/2510.00615.

Yucheng Li. Unlocking context constraints of llms: Enhancing context efficiency of llms with self-information-based content filtering, 2023. URL https://arxiv.org/abs/2304.12102.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, 2024. URL https://neurips.cc/virtual/2024/poster/93531.

Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. Context as a tool: Context management for long-horizon swe-agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. doi: 10.18653/v1/2026.findings-acl.1032. URL https://aclanthology.org/2026.findings-acl.1032/.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/3d77c6dcc7f143aa2154e7f4d5e22d68-Abstract-Conference.html.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Vedant Patel. Supersede: Diagnosing and training the memory-update gap in llm agents, 2026. URL https://arxiv.org/abs/2606.27472.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, 2025. URL https://iclr.cc/virtual/2025/poster/29831.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://iclr.cc/virtual/2024/poster/18794.

Buqiang Xu, Zirui Xue, Dianmou Chen, Chenyang Fu, Chiyu Wu, Caiying Huang, Chen Jiang, Jizhan Fang, Xinle Deng, Yijun Chen, Yunzhi Yao, Xuehai Wang, Jin Shang, Gong Yu, and Ningyu Zhang. Tokenpilot: Cache-efficient context management for llm agents, 2026. URL https://arxiv.org/abs/2606.17016.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Algorithm 1 Content anchoring for one previously read window. ALIGN returns matching line blocks; COVERAGE is the fraction of old non-blank lines contained in those blocks.

Require: old window text C, old interval $I = [s, e]$, current file F

function REFRESHWINDOW(C, I, F)
  $n \leftarrow |C|$; $C^\circ, F^\circ \leftarrow$ STRIPEACHLINE(C, F)
  if C° has no non-blank line **then**
    return NOTFOUND
  end if
  $r \leftarrow \max(60, 3n)$
  $J \leftarrow [\max(1, s - r), \min(|F|, e + r)]$
  $M \leftarrow$ ALIGN($C^\circ, F^\circ[J]$)
  if COVERAGE(M, C°) < 0.5 **then**
    $J \leftarrow [1, |F|]$; $M \leftarrow$ ALIGN(C°, F°) ▷ fall back to the full file
  end if
  if COVERAGE(M, C°) < 0.5 **then**
    return NOTFOUND
  end if
  $I' = [s', e'] \leftarrow$ FILESPAN(M, J) ▷ first through last matching block
  if $|I'| > 3n + 30$ **then**
    return NOTFOUND
  end if
  return $(I', F[s' : e'])$ ▷ return raw bytes, not normalized text
end function

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://neurips.cc/virtual/2023/poster/71645.

Mingwei Zheng, David O'Brien, Siwei Cui, Pardis Pashakhanloo, Rajdeep Mukherjee, Myeongsoo Kim, and Sachit Kuhar. Corvus: Context optimization and reduction via underlying synchronization for llm coding agents, 2026. URL https://arxiv.org/abs/2607.22711.

Yan Zhou, Yue Ouyang, Kaiyang Zheng, and Suncheng Xiang. Tepa: Revoking stale memories for conflict-robust language agents, 2026a. URL https://arxiv.org/abs/2608.07429.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=jJ6F1sDn9i.

A CONTENT-ANCHORING PROCEDURE

Algorithm 1 states the window-refresh procedure used by the evaluated implementation. All content returned to the model comes from the captured current file; per-line stripped text is used only to locate the old window.

B DETAILED MULTI-SEED RESULTS

Main-paper tables report seed means for readability. Here we report the corresponding mean $\pm$ sample standard deviation over completed matched seeds. Single-seed entries have no dispersion estimate.

Table 5: Full SWE-bench Verified results underlying Table 1. Values are mean ± sample standard deviation. Sonnet LiveState seed 2 and Haiku baseline seed 2 report accuracy over 490 and 491 returned grades, respectively.

