# SAFETY IN NUMBERS: THE KV CACHE WAS NEVER THAT FRAGILE TO REPEATED PRUNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Long-running agents accumulate stale context from failed retrievals and superseded reasoning, occupying GPU memory and distorting attention. Removing it at the text layer forces a re-encode; at the KV-cache layer it is nearly free: excise rows, re-rotate survivor positions, continue. Whether that composite cache *behaves* like an honest re-prefill of the surviving tokens is untested, and production practice stays conservative. We test it directly, holding surviving content byte-identical and varying only operation count (Q1), window position (Q2), and eviction timing (Q3); relevance is oracle-given and the mechanism a reused RoPE rotation, so the operation itself is the only remaining variable.

The caution inverts: repeating the cheap operation makes the result *more* faithful, not less. Over 8 rounds at 92.9% pruning, composite-KV quality reaches 91.3% of a clean re-prefill and improves with round count, with no drift over 26+ real-trajectory rounds (Q1). Re-indexing costs only $+0.8\%$ and matters only when survivor positions exceed the model's trained window, confirmed by a pre-registered model swap (Q2). Timing is the genuine open variable (Q3): correctness slips only a little as junk lingers, a small, task-dependent cost that timely pruning keeps low. The worry that repeated pruning compounds error is unsupported; it is timing, not repetition, that costs.

## 1 INTRODUCTION

A deep-research or coding agent can run for dozens of tool calls. Its context fills with content that goes stale the moment it is produced: failed retrievals, abandoned tool calls, execution logs, reasoning the agent has already moved past. On the real search trajectories, this is not a minor tax: context windows reach a median of roughly 54K tokens, of which only $\approx$35% is useful evidence. The rest still sits inside every subsequent attention computation for as long as it lingers, distorting attention distributions and occupying scarce GPU memory (Liu et al., 2024a; Hong et al., 2025).

Removing that junk at the **text layer** means rewriting the transcript itself, so the model re-encodes precisely the surviving tokens: the result is by construction equivalent to a clean re-prefill of them. The field has explored this layer two ways. **Summarization** compresses what is discarded, at the risk of hallucinating under length pressure (Semenov & Dorofeev, 2026). **Deletion** instead drops it verbatim and rewrites nothing. The **Key-Value (KV) layer** is different: nothing is rewritten at all. Cached activations are spliced or evicted directly. There is no reprocessing, so the operation is cheap. Survivors keep their original position indices, now gapped wherever content was excised. A fresh re-prefill, by contrast, simply renumbers positions $0, \ldots, N-1$. But cheap is not the same as faithful to a clean re-prefill. Whether the resulting cache *behaves* like an honest re-prefill of the survivors is an assumption, not a given. The equivalence-to-re-prefill guarantee text-layer deletion enjoys does not automatically carry over.

Because that guarantee is unverified, production systems play it safe, for two distinct reasons. **(a)** An untested intuition: repeated, lossy cache-level edits might compound error, the way any iterative shortcut invites suspicion. **(b)** A real, measured cost: a naive KV-layer edit relocates survivor rows and disturbs prefix-cache reuse (the reuse of already-cached KV rows across requests), collapsing the measured prefix-cache hit rate to 0–3% once eviction renumbers positions (Li et al., 2026).

In practice, both reasons lead to the same habits: avoid the operation entirely and fall back to full re-prefill on every edit, or confine it to a narrow, heavily-tested corner. Examples include a single



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.

trailing seam (Xiao et al., 2024), budget-eviction rules that promise only acceptable loss under a budget rather than equivalence to re-prefill (Zhang et al., 2023; Li et al., 2024), and, increasingly, protocol-level session APIs now being standardized by the two dominant open-source inference engines on that same untested premise (vLLM Project, 2026). Neither reason has been tested against a third possibility: what actually governs the outcome may be dwell time, how long junk lingers before removal, not how often or how cheaply the cache is touched. At the text layer, by contrast, the field has moved fast. Compaction is now standard in agent harnesses, though how it is triggered is unsettled—a source-code taxonomy of 13 scaffolds finds seven distinct strategies (Rombaut, 2026), with adaptive, progressively-shedding compaction typical of shipped harnesses (Bui, 2026). Newer schemes go further, using the agent's own semantic structure—such as tool-call cursors or typed exploration episodes—to decide what to remove.

Practitioners have begun to distrust the lazy, resource-triggered default. Compaction at 50–60% capacity is publicly recommended over the standard 95% auto-trigger (Sikkema, 2026), proactive memory management is called "the future" of context engineering (Factory.ai, 2025), and practitioners describe the field as shifting from widening context windows to actively managing them (Zylos Research, 2026). But all prior evidence is anecdotal and confounded: compacting earlier also means summarizing a shorter span, producing more accurate summaries (Sikkema, 2026). No prior study separates the timing effect from the effect of what survives. The intuitive answer is to clear junk the moment it is identified. But that intuition alone does not say whether it is *safe* to act on cheaply and repeatedly at the KV level, or which of the two field worries actually governs the outcome.

We decompose the open question into three:

- **Q1.** Is KV-level context pruning safe to use aggressively and repeatedly? Does repetition compound error?
- **Q2.** Is position re-indexing after eviction necessary, and under what conditions?
- **Q3.** Does the *timing* of removal, meaning how long junk lingered before eviction, change downstream quality?

Answering any of these requires holding the final surviving content byte-identical while varying only count, window pressure, or eviction schedule. That is impossible under text-level summarization (Semenov & Dorofeev, 2026), and therefore never previously tested.

We operate directly on the KV cache of a single co-attended trajectory: prune whole semantically-bounded chunks online, re-index survivors to contiguous positions, and continue. There is no summarization, no re-encoding, and no stitching of independently-encoded blocks. We deliberately implement this with the simplest, already-validated mechanism: Rotary Position Embedding (RoPE) re-rotation. It is the same operation underlying StreamingLLM's trailing-seam extension (Xiao et al., 2024), now shipped in production form for policy-driven edits (Ma et al., 2026). What we contribute is the controlled measurement this trusted mechanism has never received: whether it holds up when invoked aggressively and repeatedly. Where position re-indexing is concerned, we test its conditionality with a pre-registered design: identical questions, two models whose trained context windows differ. Chunk relevance is oracle-given throughout, isolating *when and how* known junk is removed as the object under test. Selection, whether a model can correctly identify junk, is a separate question, already served by existing relevance scorers.

We test all three questions on both synthetic long-context benchmarks and real agent search trajectories, varying operation count and eviction timing while comparing re-indexed against frozen-position variants, always measured against a clean re-prefill oracle.

**Q1: repetition does not compound error.** Repeating the operation across rounds does not degrade quality: it improves, reaching 91.3% of clean re-prefill quality by round 8. It tracks closer to the oracle the more often junk is cleared, with no drift on real agent trajectories either. A frozen-position ablation confirms the driver is dwell time, not reduced window overflow. Reason (a), the compounding-error worry, is retired.

**Q2: re-indexing is nearly free, and necessary only conditionally.** The overhead is negligible, just $+0.8\%$ compute. Whether it helps at all depends entirely on the model: no benefit when survivor

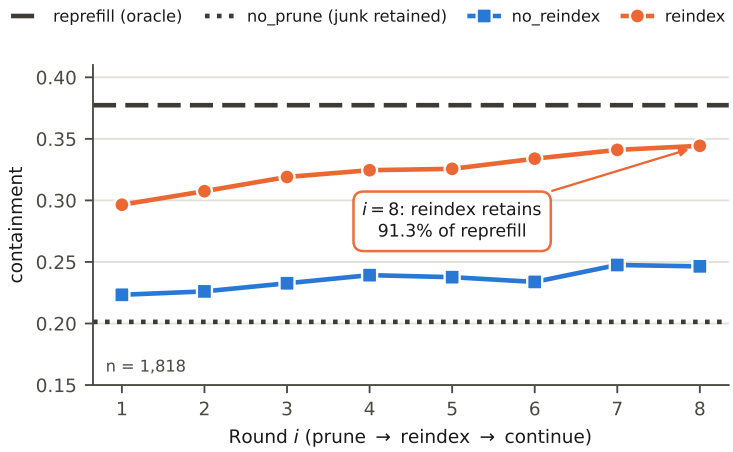


Figure 1: KV-level pruning is composable: repeating it does not compound error. `reindex` rises monotonically with round count to 91.3% of the `reprefill` oracle at $i$=8; `no_reindex`, which leaves survivor positions unfixed, rises overall too. Averaged across 1,818 LongBench-style samples.

positions already fit inside the trained context window, a decisive one when they would otherwise spill past it.

