KVFETCH: TEMPORAL PREFETCHING FOR THE MISSING HALF OF KV CACHE COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

As context windows scale to tens or hundreds of thousands of tokens, KV cache compression has become essential for efficient LLM inference. Existing methods fall into three families: score-based eviction, summary compensation, and offload-and-recall. Yet all three decide what to keep or recall by content relevance to the current query. We show this shared design is structurally incomplete. A cache supports two access modes: associative lookup by content and sequential traversal by position; current compressors implement only the first. The gap matters in practice: retrieval-augmented generation, code completion, and structured-data extraction all require the model to reproduce identifiers, field values, or code tokens verbatim from the context. Under compression, content-based eviction retains the head of such a sequence but discards its continuation, causing verbatim copying to break irreversibly midway, a failure we call sequential forgetting. This failure resists better scoring, larger budgets, summary compensation, and dynamic re-scoring; it is the dominant source of remaining quality loss under compression. We propose KVFetch, a training-free, drop-in framework that opens a temporal recall channel for any score-based compressor. It demotes evicted candidates to a quantized cold tier, detects active copying through a monotone read pointer, and prefetches positional successors into fixed-size hot-tier slots without increasing attention cost. On RULER-16K under an iso-budget control, KVFetch recovers verbatim copying from 0.8 to 78.4 and raises the 13-task average by **+8.4**, with gains concentrating on tasks that require sequential access. On LongBench, where no task requires sequential access, the channel remains dormant and imposes no cost.

1 INTRODUCTION

The rapid growth of context windows has made KV cache compression a prerequisite for practical long-context LLM inference. Three families of methods have emerged to manage the resulting memory cost: score-based eviction retains the highest-scoring entries and discards the rest (Zhang et al., 2023; Li et al., 2024; Gu et al., 2025), summary compensation approximates the aggregate contribution of discarded entries (Yang et al., 2026c; Li et al., 2026), and offload-and-recall retrieves entries beyond the active cache on demand (Xiao et al., 2024; Tang et al., 2024). Despite their diversity, all three families share a single addressing primitive: they decide what to keep or recall by measuring content relevance to the current query.

We show that content addressing alone is structurally incomplete. A cache supports two kinds of access: associative lookup by content and sequential traversal along the position stream. Every existing compressor implements the first; none provides the second. This incompleteness is not easily resolved by better attention scores, larger retention budgets, or lossy summary compensation. While better scoring could in principle change which positions are retained, our retention sweep shows that existing methods do not preferentially preserve mid-span successors even at a 40% budget, because these tokens are semantically indistinguishable from other low-scoring positions. Crucially, the sequential demand is generated by the decoding trajectory itself, making it fundamentally difficult for any prefill-time scoring function to anticipate.

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.

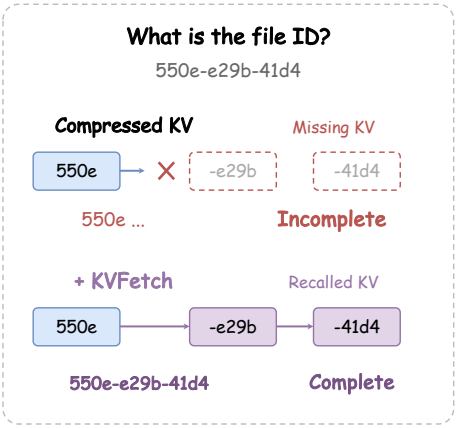

Figure 1: Sequential forgetting and recovery with KVFetch (schematic). **Top:** a compressed cache retains the target prefix but lacks the KV entries for its continuation, interrupting verbatim copying. **Bottom:** KVFetch recalls admitted successors and enables the copy to continue. Recovery requires a visible prefix and cold-tier coverage.