Agent	Seeds	Pass@1 (%)		Turns/traj.		Tokens/traj.	
		Value	Δ pp	Value	Δ (%)	Value	Δ (%)
Claude 4.7 Opus	3	79.8 ± 0.4		20.87 ± 0.28		(248.8 ± 7.3)K	
+ LiveState		77.2 ± 0.9	$\mathbf{-2.6 \pm 0.6}$	15.75 ± 0.26	$\mathbf{-24.5 \pm 2.3}$	(190.1 ± 7.3)K	$\mathbf{-23.5 \pm 5.0}$
Claude 4.6 Sonnet	3	71.7 ± 1.3		30.96 ± 1.11		(431 ± 10.4)K	
+ LiveState		70.3 ± 1.0	$\mathbf{-1.4 \pm 2.3}$	22.79 ± 0.64	$\mathbf{-26.4 \pm 4.6}$	(333 ± 20.1)K	$\mathbf{-22.6 \pm 6.0}$
Claude 4.5 Haiku	3	60.5 ± 2.5		56.81 ± 1.01		$1.0\text{M} \pm 55.1\text{K}$	
+ LiveState		57.2 ± 0.7	$\mathbf{-3.3 \pm 2.9}$	37.01 ± 0.65	$\mathbf{-34.8 \pm 2.3}$	(522 ± 11.9)K	$\mathbf{-47.9 \pm 4.1}$
Claude 5 Fable	3	82.5 ± 0.9		19.48 ± 0.94		(110.6 ± 9.0)K	
+ LiveState		82.7 ± 0.9	$\mathbf{+0.1 \pm 0.9}$	16.14 ± 0.65	$\mathbf{-17.0 \pm 5.2}$	(109.0 ± 2.4)K	$\mathbf{-1.0 \pm 6.3}$
GPT-5.6 Sol	1	84.0		17.46		168.0K	
+ LiveState		84.0	**0.0**	17.08	**−2.2**	163.2K	**−2.9**
GPT-5.6 Terra	1	75.4		17.09		163.3K	
+ LiveState		76.6	**+1.2**	14.85	**−13.1**	136.5K	**−16.4**
GPT-5.6 Luna	1	69.8		13.56		129.4K	
+ LiveState		69.8	**0.0**	12.38	**−8.7**	111.6K	**−13.7**
Gemini 3 Flash	3	54.9 ± 1.6		27.05 ± 0.47		(271.7 ± 5.8)K	
+ LiveState		61.6 ± 0.6	$\mathbf{+6.7 \pm 2.0}$	25.55 ± 0.12	$\mathbf{-5.6 \pm 1.3}$	(275.1 ± 8.8)K	$\mathbf{+1.2 \pm 1.7}$

Table 6: Detailed accounting for Figure 3. Turn, operation, and file-lifecycle counts are exact under the structural parser. Token attribution is estimated because providers report tokens for complete prompts rather than individual observations.

Quantity	Partial	Whole file	Other/ambig.
Successful read turns	64,136	7,497	1,765
Read operations	69,383	8,005	2,449
Direct read tokens (est.)	21.97M	6.18M	12.79M
Cumulative exposure (est.)	673.0M	188.2M	255.0M

Statistic	Rate
Classifier coverage of read turns	97.6%
Partial among classifiable turns	89.5%
Files first opened partially	77.0%
Files never read whole	73.3%
Partial-first later read whole	4.8%
Partial-only with repeated partial reads	60.2%
Whole-file outputs truncated	23.0%

C Partial-Read Prevalence

We apply LiveState's production structural read parser to the 8,999 baseline trajectories represented in Figure 3. Failed commands are excluded. A read turn is *partial* when it is an explicit range, tail window, or search, and *whole-file* when it is an unambiguous single-file full read. Compound outputs that cannot be assigned safely remain in the other/ambiguous category.

D Paired Accuracy Inference

Table 7 pairs outcomes by benchmark task and seed. Confidence intervals resample tasks with all available seeds attached; p-values use the analogous task-clustered sign randomization and Holm correction across the eight deployments. This avoids treating repeated seeds of the same benchmark task as independent observations.

Table 7: Paired SWE-bench Verified outcome inference. B-only and LS-only count discordant task–seed outcomes. Haiku has 1,491 rather than 1,500 comparable outcomes because its seed-2 baseline has 491 returned grades; Sonnet has 1,490 because its third LiveState seed returned 490 grades.

Model	Seeds	Pairs	B-only	LS-only	Δ pp [95% CI]	Holm p
Claude 5 Fable	3	1,500	110	112	$+0.13$ $[-1.80, +2.13]$	1.000
Claude 4.7 Opus	3	1,500	87	48	-2.60 $[-4.33, -0.87]$	0.027
Claude 4.6 Sonnet	3	1,490	95	75	-1.34 $[-3.28, +0.54]$	0.943
Claude 4.5 Haiku	3	1,491	141	90	-3.42 $[-5.45, -1.40]$	0.0099
GPT-5.6 Sol	1	500	17	17	0.00 $[-2.40, +2.20]$	1.000
GPT-5.6 Terra	1	500	28	34	$+1.20$ $[-1.80, +4.40]$	1.000
GPT-5.6 Luna	1	500	34	34	0.00 $[-3.20, +3.20]$	1.000
Gemini 3 Flash	3	1,500	124	225	$+6.73$ $[+4.27, +9.20]$	< 0.001