**Q3: timing is the genuine open variable, but its cost is small.** We hold the final survivor set fixed and vary only how long junk lingers before removal. Task correctness (`containment`) pays a small but real cost as that lingering grows: 2–3 points on evidence-synthesis (noise-sensitive) tasks, too small to detect on noise-robust ones. Byte-level fidelity (`byte-identity`) to the oracle behaves very differently: it drops sharply within the first few rounds and then flattens out, rather than continuing to decline the longer junk stays. The two costs differ in kind: a modest, genuine cost to correctness, and a far larger, early cost to exact reproducibility. The residual cost is governed by timing, not repetition, and early eviction keeps it small.

## 2 Related Work

**Context rewriting and pruning.** Long-context language models face two problems: irrelevant context degrades performance (Liu et al., 2024a; Hong et al., 2025), and retaining it raises inference cost. Prior work addresses both by rewriting the context (summarization (Semenov & Dorofeev, 2026; Xu et al., 2024)) or by removing less useful portions (pruning (Jiang et al., 2023; Kariyappa & Suh, 2026; Semenov & Dorofeev, 2026)). These approaches differ in what they sacrifice: summarization keeps coverage but replaces the original text, while deletion keeps surviving content exact but discards what is removed — a sharper risk for long-running agents, where discarded content may become useful again and its recoverability is a known failure mode (Kariyappa & Suh, 2026; Wang et al., 2026). They also share a cost: because the text itself changes, the affected portion must be reprocessed before the model can continue (Semenov & Dorofeev, 2026).

**KV cache compression.** A complementary line of work operates below the text layer on the KV cache itself, motivated by the memory and repeated access it demands during decoding. One branch manages this footprint while preserving the underlying sequence — paged allocation (Kwon et al., 2023), architectural head sharing (Ainslie et al., 2023), and low-bit quantization (Liu et al., 2024b). A different family performs content-aware eviction by importance, recency, or region under a memory budget (Zhang et al., 2023; Li et al., 2024; Xiao et al., 2024; Matam & Kim, 2026), optimizing a memory–quality trade-off rather than equivalence to a recomputed cache. More recent work keeps the cache's structural units — compression at semantic-chunk granularity (Liu et al., 2025), or a query-agnostic reconstructable subset compressed once at prefill (Kim et al., 2025).

**KV cache reuse.** KV cache reuse avoids recomputing the same context more than once. Exact prefix sharing, via prefix trees (Zheng et al., 2024) and batched shared-prefix attention (Juravsky et al., 2024), reuses a prefix identical to the input and so introduces no discrepancy with a freshly computed cache. More general schemes relax the prefix constraint — reusing cached chunks under weaker ordering (Jin et al., 2026) or making representations position-independent (Hu et al., 2025).

This leaves a gap: an independently encoded chunk lacks the cross-attention a full prefill would compute, which selective recomputation of the highest-deviation tokens partially restores (Yao et al., 2025; Yang et al., 2025). These systems make the cache a reusable representation, yet most still address reuse across requests rather than the modification of an already-running session whose states have been consumed by decoding.

**KV cache editing for long-running sessions.** This distinction matters for long-running agents, whose context changes repeatedly as tool outputs go stale and spans need replacing without restarting the session, and whose caches a recent line of work already prunes across turns (Li et al., 2026). Text-layer edits require recomputing the affected prefix; editing the cache directly raises a different question — whether the modified cache stays consistent with one computed from the edited context from scratch. Leyline addresses this directly, exposing a running session's cache as declarative edits resolved through position-aware transformations of the surviving states (Ma et al., 2026). Cache editing is therefore a different objective from reuse: not reusing a representation in another request, but transforming an existing session in place while preserving positional consistency. The open question is not whether a single edit is efficient, but how reliably such edits can be applied repeatedly over a session's lifetime. That question is now urgent at the protocol layer: vLLM and SGLang are independently standardizing session-aware KV retention and eviction APIs (vLLM Project, 2026; SGLang Project, 2026), yet these proposals specify *who decides* retention, not whether repeated splice-and-continue remains faithful to a clean re-prefill — the property this paper measures.

**Positioning of this work.** Our work sits at the intersection of context-level pruning and KV-level editing. Like the text-level methods above, we remove semantically meaningful spans rather than individual KV rows. Like cache reuse and editing, we apply changes directly to the KV cache rather than reconstructing the prefix after every edit. The distinction is repetition—we examine whether a sequence of cache-level edits stays consistent with the corresponding text-level edit followed by full recomputation, and how that consistency changes as edits accumulate. Pruning, reuse, and editing share the objective of retaining only useful context while avoiding unnecessary recomputation; we carry it to the long-running, repeatedly modified cache of an agentic session.

## 3 Design and Method

**Operation.** We study the following single-step operation applied to a running KV cache: (1) mark a contiguous, semantically-bounded chunk of tokens as irrelevant; (2) excise those rows from the cache; (3) re-rotate the rotary position embeddings (RoPE) of all surviving rows to fill the resulting gap, producing a cache whose position indices run contiguously from 0 to $N-1$; (4) continue prefilling or decoding from that state. This step can be chained: each application takes the KV state left by the previous step as input, applies another round of eviction and re-indexing, and continues. We call the resulting composite cache the product of $i$ rounds of `prune` $\rightarrow$ `reindex` $\rightarrow$ `continue`. The quantity under study is how faithfully this composite cache approximates the oracle baseline: re-prefilling the identical surviving tokens from scratch.

**Mechanism: position re-rotation (inherited, not claimed).** The re-rotation in step (3) applies the same closed-form RoPE $\delta$-shift that underlies StreamingLLM's trailing-seam extension (Xiao et al., 2024), generalizes it to multiple arbitrary eviction gaps per context, and is independently validated at production scale for policy-driven edits (Ma et al., 2026). We inherit this mechanism without modification and *do not claim it as a contribution*. Using the simplest, most-validated available implementation is deliberate: our contribution is a controlled measurement of operation safety and timing, not a new eviction mechanism, and using a contested or novel mechanism would confound measurement results with mechanism uncertainty.

**Scope: oracle-given relevance.** Chunk relevance is oracle-given in all experiments. We mark as irrelevant exactly those tokens that ground-truth annotation or the retrieval system itself identifies as non-answer-bearing. This isolates the questions under study, namely whether, how often, and when the KV-layer operation is safe, from the orthogonal question of whether a model can accurately identify irrelevant content. Selection is not in scope. Any relevance scorer from the existing literature can supply the oracle judgment, including attention-score heuristics (Zhang et al., 2023; Li et al., 2024), model-driven schedulers (Kariyappa & Suh, 2026), dependency-graph policies (Semenov &

Table 1: The three metrics. They are not interchangeable: `containment` is the task-correctness verdict, while `byte-identity` and `same_outcome` are diagnostic instruments that decompose where its cost comes from.

| Metric | Role | What it measures | Questions |
|---|---|---|---|
| `containment` | Primary | Gold answer appears as a substring of the output (task correctness) | Q1, Q2, Q3 |
| `byte-identity` | Diagnostic | Output matches `reprefill` byte-for-byte (mechanistic fidelity) | Q1 depth, Q3 |
| `same_outcome` | Diagnostic | Composite-KV and `reprefill` agree on success/failure | Q3 |

Dorofeev, 2026), or learned region priors (Matam & Kim, 2026); our findings about operation safety apply to all of them.