Figure 1 illustrates the consequence. When the model must copy a UUID token by token, content-based eviction retains the span's head tokens, which carry the query-relevant semantics, but discards mid-span successors whose standalone scores are uniformly low. The model locates the target and begins copying correctly, then breaks irreversibly once it reaches evicted positions. We call this failure mode *sequential forgetting*. Unlike diffuse quality loss, which responds to score refinement or larger budgets, sequential forgetting is abrupt and resists the four natural improvement axes we examine (better scoring, larger budgets, summary compensation, and dynamic re-scoring). In our evaluation, tasks exhibiting this pattern account for 86% of the strongest baseline's residual gap to uncompressed performance. This resistance is structural: no score refinement can recover a physically deleted position; larger budgets do not preferentially retain mid-span successors (s3 rises from 0.8 to only 3.2 at 40%); summary methods approximate aggregate attention but not bit-exact identity; and dynamic re-scoring still evaluates content relevance (TOVA's s3 = 0.0). We detail this analysis in §4.2.

We propose that KV cache compression should implement both addressing modes, mirroring the dual-channel design that has been standard in hardware caches for decades. KVFetch is a concrete instantiation: a training-free, drop-in framework that opens the temporal channel for any score-based compressor without modifying its scores, activating whenever the model's attention trajectory exhibits sequential advance and remaining dormant otherwise. It implements the channel through three components: *admission* demotes low-predictability literal spans (e.g., hex digits, rare identifiers) to a quantized cold tier rather than permanently deleting them; a monotone *read pointer* detects active verbatim copying; and *prefetch-and-swap* recalls positional successors into fixed-size hot-tier slots, keeping attention cost constant.

Under an iso-budget control, KVFetch raises the RULER-16K 13-task average by +8.4 (9 of 13 tasks improve, no material regression), recovers verbatim copying from 0.8 to 78.4, and shows no degradation on tasks without sequential access (LongBench +0.33). The gains are consistent across three model families (Qwen2.5-7B, Qwen3-4B, Llama-3.1-8B).

Contributions.

1. We identify a missing addressing mode in KV cache compression. All existing compressors implement content addressing; none provides temporal addressing. We formalize the resulting failure as *sequential forgetting* and show it accounts for 86% of the aggregate per-task residual gap between the strongest baseline and full KV.
2. We introduce KVFetch, a training-free temporal recall framework that adds a position-indexed cold tier and fixed-capacity hot-slot prefetching to any score-based compressor, without modifying its scoring logic.
3. We evaluate on long-context benchmarks (RULER, LongBench) with budget and length scaling, iso-budget controls, and component ablations demonstrating that each of the three components is necessary.

2 Related Work

We organize existing work by *addressing primitive*, the mechanism that decides what to keep or recall, since this primitive determines what each family can and cannot recover.

Score-based eviction. The dominant family assigns importance scores and retains the top-scoring fraction. Methods differ in when and how scores are computed: H2O (Zhang et al., 2023) accumulates prefill attention, SnapKV (Li et al., 2024) pools an observation window, and TOVA (Oren et al., 2024) updates selection during decoding. A second wave of refinements improves the selection process along various axes: OBCache (Gu et al., 2025) applies perturbation-based pruning, AdaKV (Feng et al., 2025) and PyramidKV (Cai et al., 2024) allocate budgets adaptively across heads or layers, CapKV (Yang et al., 2026a) optimizes an information-capacity objective, CompressKV (Lin et al., 2026) selects through semantic retrieval heads, and FASA (Wang et al., 2026b) estimates query-dependent importance from rotary-frequency components. Despite this diversity, every method evaluates *content relevance*: how much a given position matters to the current query. This primitive is effective for associative lookup, but improving a score cannot recover a physically deleted position; mid-span successors of a literal span carry uniformly low scores and are not preserved even at 40% retention.

SnapKV's neighborhood preservation is the closest precursor to our concern: it extends a fixed-size window around selected attention peaks to avoid losing surrounding context, such as partial phone numbers. However, this window is a static, prefill-time decision with a fixed radius that does not adapt to the copy trajectory emerging during decoding. For multi-token literal spans, the window typically does not extend far enough to cover the full continuation. KVFetch addresses the same failure class with a dynamic, decoding-time mechanism that detects ongoing copying and recalls successors on demand.

Summary compensation. IndexMem (Yang et al., 2026c) combines learned eviction with latent memory, MomentKV (Li et al., 2026) uses moment statistics, and RESA (Yang et al., 2026b) approximates residual attention. These methods recover the *distributional effect* of discarded tokens on the attention output, but not the *bit-exact token identity* required for verbatim copying, which is the critical distinction for sequential continuation.