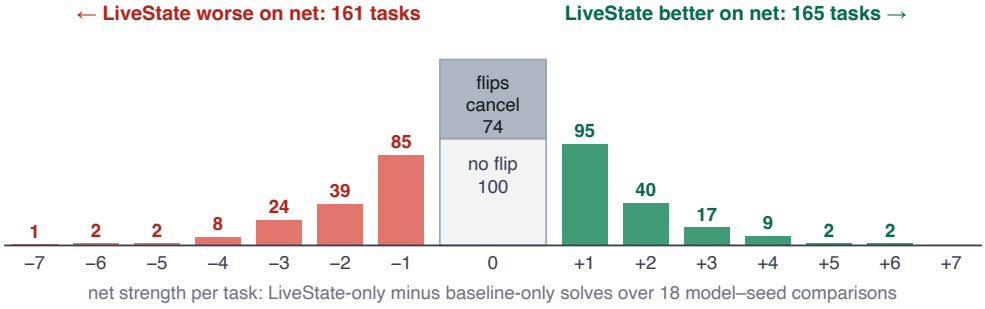

Figure 5: Task-level net strength across the 18 SWE-bench Verified model–seed comparisons. For each task, net strength is the number of LiveState-only solves minus baseline-only solves. Red and green bars give the exact number of tasks at each negative or positive value; the center separates 74 tasks whose observed flips cancel from 100 tasks with no observed flip. Overall, LiveState is better on net for 165 tasks and worse for 161. Most nonzero values are a single comparison (85 tasks at -1, 95 at $+1$), and only nine tasks reach an absolute strength of five. Missing grader reports count as unresolved. Because models contribute different numbers of seeds, this aggregation describes outcome turnover rather than an overall accuracy estimate.

E Task-Level Outcome Turnover

Figure 5 gives the full signed distribution of the task-level net strength summarized in Section 4.2.

F Shared-Resolve Efficiency

The all-task averages can change when one arm solves tasks on which the other terminates early. We therefore repeat the efficiency comparison on task–seed pairs resolved by both systems. This restriction does not estimate a causal effect because joint success is determined after treatment; it tests whether the reported savings persist when outcome composition is held fixed.

G Behavioral Diagnostics

We characterize trajectory changes separately for shared successes, baseline-only solves, and LiveState-only solves. Table 9 reports action changes in each outcome stratum, and Table 10 expands the Gemini measurements used in the main text. Because these strata are defined by final outcomes, the comparisons are descriptive rather than causal.

Table 8: Full shared-resolve efficiency statistics underlying Table 3. GPT-5.6 rows are single-seed and therefore report no deviation.

Model	Pairs	Turn saving	Token saving
Claude 4.7 Opus	1,102	$24.6 \pm 3.5\%$	$23.0 \pm 11.0\%$
Claude 5 Fable	1,128	$15.2 \pm 3.8\%$	$2.6 \pm 7.3\%$
Claude 4.6 Sonnet	972	$22.6 \pm 3.2\%$	$25.7 \pm 5.4\%$
Claude 4.5 Haiku	761	$33.6 \pm 2.6\%$	$45.8 \pm 4.4\%$
Gemini 3 Flash	699	$14.5 \pm 4.1\%$	$11.9 \pm 14.9\%$
GPT-5.6 Sol	403	1.9%	1.2%
GPT-5.6 Terra	349	0.8%	-3.3%
GPT-5.6 Luna	315	-3.2%	-11.0%

Table 9: Outcome-conditioned behavioral diagnostics. Shared columns are computed on tasks resolved by both arms. B-only and LS-only columns compare the LiveState trajectory with its paired baseline in the indicated asymmetric-outcome stratum. Selection on outcome makes these descriptive, not causal.