# 4 Evaluation

## 4.1 Setup

We structure the evaluation across three experimental settings, named for the data they draw on. The **LongBench-style experiment** (§4.2) answers Q1 on controlled synthetic data, using LongBench-style retrieval benchmarks with artificially interleaved filler documents to create composability pressure. The **BrowseComp-Plus replication** (§4.3, §4.4) replicates Q1 on real agent search trajectories and answers Q2 via a pre-registered model-swap falsification. The **AppWorld K-sweep** (§4.5) answers Q3 with an eviction-schedule sweep, holding the survivor set byte-identical.

**Conditions.** Every experiment compares four conditions over an identical surviving token set: **reprefill** is a fresh re-prefill of surviving tokens from scratch, the oracle-comparison baseline; **vanilla / no_prune** is the full unpruned context with no eviction; **no_reindex** is the composite KV cache with survivor positions left unfixed (original, potentially gapped or out-of-window); and **reindex** is the composite KV cache with survivor positions re-rotated to contiguous $[0 \dots N-1]$. Any quality gap between `reindex` and `reprefill` is the true cost of the KV-level shortcut; any gap between `reindex` and `no_reindex` is the value of position re-indexing.

## 4.2 Q1: Composability on Synthetic Data

**Setting.** This experiment assembles a composite KV cache by repeating `prune` $\rightarrow$ `reindex` $\rightarrow$ `continue` up to $T{=}8$ rounds on an expanded, filler-interleaved context that exceeds the model's native input window (e.g. Qwen3-8B 40K, Llama3.1-8B 128K) by a factor of 1.25–2.9$\times$. The pruning rate is aggressive: surviving tokens constitute only 7.15% of the original context (92.9% pruning). The question is whether composite-KV quality, measured against the `reprefill` oracle, degrades as round count $i$ grows.

**Results.** Table 2 and Figure 1 show the main result. At $i{=}8$ rounds, `reindex` retains **91.3%** of `reprefill`'s containment (0.344 vs. 0.377). Critically, quality does not degrade with round count. It *improves*: containment rises monotonically from 0.296 at $i{=}1$ to 0.344 at $i{=}8$, and `reindex` beats `no_reindex` with the gap widening rather than narrowing as rounds accumulate (comparison-level robustness detail in Appendix A.2).

**Significance.** The improvement from `reindex` at $i{=}1$ to $i{=}8$ is significant ($p{<}0.0001$), confirming that quality improves with round count rather than degrading. The residual gap between `reindex` at $i{=}8$ and `reprefill` is also significant ($p{=}0.0001$), so the 8.7% shortfall behind the 91.3% retention figure is detectable rather than noise (Appendix A.2). On real retrieval trajectories the corresponding gap is not distinguishable from zero on either model (§4.3), a descriptive contrast given the settings differ in dataset, pressure, and pruning rate.

Table 2: LongBench-style containment (ALL micro, 1,818 samples). `reindex` at 8 rounds retains 91.3% of the clean-recompute oracle at 92.9% pruning. $i$ denotes round count; $i$=0 is `no_prune`, and `reprefill` is the clean-recompute oracle, not a round count.

| Condition | $i$ | Containment | vs. reprefill |
|---|---|---|---|
| `reprefill` | – | 0.377 | 1.000 |
| `no_prune` | 0 | 0.201 | 0.534 |
| `no_reindex` | 8 | 0.246 | 0.653 |
| `reindex` | 8 | **0.344** | **0.913** |

Table 3: BrowseComp-Plus containment (`gold_recall >0` subset, `qwen3-embed-8b/gpt5`). Pruning conditions substantially beat `vanilla` on both models, and `pruning_reindex` is statistically tied with the `reprefill` oracle on both (Llama $p$=0.628; Qwen $p$=0.502). The only significant shortfall against `reprefill` is `pruning_noreindex` on Qwen-40K ($-0.037$, $p<0.001$), the out-of-window effect isolated in §4.4. Bold marks the best value in each column.

| Condition | **Llama-128K** ($n$=685) | **Qwen-40K** ($n$=669) |
|---|---|---|
| `vanilla` | 0.669 | 0.717 |
| `pruning_noreindex` | 0.717 | 0.747 |
| `pruning_reindex` | 0.721 | 0.779 |
| `reprefill` | **0.726** | **0.785** |

**Ruling out window overflow as the explanation.** A natural alternative explanation is that more rounds clear junk sooner, reducing window overflow. The `no_reindex` condition isolates this: positions are never updated there, so it cannot benefit from reduced overflow, yet its containment still rises with round count ($0.223 \rightarrow 0.246$ at $i$=8). The driver is therefore reduced *dwell time* rather than window overflow, which §4.5 isolates directly.

### 4.3 Q1: Composability on Real Retrieval Trajectories

**Setting.** This replication replays BrowseComp-Plus search trajectories (`qwen3-embed-8b/gpt5` retriever: an 8B dense embedding retriever with a GPT-5 search agent) through both Llama-3.1-8B-Instruct (128K window) and Qwen3-8B (40K window), yielding 791 complete Llama samples and 767 complete Qwen samples. Contexts reach a median of ≈54K tokens, of which ≈65% is junk from dead-end searches; the median surviving fraction after pruning is 0.35. Each retrieved search round is pruned immediately after arrival and the model generates only once at the end, so any quality difference across conditions is attributable solely to the KV-level operation.

**Task-correctness results (Table 3).** On samples where the oracle successfully retrieved an answer-bearing document (`gold_recall >0`; Llama $n$=685, Qwen $n$=669), both pruning conditions substantially improve over `vanilla` on both models, and the gain is larger on the smaller window (`reindex` − `vanilla` = $+0.052$ on Llama, $+0.062$ on Qwen): junk occupies a larger share of a 40K window, so clearing it returns more.

**`reindex` is statistically tied with the `reprefill` oracle on both models.** On the full `gold_recall >0` population, `pruning_reindex` is statistically indistinguishable from `reprefill` on both models (Llama $-0.004$, $p$=0.628; Qwen $-0.006$, $p$=0.502; Figure 2). The one condition that does fall significantly short of the oracle is `pruning_noreindex` on Qwen ($-0.037$, $p<0.001$): precisely the out-of-window effect Q2 isolates (§4.4). The KV-level shortcut costs nothing detectable on task correctness once positions are re-indexed, and the only visible penalty is mechanical, incurred by leaving positions unfixed in a window too small to hold them (test details, CIs, and the matching Llama-`noreindex` comparison are in Appendix A.2).

**No drift through 26+ consecutive rounds.** To confirm that Q1's composability result holds at the depths real agents reach, we isolate contamination-free samples (defined in §A.4) and measure

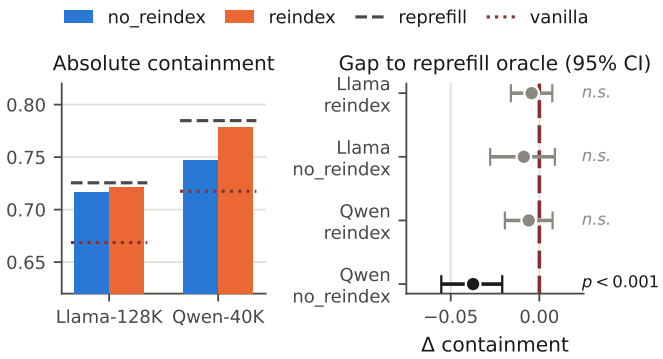


Figure 2: BrowseComp-Plus replication (Q1): gap to the re-prefill oracle. `reindex` ties `reprefill` on both models; the only significant shortfall is `noreindex` on Qwen-40K (Q2's out-of-window effect).