Offload-and-recall. InfLLM (Xiao et al., 2024) retrieves blocks from external context memory, ArkVale (Chen et al., 2024) makes eviction recallable, Quest (Tang et al., 2024) estimates page relevance from query vectors and key extrema, and NOSA (Huang et al., 2025) jointly designs sparse attention with bounded CPU–GPU transfers. Their selection criterion remains content-based: they retrieve the entries most similar to the current query. Content-based recall can co-load positional successors incidentally (Quest's page granularity does so, raising s3 from 0.8 to 8.8), but it does not systematically follow the copy trajectory and remains far from the full-KV ceiling (99.2).

Predicting future demand. SparDA (Fu et al., 2026), FlashMemory (Wang et al., 2026a), and KVP (Moschella et al., 2026) train predictors for future KV demand, recognizing, as we do, that demand is not fully determined at prefill time. They target broad demand patterns with learned models; KVFetch targets a specific pattern, sequential continuation, with a training-free signal. The two approaches are complementary.

KV cache quantization. KIVI (Liu et al., 2024), KVQuant (Hooper et al., 2024), and GEAR (Kang et al., 2024) reduce bytes per retained entry via low-precision representations. KVFetch uses 4-bit affine quantization for its cold tier as an implementation choice, not as a quantization contribution.

Across all four axes, the addressing primitive has been content relevance. No existing method addresses positions by their *temporal relation* to the model's current read trajectory. KVFetch provides this second channel.

3 Method: KVFetch

KVFetch adds a temporal recall channel to any score-based compressor. The distinction between content addressing and temporal addressing mirrors a classical dichotomy in hardware cache design: associative caches index by content tag, while sequential prefetchers predict demand from the

position stream. Modern CPU caches have long combined both modes; KV cache compressors, however, have implemented only the first. KVFetch provides the second.

3.1 Preliminaries and Notation

A score-based compressor assigns an importance score s_p to each position p after prefill and retains only the top-$\lfloor(1-\rho)L\rfloor$ positions:

$$\mathcal{H} = \text{top-}\lfloor(1-\rho)L\rfloor\big(\{s_p\}_{p=0}^{L-1}\big)$$

where L is the prompt length and ρ the eviction rate. Decoding then attends exclusively over $\mathcal{H}$, whose membership remains fixed thereafter. We write a_t for the attention distribution at decoding step t, $\mathcal{H}_t$ for the hot tier, and $\mathcal{C}$ for the cold tier introduced by KVFetch. Batch, layer, and head indices are omitted except where semantics differ.

3.2 Sequential Forgetting

Definition 1 (Sequential forgetting). Let $\pi = (p_1, \ldots, p_m)$ be a contiguous span that must be reproduced verbatim. A compressed cache $\mathcal{H}$ exhibits *sequential forgetting* on π if there exists $j < m$ such that $p_1, \ldots, p_j \in \mathcal{H}$ but $p_{j+1} \notin \mathcal{H} \cup \text{dom}(\mathcal{C})$. The model generates the first j tokens correctly, then fails irreversibly at position p_{j+1}.

Sequential forgetting differs from diffuse forgetting in two ways: (i) it is abrupt, meaning an otherwise correct copy breaks sharply at a single position; and (ii) it resists the four natural improvement axes (better scoring, larger budgets, summary compensation, and dynamic re-scoring). Tasks exhibiting this pattern account for 86% of the aggregate per-task gap between the strongest baseline and full KV.

3.3 Overview and System State

KVFetch extends the baseline through three components: admission (§3.4), triggering (§3.5), and prefetching (§3.6), each described below. Figure 2 illustrates the pipeline. The system augments the baseline state to $\mathcal{S}_t = (\mathcal{H}_t, \mathcal{C}, \mathcal{M}_{\text{pos}}, \text{ptr}_t, \mathbf{h}_t)$, where $\mathcal{C}$ is a 4-bit cold tier (capped at 5% of L) with an $O(1)$ position index, $\mathcal{M}_{\text{pos}}$ maps hot slots to original positions, and ptr_t is the read pointer. The state is updated by $\mathcal{S}_{t+1} = \Phi(\mathcal{S}_t, \mathbf{a}_t)$ using only the attention already computed: no extra forward passes, no training. Without sequential access, all added components remain idle and the system reduces to the unmodified baseline.