	Both resolve			B-only		LS-only		
Model	n	Re-read Δ	Test Δ	n	Action Δ	n	Action Δ	Test Δ
Claude 4.7 Opus	1,102	-27.6%	-57.4%	95	-33.5%	46	-33.2%	-54.1%
Claude 5 Fable	1,128	-7.5%	$+0.4\%$	110	-40.1%	112	$+33.3\%$	$+44.4\%$
Claude 4.6 Sonnet	972	-32.2%	-41.1%	103	-30.5%	75	-44.6%	-22.6%
Claude 4.5 Haiku	761	-37.5%	-47.1%	139	-34.6%	97	-42.1%	-58.6%
GPT-5.6 Sol	403	-5.7%	-12.0%	17	$+2.9\%$	17	-5.8%	$+25.0\%$
GPT-5.6 Terra	349	-6.5%	-0.2%	28	-6.7%	34	-44.8%	-14.3%
GPT-5.6 Luna	315	$+4.5\%$	-4.7%	34	$+1.2\%$	34	-45.2%	-44.0%
Gemini 3 Flash	699	-33.4%	$+4.5\%$	124	-23.0%	224	$+31.7\%$	$+169.0\%$

H State-Reuse and Trajectory-Path Decomposition

Let $T(B, P)$ be the observed token use of the baseline trajectory under the plain transcript, $T(L, S)$ the observed LiveState trajectory under synchronized state, and $T(L, P)$ a counterfactual replay of the same LiveState actions with their saved raw observations restored. We use the accounting identity

$$T(B,P) - T(L,S) = \underbrace{T(L,P) - T(L,S)}_{\text{state reuse}} + \underbrace{T(B,P) - T(L,P)}_{\text{trajectory path}}. \tag{1}$$

Actual endpoint token counts are recorded. Because proprietary tokenizers are unavailable, $T(L, P)$ is estimated using each request's observed token-to-visible-character ratio. The split is path-dependent and not causal: it assigns the interaction between the representation and the actions it induces to the trajectory term.

The result changes the interpretation of "compression." On the trajectories LiveState actually induces, the synchronized representation is modestly larger than restoring their relatively few raw reads. Net reductions arise because those trajectories are shorter. This does not imply that the representation is unnecessary: it may be what causes the agent to avoid later rereads and actions. A symmetric decomposition would additionally require applying LiveState to the baseline trajectory with exact post-command repository snapshots, which the baseline runs did not record.

I Detailed DeepSWE Results

DeepSWE tests whether the SWE-bench Verified behavior transfers to longer trajectories and a native tool interface. We report accuracy over all scored rollouts; accuracy deltas compare the displayed seed-level rates, while resource deltas use the shared scored set within each model and seed. Fable, Sonnet, and Astra each aggregate three seeds; the Sonnet aggregate uses seeds 0, 2, and 3. We exclude an earlier DeepSWE run because 57 LiveState trajectories terminated on a missing-snapshot adapter error, mechanically shortening them; all reported results come from the subsequent corrected runs.

Table 10: Additional Gemini 3 Flash behavioral statistics. Values are mean ± sample standard deviation over three seeds.

Metric	Baseline	LiveState	Δ
Pass@1 (%)	54.9 ± 1.6	61.6 ± 0.6	$\mathbf{+6.7 \pm 2.0}$ **pp**
Exclusive solves/seed	41.3 ± 7.1	75.0 ± 5.3	$\mathbf{+33.7 \pm 10.2}$
Turns/trajectory	27.05 ± 0.47	25.55 ± 0.12	$\mathbf{-5.6 \pm 1.3}\%$
Total tokens/trajectory	(271.7 ± 5.8)K	(275.1 ± 8.8)K	$\mathbf{+1.2 \pm 1.7}\%$
Context tokens/turn	(9.81 ± 0.31)K	(10.45 ± 0.37)K	$\mathbf{+6.6 \pm 3.1}\%$
Unique source files/trajectory	2.10 ± 0.06	2.09 ± 0.06	$\mathbf{-0.3 \pm 3.3}\%$
Source-file re-reads/trajectory	5.20 ± 0.44	3.96 ± 0.21	$\mathbf{-23.5 \pm 4.9}\%$
Post-edit re-reads/trajectory	0.48 ± 0.04	0.28 ± 0.02	$\mathbf{-40.4 \pm 8.6}\%$

Table 11: One-sided accounting decomposition over all paired SWE-bench Verified trajectories. Components are percentages of baseline total tokens and sum to the observed net saving up to rounding. Positive values denote savings; negative values denote added volume.