Table 4: Pre-registered model-swap falsification (BrowseComp-Plus, same 669 questions, only model differs). The null result on Llama-128K and the decisive result on Qwen-40K were both predicted in advance from the window-overflow mechanism. $\Delta$ is `noreindex` − `reindex`.

| Model | no_rei. | reindex | $\Delta$ | McNemar $p$ | 95% CI |
|---|---|---|---|---|---|
| Llama-128K | 0.731 | 0.735 | −0.004 | 0.749 | [−0.022, +0.013] |
| Qwen-40K | 0.747 | **0.779** | −0.031 | $<$ **0.001** | [−0.048, −0.016] |

byte-level fidelity to `reprefill` as a function of search depth (bucket values in Table 10). Fidelity shows no *monotonic* decline with depth on either model: the deepest bucket sits *above* the shallowest on both. A compounding-error mechanism would show declining fidelity as rounds accumulate; neither model does, confirming the LongBench-style composability result (§4.2) at real-world depths.

### 4.4 Q2: Reindexing Conditionality via a Pre-Registered Model-Swap

**Pre-registered design.** We predicted before running the BrowseComp-Plus replication that reindexing would have no measurable quality effect on Llama-3.1-8B-Instruct (128K trained window) but a decisive positive effect on Qwen3-8B (40K trained window), for a mechanistic reason: without reindexing, survivor positions land at their original indices, placing the question at a median position of ≈55K, which is entirely within Llama's 128K window but past Qwen's 40K window for the majority of questions. We test this with a paired design: the same 669-question subset, only the model swapped. This 669-question subset is the intersection of both models' `gold_recall>0` sets, hence a proper subset of the $n$=685 Llama population in Table 3 (population accounting in §A.5).

**Results.** Both predictions are confirmed (Table 4). On **Llama-128K**, `noreindex` − `reindex` $= -0.004$, McNemar $p = 0.749$, 95% CI $= [-0.022, +0.013]$: the CI straddles zero, and the null hypothesis of no effect is not rejected. On **Qwen-40K**, `noreindex` − `reindex` $= -0.031$, McNemar $p < 0.001$, 95% CI $= [-0.048, -0.016]$: the CI excludes zero, and the effect is statistically decisive. In direct answer to Q2, position re-indexing is necessary only when survivor positions would otherwise fall outside the model's trained context window: it buys nothing in-window (Llama-128K) and is decisive out-of-window (Qwen-40K).

**Mechanism.** On Qwen (40K), 503 of 767 questions (65%) fall outside the trained window without reindexing—RoPE is extrapolating past its trained range—and reindexing compresses the median position from ≈55K to ≈14K, leaving only 40 (5%) out-of-window. On Llama (128K), no question falls outside the window under either condition, so the null effect is mechanically expected.

### 4.5 Q3: Dwell-Time K-Sweep

**Setting.** This sweep reuses the same three AppWorld trajectories (`react_llama3`, `plan_exec`, `full_code_refl`; two models) listed in the experimental Setup (Table 5), whose $K$=0 column

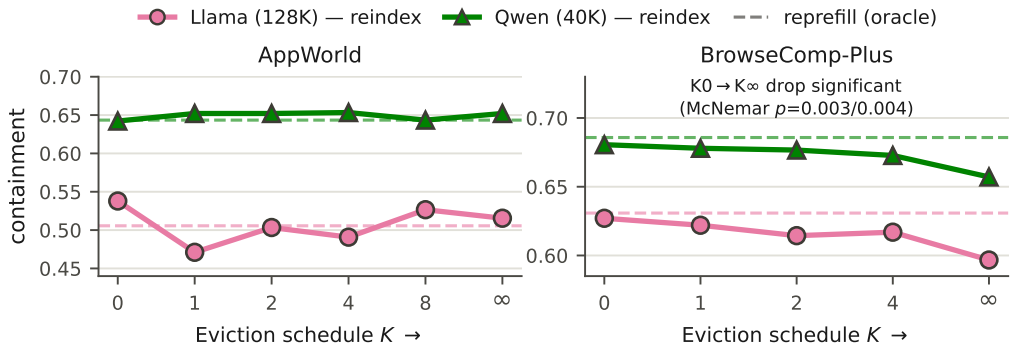


Figure 3: Containment vs. eviction schedule $K$, by model. **Left:** AppWorld ($n$=90/model) and **Right:** BrowseComp-Plus ($n\approx$780): the same small, real cost of delayed eviction, significant with enough statistical power (right), invisible without it (left).

is the immediate-eviction condition, but now varies only the eviction schedule $K$: junk is evicted every $K+1$ rounds ($K=0$ is immediate eviction—junk never survives into a later round—and $K=\infty$ prunes only once, at the end, so every survivor is prefilled with all prior junk still present). A final forced prune at the end of every trajectory guarantees the surviving token set fed to generation is byte-identical across all $K$, so any quality difference is attributable solely to how long junk lingered, not to what ultimately survived. We sweep $K \in \{0, 1, 2, 4, 8, \infty\}$, with the intermediate points resolving the dose-response shape between the near-immediate and long-dwell extremes.

**Results: containment shows a small, real, task-dependent cost.** `containment`, the paper's correctness metric (Table 1), is checked independently against ground truth rather than against the system's own `reprefill` run, and it is not flat across $K$. On BrowseComp-Plus real search trajectories ($K \in \{0, 1, 2, 4, \infty\}$; $n$=791 Llama / 767 Qwen, the full sample rather than the `gold_recall>0` subset of Table 3), delaying eviction from immediate ($K$=0) to maximally deferred ($K$=$\infty$) drops containment 2–3 points on both models (Llama $0.627 \rightarrow 0.597$, Qwen $0.681 \rightarrow 0.657$), and both drops are statistically significant (continuity-corrected McNemar: Llama $p$=0.0025, Qwen $p$=0.0036; Figure 3, right panel). The AppWorld K-sweep, aggregated to the `answer`-task subset across its three trajectories ($n$=90/model), shows the same comparison move in the same direction on Llama ($0.538 \rightarrow 0.515$, a comparable 2.3-point drop) but not on Qwen ($0.642 \rightarrow 0.652$), and neither reaches significance (McNemar $p$=0.7728 Llama, $p$=1.0000 Qwen; Figure 3, left panel) — AppWorld's per-model $n$=90 is underpowered relative to BrowseComp-Plus's $n\approx$780, and several intermediate $K$ nominally exceed `reprefill` within that noise (Appendix Table 6; full values in §A.6).

The two datasets diverge primarily because the tasks differ in how robust they are to noise: BrowseComp-Plus requires synthesizing evidence across many search rounds, where correctness depends on precise attention allocation and is vulnerable to contamination from junk that was once present and later deleted, while AppWorld's `answer` tasks are mostly single-fact lookups where background noise matters less; detectability is further constrained by the two datasets' differing trajectory lengths and sample sizes (Appendix A.6).

**Diagnostic instruments: both mask the cost.** `byte-identity` costs far more than `containment` does, and `same_outcome` is too coarse to register the cost at all, so byte-level mismatch is a poor proxy for whether a task was answered wrong: most of what drives it is contamination reshaping wording, not changing the answer. Neither is Q3's correctness verdict (that is `containment` above); the full decomposition is in Appendix A.7 (Figure 5).

## 5 LIMITATIONS

Chunk relevance is oracle-given throughout (ground-truth annotation or retrieval labels), so our findings describe the operation's safety under a correct selection policy, not end-to-end quality with a deployed selector; selection errors would compound with the contamination effect in §A.9. Model coverage is three 8–9B dense checkpoints (two of them for the BrowseComp-Plus and AppWorld experiments), plus a limited experiment on a larger-scale checkpoint (a single Mixture-of-Experts model, Appendix A.8); instruction-tuned versus base models and other attention architectures remain uncovered, and the re-indexing threshold is window-size-dependent, hence model-specific. Every experiment also prunes a fixed, pre-recorded trajectory, so we do not test whether an early pruning-induced error can be amplified through the model's own live decision loop—the deployment case that motivates this work, and the most consequential question this paper leaves open.