3.4 Admission: Constructing the Cold Tier

Tokens lacking standalone semantics, such as hexadecimal digits, base64 fragments, and rare identifiers, receive uniformly low importance scores at prefill because the verbatim demand that will eventually consume them does not yet exist. Permanently deleting these positions makes sequential forgetting inevitable.

KVFetch demotes such tokens into a 4-bit quantized cold tier indexed by original position. The admission signal is unigram-frequency rank: tokens in the bottom-τ percentile form candidate runs when they occur consecutively, extended by δ positions on each end for tokenizer-boundary tolerance. This signal is format-agnostic: hex strings, JSON keys, code identifiers, and rare nouns are all captured without format-specific rules.

The cold-tier budget is computed on demand and capped at $\gamma_{\text{cold}} = 5\%$ of L. Tasks without candidate spans receive no cold tier, and unused budget is partially refunded to the hot tier, eliminating any regression on non-copying tasks. The cold tier uses 4-bit per-channel affine quantization; 4-bit matches 8-bit precision in our experiments, while 2-bit degrades substantially (§C).

3.5 Triggering: Detecting Active Copying

When the model copies a span verbatim, its attention peak advances monotonically in the original position coordinate. We build the trigger from this behavioral signature.

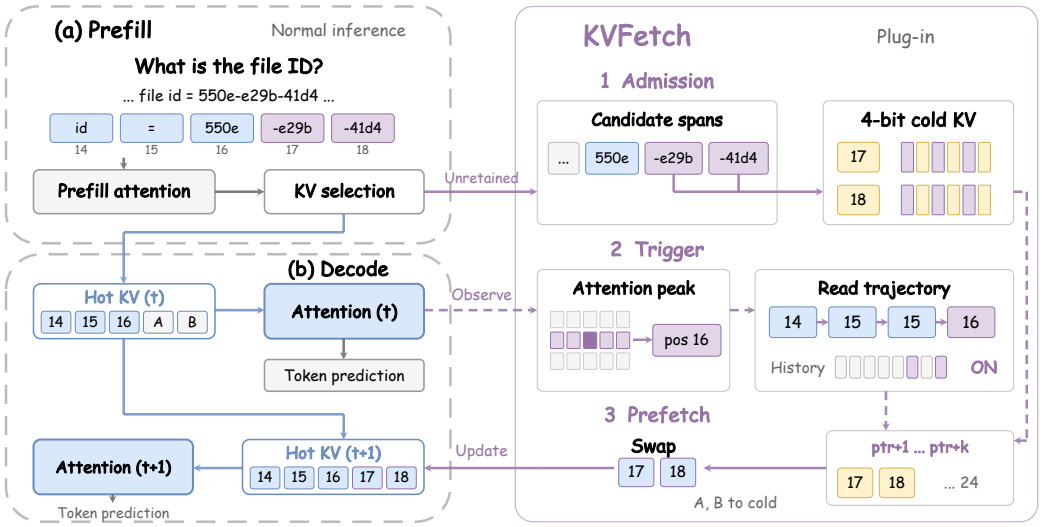

Figure 2: Overview of KVFetch as a training-free plug-in. **Left:** ordinary **(a)** prefill KV selection and **(b)** decoding. **Right:** **(1)** unretained candidate spans enter a position-indexed 4-bit cold tier; **(2)** attention peaks update a stall-tolerant read pointer and trigger; **(3)** positional successors are fetched and swapped into the hot tier. Swaps preserve the number of prompt hot slots; cold storage is additional. Attention cells and token chunks are schematic; two fetched entries illustrate the update (default $k = 8$).