Model	Pairs	Net saving	State reuse	Trajectory path
Claude 4.7 Opus	1,500	+26.1%	−3.8%	+29.9%
Claude 5 Fable	1,500	+1.4%	−7.0%	+8.3%
Claude 4.6 Sonnet	1,500	+22.6%	−1.5%	+24.1%
Claude 4.5 Haiku	1,498	+48.1%	−0.2%	+48.3%
GPT-5.6 Sol	500	+2.9%	−2.4%	+5.2%
GPT-5.6 Terra	500	+16.4%	−1.4%	+17.8%
GPT-5.6 Luna	500	+13.7%	−0.9%	+14.7%
Gemini 3 Flash	1,499	−1.3%	−5.0%	+3.7%

The response remains model dependent. Fable reduces turns by 37.71% and tokens by 33.34% with a -0.40-point mean accuracy change. Sonnet shows similarly large reductions, whereas Astra reduces turns modestly but uses more tokens. These results support transfer of the turn reduction, not a model-independent claim about token use.

J Anchoring and Panel Lifecycle Audit

We replayed every panel-bearing model call in the *replay set*: the 17 SWE-bench Verified model–seed comparisons with complete reconstruction records (three seeds each for Fable, Opus, Sonnet, and Gemini, two for Haiku, and one per GPT-5.6 deployment). Each displayed line is compared with the file snapshot captured after the preceding command. We call a file *dirty* when its working-tree bytes differ from the last commit (HEAD); a *dirty-to-clean* transition is a dirty file returning to HEAD. Counts are exposure-weighted: a dirty view is checked again on every subsequent model call. Table 13 shows that rejection is uncommon and that accepted changed-file views are byte-exact, while also making the rare implementation defects explicit.

The 37 dirty-to-clean transitions occur in 22 model–task occurrences. Exact replay finds four baseline-only trajectories in which stale redisplay changes subsequent reasoning. A provenance guard that withholds such a view until a new read verifies it removes this case. Separately, the 929 failed rereads cluster in 156 model–task occurrences but only 29 tasks; their outcomes are 11 baseline-only, 19 LiveState-only, and 126 other. Keeping the newest successful reread inline for one turn, instead of replacing it with an unavailable pointer, removes this failure.

Exact duplicate fragments create a narrower addressing ambiguity. We find 230 distinct windows in which alignment chooses a non-nearest occurrence with identical normalized content. The bytes are current, and affected outcomes favor rather than disfavor LiveState (8 baseline-only versus 16 LiveState-only), but line identity may be wrong. A deterministic nearest-exact tie break removes this ambiguity without adding model-visible tokens.

Table 12: Full DeepSWE v1.1 statistics underlying Table 2. *Scored/expected* reports grade coverage. Pass@1 cells show pooled resolved/scored counts followed by the mean ± sample standard deviation of seed-level rates. Accuracy deltas are differences of the displayed seed means; resource deltas use within-seed paired sets. Every configuration has three seeds; Sonnet uses seeds 0, 2, and 3. Missing grades (6/7 for Fable baseline/LiveState, 1/4 for Sonnet, and 2/0 for Astra) are system errors rather than task failures.

Agent	Coverage	Accuracy		Turns/traj.		Tokens/traj. (M)	
	Scored/expected	Pass@1	Δ (pp)	Mean	Δ (%)	Mean	Δ (%)
Claude 5 Fable	333/339	235/333 $70.57 \pm 1.38\%$		82.63 ± 0.17		11.99 ± 0.13	
+ LiveState	332/339	233/332 $70.17 \pm 3.21\%$	$\mathbf{-0.40 \pm 4.31}$	51.51 ± 0.54	$\mathbf{-37.71 \pm 0.67}$	8.00 ± 0.15	$\mathbf{-33.34 \pm 1.87}$
Claude 4.6 Sonnet	338/339	121/338 $35.80 \pm 1.98\%$		117.57 ± 1.49		11.99 ± 0.38	
+ LiveState	335/339	122/335 $36.42 \pm 2.84\%$	$\mathbf{+0.62 \pm 1.51}$	68.35 ± 0.61	$\mathbf{-41.81 \pm 1.27}$	9.31 ± 0.44	$\mathbf{-22.01 \pm 3.88}$
GPT-6 Astra	337/339	240/337 $71.22 \pm 4.17\%$		26.30 ± 0.65		1.84 ± 0.09	
+ LiveState	339/339	243/339 $71.68 \pm 2.65\%$	$\mathbf{+0.46 \pm 1.77}$	25.24 ± 0.48	$\mathbf{-4.10 \pm 2.97}$	2.07 ± 0.08	$\mathbf{+12.60 \pm 2.30}$

Table 13: Offline anchoring and lifecycle audit of LiveState.