## 6 CONCLUSION

We asked whether the KV-layer shortcut, which excises cached rows, re-indexes survivor positions, and continues, is safe to apply aggressively and repeatedly to a running agent context. The mechanism (position re-rotation) was already validated; what was missing was a controlled measurement of its safety across operation count, window position, and eviction schedule.

Three findings emerge. **Q1:** Repetition does not compound error. At 92.9% pruning over 8 rounds, composite-KV quality reaches 91.3% of a clean re-prefill oracle and improves monotonically with round count. The improvement persists even when positions are held fixed throughout, ruling out window-overflow reduction and pointing to dwell time as the quality driver. No drift appears through 26+ consecutive rounds on real retrieval trajectories. **Q2:** Position re-indexing is nearly free but strictly conditional. It has no measurable effect when survivor positions fit within the trained context window (Llama-128K, $p$=0.749) and a decisive positive effect when they do not (Qwen-40K, $p$<0.001). The condition is structural, determined by trained window size and typical trajectory depth, and does not require tuning. **Q3:** Dwell time's effect is metric-dependent. On task correctness (`containment`), delaying eviction carries a small but real, task-dependent cost: 2–3 points, and statistically significant, on BrowseComp-Plus's larger real-trajectory replication, present in the same direction but not distinguishable from zero on AppWorld's smaller sample. On byte-level fidelity (`byte-identity`), the cost is far larger and threshold-shaped: only immediate eviction preserves the mechanistic advantage, and it attenuates toward the unindexed baseline within a few rounds once junk co-resides with survivors.

The practical upshot is narrower than the folk intuition suggests: repeated KV-layer eviction does not progressively degrade quality, but cache-invalidation cost is real and remains the binding constraint on how aggressively to evict. Within that constraint, the guidance is: evict early whenever byte-level reproducibility matters or the task synthesizes evidence across many rounds, where delay's correctness cost is real and measurable; simpler single-fact tasks can use whatever schedule is cheapest; re-index only when positions would otherwise leave the trained window.

Two bounds remain. All experiments use oracle-given relevance, so this paper does not answer *which* tokens to evict—selection policy is outside scope—and it does not measure serving-layer throughput or latency, so reason (b)'s cache-invalidation cost must be profiled per deployment. For systems that cannot tolerate any divergence from a clean-state baseline, text-layer deletion remains the correct choice; KV-layer eviction is for systems that accept the small, task-dependent timing cost reported above.

## REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.298.

Nghi D. Q. Bui. Building effective AI coding agents for the terminal: Scaffolding, harness, context engineering, and lessons learned, 2026. URL `https://arxiv.org/abs/2603.05344`.

Bradley Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1): 1–26, 1979. doi: 10.1214/aos/1176344552.

Factory.ai. Compressing context. Blog post, https://factory.ai/news/compressing-context, July 2025. Accessed 2026-08-29.

Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts LLM performance. Chroma Technical Report, July 2025. URL https://trychroma.com/research/context-rot. Accessed 2026-08-29.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24391–24402. PMLR, 2025. URL https://proceedings.mlr.press/v267/hu25j.html.

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*, pp. 13358–13376, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.825.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. RAGCache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 44(1):2:1–2:27, 2026. doi: 10.1145/3768628.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-throughput LLM inference with shared prefixes, 2024. URL https://arxiv.org/abs/2402.05099.

Sanjay Kariyappa and G. Edward Suh. SideQuest: Model-driven KV cache management for long-horizon agentic reasoning, 2026. URL https://arxiv.org/abs/2602.22603.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic KV cache compression with context reconstruction. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2505.23416.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165.

Junjie Li, Jiong Lou, and Jie Li. IntentKV: Cross-turn intent-aware KV cache pruning for agent inference, 2026. URL https://arxiv.org/abs/2606.09916.

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*, volume 37, 2024. URL https://arxiv.org/abs/2404.14469.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a. doi: 10.1162/tacl_a_00638.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. ChunkKV: Semantic-preserving KV cache compression for efficient long-context LLM inference. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2502.00299.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 2024b. URL https://proceedings.mlr.press/v235/liu24bz.html.

Bole Ma, Jan Eitzinger, and Harald Koestler. Leyline: KV cache directives for agentic inference, 2026. URL https://arxiv.org/abs/2606.01065.

Venkatesha Matam and Keon Kim. MemDecay: Region-aware KV cache eviction for efficient LLM agent inference, 2026. URL https://arxiv.org/abs/2607.10582.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. doi: 10.1007/BF02295996.

Benjamin Rombaut. Inside the scaffold: A source-code taxonomy of coding agent architectures, 2026. URL https://arxiv.org/abs/2604.03515.

Andrew Semenov and Svyatoslav Dorofeev. Beyond compaction: Structured context eviction for long-horizon agents, 2026. URL https://arxiv.org/abs/2606.11213.

SGLang Project. [RFC] agent-aware KV cache phase 1 for agentic workloads. GitHub RFC, sgl-project/sglang #24656, https://github.com/sgl-project/sglang/issues/24656, 2026. Accessed 2026-09-23.

Albert Sikkema. Why I shrunk Claude code's context window back to 200k. Blog post, https://www.albertsikkema.com/ai/development/tools/2026/04/23/smaller-context-window-better-claude-code.html, April 2026. Accessed 2026-08-29.

vLLM Project. Programmable KV cache: Composable policies for agentic serving. GitHub RFC, vllm-project/vllm #57103, https://github.com/vllm-project/vllm/issues/57103, 2026. Accessed 2026-09-23.

Yifei Wang, Ziteng Wang, Yuling Shi, Silin Chen, Xinrui Wang, Yueqi Wang, Beijun Shen, Linjing Li, Xiaodong Gu, Julian McAuley, and Daniel Dajun Zeng. Context compression for LLM agents: A survey of methods, failure modes, and evaluation. Preprints.org preprint, May 2026. URL https://doi.org/10.20944/preprints202605.2065.v1. Accessed 2026-08-29.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*. OpenReview.net, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*. OpenReview.net, 2024. URL https://openreview.net/forum?id=mlJLVigNHp.

Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. Kvshare: An LLM service system with efficient and effective multi-tenant KV cache reuse, 2025. URL https://arxiv.org/abs/2503.16525.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109. ACM, 2025. doi: 10.1145/3689031.3696098.

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*, volume 36, 2023. URL https://arxiv.org/abs/2306.14048.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2312.07104.

Table 5: Experimental setup across settings. In the LongBench setting the `@80K`/`@160K` suffix is the constructed sample length, which exceeds the models' native input windows (Qwen3-8B 40K; Llama3.1-8B 128K) to simulate trajectory depth running into the positional wall.

| Setting (answers) | Dataset | Models | n | Metric |
|---|---|---|---|---|
| Q1 (§4.2) | LongBench-style (2wikimqa, hotpotqa, multifieldqa, musique, qasper) | Qwen3-8B@80K; Yi-1.5-9B@80K; Llama3.1-8B@160K | 1,818 | `cont.` |
| Q1, Q2 (§4.3, §4.4) | BrowseComp-Plus qwen3-embed-8b/gpt5 | Llama-3.1-8B (128K); Qwen3-8B (40K) | 791 / 767 | `cont.`, `byte-id.` |
| Q3 (§4.5) | AppWorld (react_llama3, plan_exec, full_code_refl); BrowseComp-Plus | Llama-3.1-8B (128K); Qwen3-8B (40K) | 3 traj. × 2 × 6K; 791 / 767 | `cont.`, `byte-id.`, `same_out.` |

Zylos Research. AI agent context compression: Strategies for long-running sessions. Blog post, https://zylos.ai/research/2026-02-28-ai-agent-context-compression-strategies/, February 2026. Accessed 2026-08-29.

# A Appendix

## A.1 Experimental setup details

**Eligibility (LongBench-style experiment).** A sample is included only if the number of document chunks $\geq T+1$, the clean survivor baseline fits within the model's context window ($|B| \leq M$), and the expanded junk-containing context exceeds the window ($|X| \geq M$), ensuring that every sample actually creates composability pressure.