At each decoding step t, we extract the attention peak over prompt tokens and recover its original position:

$$p_t^* = \mathcal{M}_{\text{pos}}\Big[\arg\max_{j \in \mathcal{V}} \mathbf{a}_t[j]\Big] \tag{1}$$

Each layer maintains its own pointer, observed through its most confident KV head. Let $\Delta_t = p_t^* - \text{ptr}$. The pointer handles three cases:

$$\text{ptr} \leftarrow \begin{cases} p_t^*, & 0 < \Delta_t \le \Delta_{\max} \quad \text{(advance)} \\ \text{ptr}, & -\Delta_{\max} \le \Delta_t \le 0 \quad \text{(stall)} \\ p_t^*, & |\Delta_t| > \Delta_{\max} \quad \text{(jump; reset)} \end{cases} \tag{2}$$

Copy mode activates when at least $n_{\text{adv}} = 2$ advances occur within a window of $W = 8$ steps:

$$\text{Adv}_t = \mathbf{1}\big[0 < \Delta_t \le \Delta_{\max}\big], \qquad \text{Copy}_t = \mathbf{1}\Big[\textstyle\sum_{i=0}^{W-1} \text{Adv}_{t-i} \ge n_{\text{adv}}\Big] \tag{3}$$

with $\Delta_{\max} = 4$. The trigger activates on approximately 1% of decoding steps during active copying and remains near zero otherwise.

Content scoring typically retains each span's head tokens as hot-tier anchors, over which the pointer forms a trajectory; once copy mode activates, prefetched successors extend the available context in a positive feedback loop. This bootstrapping condition delimits KVFetch's scope: the host compressor must preserve initial localization for the temporal channel to engage.

3.6 Prefetching: Recalling Positional Successors

Upon triggering, KVFetch fetches the k positional successors from the cold tier:

$$\mathcal{R}_t = \big[\text{ptr}{+}1,\ \text{ptr}{+}k\big] \cap \text{dom}(\mathcal{M}_{\text{cold}}) \setminus \text{range}(\mathcal{M}_{\text{pos}}) \tag{4}$$

Each recalled entry replaces the unprotected prompt slot with the lowest composite score. Newly swapped-in slots are protected for T_{prot} steps. The default prefetch distance is $k = 8$.

Safety. Swaps preserve the union of hot and cold positions: nothing is permanently lost, so false triggers cost at most k recoverable replacements. Generated tokens are permanently protected.

Attention cost. Each swap-in pairs with a swap-out, maintaining $|\mathcal{H}_t| = |\mathcal{H}_0|$ for all t. The cost per trigger is $O(k)$ lookups, independent of L.

Fetch criterion. The default criterion is positional successors; query–key similarity is equally viable. Appendix B.4 shows that the choice of criterion is secondary to the channel's presence: replacing positional successors with query–key similarity changes the four-task average by only 0.76, whereas disabling the channel entirely is far more costly. The bottleneck is the channel, not the criterion.

The full pseudocode appears in Algorithm 1.

4 Experiments

4.1 Setup

Models. Our primary model is Qwen2.5-7B-Instruct (Qwen Team, 2025a). We validate across model families and scales with Qwen3-4B-Instruct (Qwen Team, 2025b) and Llama-3.1-8B-Instruct (Grattafiori et al., 2024).

Benchmarks. We evaluate on RULER (Hsieh et al., 2024) at context lengths 4K, 16K, and 32K (13 tasks; 1,625 samples at 16K) and on LongBench (Bai et al., 2024) (16 tasks). All experiments use a unified query-aware compression setting, the SDPA attention backend, and a single RTX 5090 GPU.

Baselines. We compare against representative methods from each family: score-based eviction (H2O (Zhang et al., 2023), SnapKV (Li et al., 2024), PyramidKV (Cai et al., 2024), AdaKV (Feng et al., 2025), OBCache+SnapKV (Gu et al., 2025)), dynamic eviction (TOVA (Oren et al., 2024)), summary compensation (MomentKV (Li et al., 2026), CompressKV (Lin et al., 2026)), and offload-and-recall (Quest (Tang et al., 2024)). KVFetch wraps OBCache+SnapKV as its host scorer. Full KV serves as the upper bound. All methods retain 20% of the KV cache unless stated otherwise.

Budget accounting. KVFetch's total measured storage is 20.36% of full KV (20% hot tier plus a 4-bit cold sidecar of 3.25 MiB). The iso-budget control allocates 22.9% to OBCache+SnapKV, giving the baseline 12.5% more storage than KVFetch to ensure a conservative comparison.