Quantity	Count (rate)
Relocation attempts	146,381
Accepted relocations	143,325 (97.91%)
Coverage-gate rejections	2,930 (2.00%)
Span-bound rejections	126 (0.086%)
Changed-file displayed views	67,530
Exact current-byte matches	67,530 (100%)
All view-turns	203,281
Shown / withheld / evicted	94.86% / 1.98% / 3.16%
Unsafe dirty-to-clean redisplays	37 transitions
Successful rereads remaining withheld	929 attempts

K Failure Localization and Prompt Timing

Across 8,446 scorer-accounted pairs in the 17-comparison replay set, 614 outcomes are baseline-only and 588 are LiveState-only. Table 14 locates the first material action divergence. Most paths split before any panel is visible in both directions (94.5% of baseline-only and 95.6% of LiveState-only outcomes), so early divergence cannot be assigned to stale file content. The seven repeated regressions whose first divergence follows a panel all receive current content covering the correct lines; their failures are incomplete semantic patches or command-format loops, not demonstrated stale views. This does not make the instruction prefix causal—hosted sampling alone often changes the first command—but it rules out corrupted panel content as a general explanation for the observed accuracy trade-off.

The evaluated instruction is 1,019 characters and precedes panel activation by a median of 3–6 turns, depending on model. Exact first commands differ frequently even across baseline replicates, so these results do not isolate an instruction effect. Coarse first-action distributions nevertheless shift beyond within-baseline variation for Fable, Haiku, and Gemini.

Step-limit accounting also argues against presenting isolated spiral examples as the aggregate effect. LiveState induces 78 limit failures but removes 197, for 119 fewer net spirals. Of the induced cases, 60 are command-format exhaustion (54 Fable and 6 Haiku), 9 are sandbox outages, and 9 are semantic or termination failures. We therefore treat provider-specific command sanitation and infrastructure retry as symmetric harness issues, not as evidence for the anchoring mechanism.

Table 14: First material divergence in asymmetric paired outcomes. One LiveState-only trajectory is not replayable.

Outcome	First action	Later, pre-panel	Post-panel	Total
Baseline-only	487	93	34	614
LiveState-only	493	68	26	587

Table 15: Existing-log token volume and cache telemetry. Each cell shows baseline → LiveState. A dash means that the provider field is not available, not that caching is absent.

Model	Tokens/traj.	Cache-read share	Cache-create share
Claude 4.7 Opus	248.8K → 190.1K	89.9% → 5.5%	5.2% → 89.2%
Claude 5 Fable	110.6K → 109.0K	86.6% → 30.3%	13.3% → 69.7%
Claude 4.6 Sonnet	430.6K → 333.4K	94.3% → 7.6%	5.7% → 92.3%
Claude 4.5 Haiku	1,005.6K → 522.0K	95.8% → 2.7%	3.0% → 95.2%
Gemini 3 Flash	271.7K → 275.1K	58.5% → 45.2%	–

L Serving Telemetry

Table 15 separates submitted token volume from provider-reported prefix-cache behavior. The dynamic trailing panel substantially reduces reported cache-read share for Claude. OpenAI logs do not expose usable cache fields in these runs, and the available spend snapshots are shared, missing, or discontinuous; we therefore omit monetary expenditure estimates.

Timestamp-derived latency is omitted because arms were not run in synchronized A/B blocks and the measurements conflate provider load with the intervention. The robust conclusion is narrower: logical token savings and provider serving requirements are different quantities, and the present system's changing suffix can trade fewer submitted tokens for less cache reuse.

M Future Directions

Cache-aware panels. A cache-aware LiveState could keep unchanged entries in stable blocks. Engine-level position correction, such as RoPE-aware remapping of cached keys, may also help when both a block and its causal prefix are unchanged; changed or context-dependent blocks must still be recomputed.

Adaptive panel selection. The current system renders every non-evicted entry and applies a least-recently-read policy only when the panel exceeds its character budget. A stronger context manager could decide before each call which files and windows to show, using the current task, recent edits, access history, or model uncertainty. Omitted entries could remain as recoverable stubs, preserving LiveState's explicit failure behavior while reducing the average panel size. This extension would require measuring both token reduction and the task-success effect of hiding otherwise valid context.

Separate compression of non-file history. LiveState separates verified file windows from the rest of the interaction history. This makes a hybrid design possible: keep selected current file windows verbatim in the panel while summarizing or pruning the remaining dialogue and tool observations. Such a design could obtain larger context reductions without asking a summarizer to rewrite source code, but it must still preserve task instructions, test failures, and other non-file evidence needed to solve the task.