**Checkpoint reuse for composability measurement.** Measuring composability over $T$ rounds naïvely requires re-prefilling the full expanded context $T$ times. We avoid this by prefilling the expanded context once and cloning the raw KV state after each original content chunk to CPU-resident checkpoints. Each round $i$ reloads the appropriate checkpoint to GPU and runs only $i-1$ continuation steps, with no additional full prefill. The operation's re-indexing step adds $+0.8\%$ to per-round wall time; this design choice affects experimental efficiency but not the correctness of any measurement.

## A.2 Statistical testing for Q1

**LongBench-style (synthetic) significance and robustness.** The two significance claims in §4.2 use paired comparisons on the pooled $n$=1,818 samples carrying the complete 1+1+8+8 record set across the three configurations and five datasets. We use the continuity-corrected McNemar test (McNemar, 1947) and a paired bootstrap with a fixed random seed and 5,000 resamples (Efron, 1979). The comparisons are pooled because the estimand is the aggregate composability effect across the pre-defined Q1 population, rather than 15 separate per-configuration×dataset effects; configuration- and dataset-level values remain available in the underlying results for heterogeneity checks.

For `reindex` at $i$=1 versus $i$=8, the McNemar test gives $p<0.0001$ with a 95% CI of $[+0.031, +0.064]$. For `reindex` at $i$=8 versus `reprefill`, it gives $p$=0.0001 with a 95% CI of $[+0.017, +0.050]$. These tests establish the reported directional judgments; they do not imply that the pooled effect is uniform across every configuration or dataset. On the primary metric, `reindex` > `no_reindex` holds in 115 of 120 comparisons (3 model×budget configurations ×

5 datasets × 8 iterations), with 4 ties and a single reversal of 0.006 magnitude, a difference at noise scale rather than a systematic exception. This advantage widens with rounds overall, not narrows (+0.073 at $i$=1 to +0.098 at $i$=8); the widening is not step-wise monotonic, since two rounds show a small pullback, but the endpoint trend holds.

**BrowseComp-Plus (real trajectories) significance.** We test each pruning condition in §4.3 against `reprefill` with the same paired McNemar test and bootstrap, on the full `gold_recall>0` population. `pruning_reindex` is statistically indistinguishable from `reprefill` on both models (Llama $-0.004$, $p$=0.628, CI $[-0.016, +0.007]$; Qwen $-0.006$, $p$=0.502, CI $[-0.019, +0.007]$), with the CIs bounding any residual cost below 2 pp in either direction. `pruning_noreindex` is likewise indistinguishable from `reprefill` on Llama ($-0.009$, $p$=0.440, CI $[-0.028, +0.009]$), where survivor positions stay inside the trained window, but falls significantly short on Qwen ($-0.037$, $p<0.001$, CI $[-0.055, -0.021]$), where they do not. The pattern is therefore symmetric across models: the KV-level shortcut costs nothing detectable on task correctness once positions are re-indexed, and the only visible penalty is mechanical, incurred by leaving positions unfixed in a window too small to hold them.

### A.3 On the f1 metric

We report `containment` as the primary metric throughout. `f1` shows a length-bias artifact in the LongBench-style experiment: `reindex` achieves higher f1 than `reprefill` (0.220 vs. 0.206), a paradoxical result traceable to `reprefill`'s longer, more verbose outputs being penalized by f1's precision component. All primary claims in this paper use `containment` or the fidelity metrics defined in Table 1.

This appendix reports the AppWorld K-sweep's `containment` numbers at full per-trajectory resolution, the level at which the aggregated $n$=90-per-model result in §4.5 (Figure 3, left panel; Table 6) is actually computed, as a robustness check on that aggregate. Figure 4 disaggregates it by the three underlying trajectories (`react_llama3`, `plan_exec`, `full_code_refl`; $n$≈27–33 per trajectory×model pair). Individual pairs move in both directions between $K$=0 and $K$=∞: `plan_exec` declines on both models while `react_llama3`-Qwen and `full_code_refl`-Qwen do not. This heterogeneity is consistent with a small, real effect that is noisy at this sample size and only resolves into a significant signal at BrowseComp-Plus's much larger $n$, not with the effect being uniformly present or uniformly absent at the trajectory level.

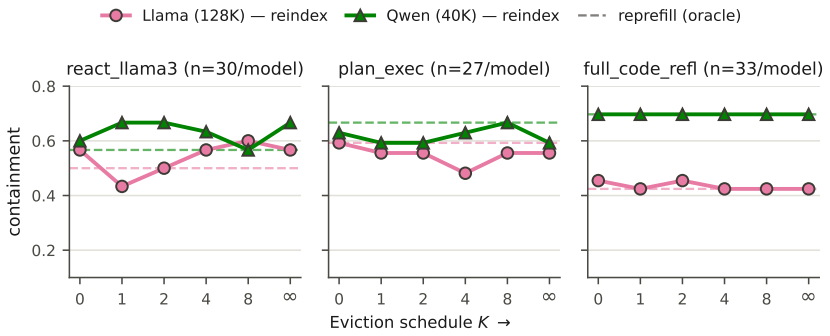


Figure 4: AppWorld K-sweep, `containment` at full per-trajectory resolution (3 trajectories × 2 models, $n$≈27–33 per pair). A fine-grained robustness check on the aggregated $n$=90-per-model result reported in Table 6: individual trajectories are noisier, as expected at this sample size, but reveal no pattern the aggregate conceals.

### A.4 Contamination-free samples as an exact fidelity instrument

For the depth-stability analysis in §4.3, we isolate contamination-free samples, defined as `contaminated_tokens = 0`: no search round had junk documents sitting causally *before* the

evidence document within that same round's prefill order. This is a same-round adjacency condition, not a claim that these samples were never pruned: whole dead-end rounds (no evidence at all) are still evicted, but since eviction is immediate ($K$=0) in this setting, that junk is gone from the cache before any later round's survivor tokens are even prefilled, so it cannot causally influence their keys/values either. On these samples, no surviving token's own attention computation was ever exposed to junk, so `pruning_reindex` is mechanically identical to `reprefill`: the re-rotation applies a zero shift ($\Delta p = 0$, $\cos 0 = 1$, $\sin 0 = 0$), introducing no approximation. Byte-identity is therefore an exact fidelity instrument in this setting.

### A.5 Population accounting across the two tables

The Llama `reindex`/`no_reindex` numbers in Table 4 (0.735/0.731, paired 669) differ slightly from Table 3 (0.721/0.717, full 685): both are correct, computed on deliberately different populations for different purposes. Table 4 is the pre-registered same-669-question paired comparison, while Table 3 reports the full $n$=685/669 `gold_recall >0` populations. The Qwen rows coincide across the two tables because Qwen's own `gold_recall >0` population *is* the 669-question intersection.

### A.6 AppWorld and BrowseComp-Plus K-sweep containment

The two aggregate K-sweeps reported in the body (Figure 3) are reproduced here at full K-resolution. AppWorld (Table 6) shows a same-direction drop on Llama but neither model reaches significance at this sample size; BrowseComp-Plus (Table 7) shows a statistically significant 2–3-point drop on both.

**Why the two datasets diverge.** Three factors are responsible. Primarily, the tasks differ in how robust they are to noise: BrowseComp-Plus requires synthesizing evidence across many search rounds (median 21, up to 47) where only ≈35% of the context carries useful signal, so correctness depends on precise attention allocation and is more vulnerable to contamination from junk that was once present and later deleted, while AppWorld's `answer` tasks are mostly single-fact lookups pinned to one app or record, where the answer usually lives in one place and background noise matters less. Secondarily, the two datasets expose survivors to different amounts of accumulated junk even at matched $K$: BrowseComp-Plus trajectories run a median of 21+ rounds, so $K$=$\infty$ means every survivor was prefilled alongside all of that junk, whereas AppWorld trajectories are shorter (median 12–17 rounds), so the same maximal-dwell condition carries substantially less absolute exposure. Also secondarily, the sample sizes themselves differ enough to matter statistically.