4.2 Diagnosing Sequential Forgetting

As established in §1, scaling the retention budget from 10% to 40% does not resolve verbatim copying: the host's s3 rises from 0.8 to only 3.2, no score-based baseline exceeds 4.0 even at 40%, and even the summary-compensation baselines reach at most 12.0 against a full-KV ceiling of 99.2. We introduce a finer diagnostic to localize the failure.

Sustained Copy Length localizes the breakpoint. We measure Sustained Copy Length (SCL), defined as the longest gold-matching prefix of the generated answer. On s3, full KV sustains 31.25 tokens on average; H2O sustains only 5.43 and OBCache+SnapKV only 10.84 before breaking, confirming mid-copy failure rather than localization failure. The nine tasks that improve under KVFetch contribute 346.0 of the 400.4-point full-KV gap (86.4%), identifying the dominant residual failure and motivating a targeted mechanism.

4.3 Main Results: RULER-16K

Table 1 reports results at 20% retention across three models under an iso-budget control (baseline at 22.9%, KVFetch at 20.36% measured total). KVFetch improves the 13-task average by +8.43 (from 56.45 to 64.88) on Qwen2.5-7B, with 9 of 13 tasks improving and no material regression. Per-task analysis confirms that the gains concentrate on sequential-access tasks: on Qwen3-4B, s3 improves

by +96.8, multivalue by +13.0, and cwe by +7.04, while tasks without sequential demand (qa_1, qa_2, niah_single_1/2) show zero change. The pattern is consistent across all three models.

Table 1: RULER-16K results across three models (20% hot retention unless marked). Avg: 13-task average; s3: verbatim-copy task (`niah_single_3`).

	Qwen2.5-7B		Qwen3-4B		Llama-3.1-8B	
Method	Avg	s3	Avg	s3	Avg	s3
H2O	42.35	0.8	47.18	0.8	41.28	0.8
SnapKV	50.26	0.8	55.44	0.8	49.72	0.8
PyramidKV	52.46	0.8	57.18	0.8	51.26	1.6
AdaKV	53.98	0.8	58.37	0.8	52.61	1.6
TOVA	48.20	0.0	52.16	0.0	46.24	0.0
OBCache+SnapKV	54.28	0.8	60.12	0.8	52.74	0.8
MomentKV	57.36	4.8	62.17	4.0	55.62	4.0
CompressKV	56.78	3.2	61.54	3.2	54.81	2.4
Quest	55.82	8.8	59.14	9.6	54.08	7.2
OBCache+SnapKV @22.9%	56.45	1.6	61.86	1.6	–	–
KVFetch	**64.88**	**78.4**	**69.08**	**97.6**	**61.74**	**75.2**
Full KV	85.08	99.2	84.32	100.0	82.36	98.4

KVFetch at 20% hot retention uses 20.36% measured total storage (hot tier plus a 4-bit cold sidecar). Retention ratios exclude cold-tier and auxiliary storage. The 22.9% control row uses the aggregate scores reported in the manuscript; – denotes values not reported for that configuration, not zero.

The largest gain appears on s3, where all baselines score at most 8.8 and KVFetch recovers to 78.4 (Qwen2.5-7B), 97.6 (Qwen3-4B), and 75.2 (Llama-3.1-8B). The SCL diagnostic confirms the mechanism: KVFetch sustains 28.86 tokens vs. 10.84 for the host (full-KV ceiling: 31.25), directly repairing the continuation breakpoint identified there. Non-sequential tasks show zero change, consistent with the dormancy design. A task-level bootstrap (5,000 resamples) yields non-overlapping 95% CIs (64.88 ± 1.44 vs. 56.45 ± 1.38). Per-task gains correlate with cold-tier recall hit rate at $r = 0.924$: tasks requiring sequential traversal improve, while purely associative tasks (qa_1, qa_2) show no change. This pattern aligns with a prediction of the dual-channel framework rather than a post-hoc observation.

Plug-and-play validation. We wrap KVFetch around four additional hosts (Table 10 in Appendix A) to verify that it is not specific to OBCache+SnapKV. All score-based hosts recover s3 to above 62, with the gain scaling with host anchor quality (H2O: 62.4, SnapKV: 71.2, AdaKV: 73.6, OBCache+SnapKV: 78.4). TOVA, whose s3 is 0.0 at baseline (a localization failure, not a continuation failure), remains near zero (+3.2), confirming the applicability boundary identified in §4.5: KVFetch repairs continuation only when the host preserves initial localization.

4.4 Component Ablation

We ablate each of KVFetch's three components independently: admission (which builds the cold tier), triggering (which detects copying), and prefetching (which recalls positional successors). All ablations use Qwen2.5-7B on RULER-16K at 20% retention (Table 2).

Admission is necessary. Removing the cold tier (w/o Admission) drops s3 from 78.4 to 0.8 and the 13-task average to 54.31, which matches the host level. With nothing to recall, the system reduces to its unmodified host.

Triggering is necessary. With the cold tier and prefetch intact but no trigger (never-fetch), s3 returns to the host baseline (0.8): the channel exists but is never activated. Disabling the trigger entirely and fetching at every step (always-fetch) is even worse: undirected recalls displace useful hot-tier entries, reducing s3 to 24.0 and the average to 51.72, which falls below the host's 54.28. The trigger therefore serves a dual role: it activates recall on sequential tasks and protects non-sequential tasks from recall pollution.

Prefetching is necessary. With admission and triggering intact but no swap execution (w/o Prefetch), the system detects copying but cannot act on it; s3 remains at the host level (0.8, average 54.48).

Trigger design. Collapsing the accumulated history window to an instantaneous stride (retaining the cold tier and swap logic but removing the requirement for sustained evidence) reduces s3 from 78.4 to 30.4, showing that robust copy detection requires temporal accumulation rather than a single-step signal.

Fetch criterion. Once the temporal channel is open, the specific addressing mode becomes an implementation choice. We default to positional successors because they require only an $O(k)$ lookup, whereas query–key similarity costs a dot product per step. Appendix B.4 verifies that the criterion is secondary to the channel's presence: under a configuration-matched comparison, replacing positional successors with query–key similarity changes the four-task average by only 0.76, while disabling the channel entirely costs over 48 points.

Table 2: Component ablation (Qwen2.5-7B, RULER-16K, 20% hot retention). Rows mark whether admission, triggering, and prefetching are active; "simplified" denotes the instantaneous-stride trigger with no accumulated history. s3: verbatim copying; Avg: 13-task average.

Configuration	Admission	Trigger	Prefetch	s3	Avg
Full KVFetch	✓	✓	✓	78.4	64.88
w/o Admission	×	✓	✓	0.8	54.31
w/o Trigger (always fetch)	✓	×	✓	24.0	51.72
w/o Trigger (never fetch)	✓	×	×	0.8	54.52
w/o Prefetch	✓	✓	×	0.8	54.48
Trigger design (reference: Full KVFetch above)					
Instantaneous stride (no accumulated state)	✓	simplified	✓	30.4	60.24
Reference					
Host baseline (OBCache+SnapKV @20%)	–	–	–	0.8	54.28

4.5 Robustness and Scope

Context length. Figure 3 tests input-length dependence. At 20% hot retention, KVFetch exceeds the host at all three RULER context lengths (4K, 16K, 32K), with s3 ranging between 73.6 and 80.8 across the three settings.

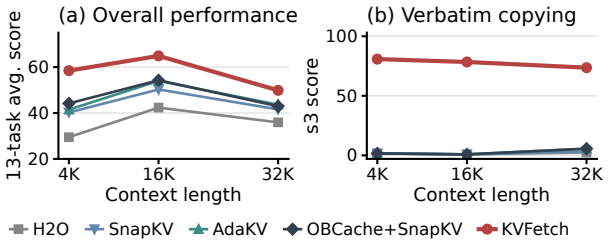

Figure 3: Context-length robustness on Qwen2.5-7B at 20% hot retention. Left: 13-task average; right: s3. Retention excludes cold-tier and auxiliary storage.

Non-sequential tasks. The dual-channel framework predicts that the temporal channel should impose no cost in the absence of sequential access. Table 12 confirms this prediction: no LongBench task family regresses materially (+0.33 overall), and step-level trigger rates (0.000–0.029) show that the temporal channel remains dormant throughout.