Table 6: AppWorld K-sweep, `containment` by model, aggregated over 3 trajectories (`answer`-task subset, $n$=90/model). K0→K$\infty$ (continuity-corrected McNemar): Llama $p$=0.7728, Qwen $p$=1.0000, neither significant at this sample size.

| Model | K0 | K1 | K2 | K4 | K8 | K$\infty$ | reprefill |
|---|---|---|---|---|---|---|---|
| Llama-128K | 0.538 | 0.471 | 0.503 | 0.491 | 0.527 | 0.515 | 0.506 |
| Qwen-40K | 0.642 | 0.652 | 0.652 | 0.653 | 0.643 | 0.652 | 0.643 |

Table 7: BrowseComp-Plus K-sweep, `containment` by model ($n$=791 Llama / 767 Qwen). K0→K$\infty$ (continuity-corrected McNemar): Llama $p$=0.0025, Qwen $p$=0.0036, both significant. This is the primary correctness result of the K-sweep: a small but real cost from delaying eviction, measured at a sample size large enough to detect it.

| Model | K0 | K1 | K2 | K4 | K$\infty$ | reprefill |
|---|---|---|---|---|---|---|
| Llama-128K | 0.627 | 0.622 | 0.614 | 0.617 | 0.597 | 0.631 |
| Qwen-40K | 0.681 | 0.678 | 0.677 | 0.673 | 0.657 | 0.686 |

### A.7 DIAGNOSTIC INSTRUMENTS FOR THE K-SWEEP

**`same_outcome` is a plateau, too coarse to see the cost.** `same_outcome`—whether the composite-KV output and `reprefill` agree on task success or failure, regardless of exact wording—stays within 0.92–1.00 across every $K$ and every trajectory×model pair on AppWorld (Table 8), with no trend as $K$ grows; even at $K{=}\infty$, where every surviving token was prefilled in the presence of *all* prior junk, agreement with a clean re-prefill is unchanged within noise. The same holds on BrowseComp-Plus, where `same_outcome` declines only gradually, from 0.97–0.98 at $K{=}0$ to 0.93–0.95 at $K{=}\infty$. This is consistent with, not contrary to, the containment cost above: a 2–3-point absolute drop in whether the gold answer is present is well within the resolution this coarser, success/failure-only instrument can register, so its flatness reflects a lack of sensitivity, not a lack of cost.

**`byte-identity` is a threshold, not a slope.** `byte-identity`—whether the output matches `reprefill` character-for-character—collapses as soon as junk is allowed to co-reside with survivors for even one round, then plateaus at a floor shared with `no_reindex` and `vanilla`. On AppWorld, the number of rounds needed to reach that floor is trajectory-dependent: `react_llama3` and `plan_exec` lose 78–100% of their $K{=}0$ byte-identity in the single step $K{=}0{\to}1$ and are indistinguishable from noise afterward; `full_code_refl`, longer, more deterministic code-completion outputs with a much higher $K{=}0$ starting point (0.71–0.72), declines in three resolved steps, $K{=}0{\to}1{\to}2$, reaching its floor only at $K{=}2$. On BrowseComp-Plus the same collapse is sharper still: byte-identity falls from 0.21–0.24 at $K{=}0$ to 0.01–0.02 by $K{=}1$, already at floor after a single round of dwell. Continuing either sweep past $K{=}4$ mainly confirms the floor rather than revealing new behavior: once reached, it does not continue to erode. This threshold collapse is far larger in magnitude than the 2–3-point containment cost above, tens of points against a handful, which is why byte-level mismatch is a poor proxy for whether a task was actually answered wrong: most of what drives it is contamination reshaping exact wording, not contamination changing the answer.

Table 8: Diagnostic instruments for the AppWorld K-sweep (3 trajectories × 2 models): `byte-identity` vs. `reprefill`, averaged within each trajectory group, and `same_outcome` as the range across all six trajectory×model pairs. These decompose *why* the containment cost in Table 6 is hard to see: byte-identity collapses far more sharply (a threshold reached within 1–2 rounds) and `same_outcome` is too coarse to move at all. Neither is Q3's correctness verdict; see Table 6 and Table 7 for that.

| **Metric (avg. byte-id.)** | **K0** | **K1** | **K2** | **K4** | **K8** | **K$\infty$** |
|---|---|---|---|---|---|---|
| Fast-floor (`react_llama3`, `plan_exec`) | 0.16 | 0.02 | 0.00 | 0.01 | 0.00 | 0.00 |
| Slow-floor (`full_code_refl`) | 0.72 | 0.42 | 0.32 | 0.32 | 0.27 | 0.27 |
| `same_outcome`, all 6 pairs | | 0.92–1.00, no trend with $K$ | | | | |

### A.8 ROBUSTNESS TO A SPARSE-FFN (MIXTURE-OF-EXPERTS) ARCHITECTURE

The dwell-time sweep in §4.5 uses two dense checkpoints. To check that its conclusions do not depend on dense attention, we repeat it on `Qwen3-30B-A3B` (128 experts, top-8 routing per token), whose attention path—GQA with 1-D RoPE—is identical to the dense `Qwen3-8B` checkpoint used elsewhere; only the feed-forward layer is sparse, so `reindex` applies unmodified and any difference is attributable to routing. We ran the full sweep ($K \in \{0, 1, 2, 4, 8, \infty\}$) on `react_llama3` ($n{=}155$; $n{=}30$ `answer`-scope) and the $K{=}0$ point only on the other two trajectories, all in `fp16`.

**`same_outcome` and `containment` replicate.** Both hold flat across $K$ on the MoE checkpoint, matching the dense range on the same trajectory, and the $K{=}0 \to K{=}\infty$ containment decline stays non-significant on both checkpoints (Table 9, Figure 6). Pruning remains free of measurable cost on task correctness with the FFN sparse.

**`byte-identity` floors instead of collapsing.** It sits at 0.00 for every $K$, including immediate eviction, rather than starting high and collapsing after one round of dwell as on the dense checkpoint (Table 9). A likely explanation is that top-8-of-128 routing is a discrete decision that even a small

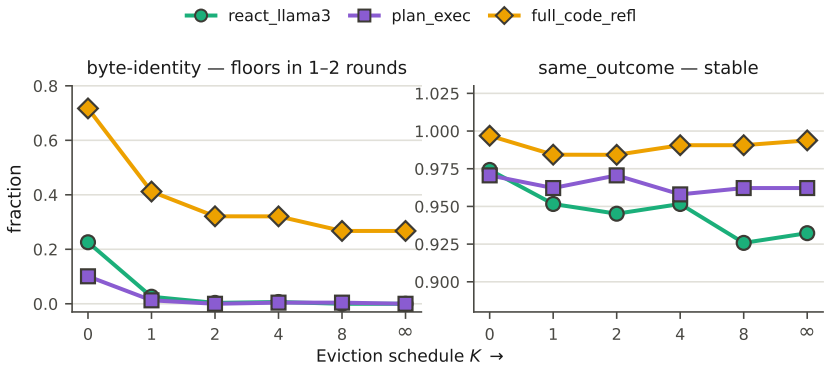


Figure 5: Diagnostic instruments vs. eviction schedule $K$: `byte-identity` (threshold collapse within 1–2 rounds, both AppWorld and BrowseComp-Plus) and `same_outcome` (plateau at 0.92–1.00, both datasets). Neither metric is sensitive enough to register the 2–3-point containment cost shown in Figure 3; they are shown here to decompose where that cost does and does not show up mechanistically, not as an independent correctness result.

KV change can flip early in decoding, cascading into wording divergence; the contrast is sharpest on `full_code_refl`, the dense model's most byte-faithful trajectory (dense 0.72, MoE 0.01 at $K$=0). `fp16` was not isolated as a possible contributor, and this is a single routing configuration on one fully-swept trajectory.

Table 9: Diagnostic instruments for the MoE check (`react_llama3` only, $n$=155): `byte-identity` vs. `reprefill` by $K$, dense `Qwen3-8B` vs. MoE `Qwen3-30B-A3B`, and `same_outcome` as the range across the sweep—same format as Table 8. The dense row shows the usual threshold collapse; the MoE row sits at the floor already at $K$=0, so there is no cliff to show. Neither metric is Q3's correctness verdict; see Figure 6.

| **Metric (byte-id.)** | **K0** | **K1** | **K2** | **K4** | **K8** | **K∞** |
|---|---|---|---|---|---|---|
| Qwen3-8B (dense) | 0.22 | 0.01 | 0.01 | 0.01 | 0.00 | 0.00 |
| Qwen3-30B-A3B (MoE) | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| `same_outcome`, dense | | 0.92–0.97, no trend with $K$ | | | | |
| `same_outcome`, MoE | | 0.93–0.96, no trend with $K$ | | | | |

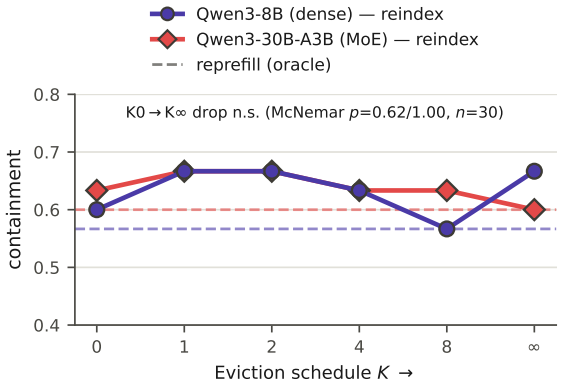


Figure 6: `containment` (`answer` subset, $n$=30, `react_llama3`), dense vs. MoE. Both track their own `reprefill` reference with no systematic drift across $K$; the $K$=0 → $K$=∞ decline is not significant on either model (continuity-corrected McNemar, annotated).

### A.9 RESIDUAL ERROR SOURCE: CONTAMINATION, NOT THE OPERATION

Where composite-KV output diverges from `reprefill`, the cause is *contamination*, the influence of formerly-present junk on the surviving tokens' keys and values, not the prune-and-reindex operation itself. The split is stark at the byte level: on contamination-free samples, `pruning_reindex` reproduces `reprefill`'s output verbatim 0.851 (Llama) / 0.833 (Qwen) of the time, while on contaminated samples byte-identity collapses to 0.111 / 0.089. The operation itself cannot explain that collapse: on clean samples the re-rotation is an exact identity ($\Delta p{=}0$), so what separates the two groups is contamination, not the splice. Contamination is also the common case here, not a corner: 83% of all samples and ≈90% of answerable ones carry some, at a median magnitude of ≈23% of the surviving tokens.

What that byte-level collapse does *not* do is measurably cost task correctness. Testing the causal effect directly, by comparing `pruning_reindex` against `reprefill` restricted to contaminated, answerable samples, where `reprefill` is by construction contamination-free, the difference is $-0.005$ on Llama ($p{=}0.628$, CI $[-0.018, +0.008]$) and $-0.007$ on Qwen ($p{=}0.502$, CI $[-0.022, +0.007]$). These $p$-values match the full-population test in Table 3 exactly by construction, not transcription: McNemar depends only on discordant pairs, and the contamination-free samples excluded here contribute none, so $b$ and $c$ are unchanged and only $\Delta$ moves with $n$. Neither is statistically distinguishable from zero: we detect *no* causal cost of contamination on task correctness on either model. This is a claim about detection, not about the absence of an effect, since the CIs still admit a cost of up to roughly 2 pp, but it is a symmetric, two-model null. Functional fidelity is correspondingly high: `pruning_reindex` reaches the same right/wrong verdict as a full recompute on 0.975 (Llama) / 0.970 (Qwen) of answerable samples, recovering 98.0% / 97.7% of `reprefill`'s correct answers. Nor does this cost accumulate with depth, as the depth-stability analysis in §4.3 establishes. This empirically separates our error source from the assembly-camp's error source: papers such as CacheBlend and EPIC stitch independently-encoded blocks and require boundary recomputation to repair missing cross-attention (Yao et al., 2025); our error is entirely attributable to the attention-mediated influence of content that was present and has been removed, not to missing cross-attention at seams.

Table 10: Attribution: causal effect of contamination on `containment` and depth stability. Contamination governs byte-level fidelity but carries no statistically detectable cost in task correctness on either model, and does not grow with depth; the operation itself contributes ≤0.8% overhead and no additional quality loss beyond contamination. The four depth rows partition the contamination-free samples, so bucket-level fluctuation is sampling noise.

| **Measurement** | **Llama-128K** | **Qwen-40K** |
|---|---|---|
| Contamination causal cost (McNemar $\Delta$) | $-0.005$ ($p{=}0.628$, n.s.) | $-0.007$ ($p{=}0.502$, n.s.) |
| Byte-identity, clean samples (all depths) | 0.851 ($n{=}134$) | 0.833 ($n{=}126$) |
| Byte-identity, contaminated samples | 0.111 ($n{=}657$) | 0.089 ($n{=}641$) |
| Byte-identity, clean samples, rounds 1–15 | 0.833 ($n{=}48$) | 0.792 ($n{=}48$) |
| Byte-identity, clean samples, rounds 16–20 | 0.765 ($n{=}17$) | 0.765 ($n{=}17$) |
| Byte-identity, clean samples, rounds 21–25 | 0.750 ($n{=}16$) | 0.875 ($n{=}16$) |
| Byte-identity, clean samples, rounds 26–100 | 0.925 ($n{=}53$) | 0.889 ($n{=}45$) |
| Re-indexing compute overhead | +0.8% wall time | |

### A.10 LIMITATIONS DETAIL

The following finer-grained limitations are recorded here for completeness.

**K-sweep dwell-time resolution.** The K-sweep (§4.5) covers $K \in \{0, 1, 2, 4, 8, \infty\}$ on three AppWorld trajectories × two models, with $K{=}1, 2, 4$ added specifically to resolve the shape of the curve between the near-immediate ($K{=}0$) and long-dwell ($K{=}8, \infty$) extremes. This confirms that essentially all of the byte-identity dose-response signal is concentrated in $K{\leq}2$: two of the three trajectories reach their floor at $K{=}1$, the third (`full_code_refl`) at $K{=}2$, and $K{=}4, 8$ mainly re-confirm the floor rather than reveal further change. Additional dose points beyond $K{=}4$ are therefore unlikely to alter the paper's conclusions; the more valuable next step, if pursued, would be finer resolution between $K{=}0$ and $K{=}2$. The K-sweep also does not include the `react_gpt4o`

trajectory (K=0 only) or `test_challenge` conditions, leaving trajectory-type generalization partially open, though the independent BrowseComp-Plus K-sweep (§4.5) reproduces the same threshold/plateau split on a different, real-trajectory dataset.

**Task-correctness signal strength.** The `same_outcome` plateau result rests primarily on the `action` task type (API call generation), where empty targets create a structural accuracy floor that inflates agreement. On the `answer` task type ($n\approx$27–33 per trajectory), same-outcome agreement shows a real, gradual attenuation in three of four affected trajectory$\times$model pairs (roughly 12–16 pp from $K{=}0$ to $K{=}8$, saturating rather than continuing to fall); the fourth pair (`plan_exec`-Qwen) shows no decline and, given its small sample size, appears noise-dominated rather than a genuine reversal. Larger-scale replication specifically targeting the answer task type would provide a more precise bound on this crack. The same power limitation applies to this paper's headline correctness result: the `containment` cost identified in §4.5 is likewise too small to reach significance on AppWorld's $n{=}90$-per-model K-sweep, which is why the larger BrowseComp-Plus replication ($n\approx$780) carries the statistical weight for that claim.