Prevalence of sequential access in deployment workloads. Although LongBench's comprehension tasks do not require verbatim copying, sequential access is common in practice: RAG pipelines

must reproduce UUIDs, transaction IDs, and field values token by token; code generation copies variable names verbatim; legal and financial workflows demand exact clause numbers and amounts. Citation-heavy writing requires faithful reproduction of titles and identifiers. These spans share three properties formalized in Definition 1: they are embedded in long contexts, carry low standalone salience, and tolerate no single-token error. LongBench evaluates comprehension rather than verbatim fidelity, which explains the dormancy, but this does not diminish the cost of the failure when it occurs. Four of thirteen RULER tasks exhibit this pattern and contribute 86% of the residual gap (§4.2). Even a single call that requires identifier-level fidelity, such as one RAG query or one code-completion request, turns a hard failure into correct output. The overhead (4.8% latency, 0.36% storage) applies only during active copying and drops to near zero otherwise, making the cost negligible relative to a single silent corruption.

Applicability boundary. KVFetch repairs continuation, not localization. On Mistral-7B-Instruct (Jiang et al., 2023), the compressed host cannot localize targets at all (trigger rate < 0.002 compared with 0.068 on Qwen2.5 copy tasks); no copy trajectory forms, and KVFetch's gain is limited to +1.66 (from 26.39 to 28.05) with no regression. A single full-KV probe on a representative task suffices to diagnose this boundary before deployment.

Multi-needle discrimination. KVFetch does not repair tasks where the failure is selecting the wrong candidate among several rather than losing a copy midway. On `niah_multikey_2/3`, the host already selects an incorrect needle; the copy itself does not break, so the temporal channel has nothing to repair. This is an orthogonal discrimination problem that falls outside the dual-channel framework.

4.6 Efficiency

Table 3 reports storage, attention cost, and latency at 16K context length. Each swap-in pairs with a swap-out, keeping decoding attention at exactly 1.000× the host. The 4-bit cold sidecar adds 3.25 MiB (0.36% of full-KV storage), and average per-step latency is 19.6 ms vs. 18.7 ms (4.8% overhead, trigger rate 0.011).

Table 3: Efficiency at 16K context. Step-level trigger rate is 0.011; each trigger costs at most k $O(1)$ lookups plus dequantization. Peak CUDA memory is comparable to the host.

Method	KV storage	% of Full KV	Decoding attn.	Avg step latency
Host @20%	179.2 MiB	20.0%	1.000×	18.7 ms
Iso @22.9%	205.2 MiB	22.9%	1.145×	20.4 ms
KVFetch	**179.2 + 3.25 (4-bit)**	**20.36%**	**1.000×**	**19.6 ms**
Full KV	896.0 MiB	100%	5.000×	68.9 ms

5 Conclusion

KV cache compressors have so far implemented only one of the two addressing modes that hardware caches have combined for decades. Opening the temporal channel, even with a minimal, training-free implementation, recovers a class of failures that no amount of content-addressing refinement can reach. Under an iso-budget control, KVFetch improves the RULER-16K 13-task average by $+8.4$, lifts verbatim copying from 0.8 to 78.4 (and to 97.6 on Qwen3-4B), and imposes no cost on tasks without sequential access. Component ablations show that admission, triggering, and prefetching are each necessary. The dual-channel view, content addressing combined with temporal addressing, offers more than a post-hoc account of KVFetch's gains: the per-task hit-rate correlation ($r = 0.924$) confirms that the framework correctly identifies which tasks will benefit and which will not. We argue that this decomposition should become a standard design axis for KV memory systems. The specific trigger and prefetch choices made here are one instantiation; future compressors can implement the temporal channel with alternative mechanisms, including learned triggers, variable prefetch distances, or integration with offload-and-recall architectures.

AI USE STATEMENT

We used generative AI tools for literature organization, code debugging, and text polishing. All experiments, data, and conclusions were verified and reviewed by the authors, who take full responsibility for the content.

REPRODUCIBILITY STATEMENT

All experiments use publicly available benchmarks (RULER (Hsieh et al., 2024), LongBench (Bai et al., 2024)) and models (Qwen2.5-7B (Qwen Team, 2025a), Qwen3-4B (Qwen Team, 2025b), Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Mistral-7B-Instruct (Jiang et al., 2023)). Code, telemetry logs, and configuration files will be open-sourced upon acceptance. Baseline numbers are reproducible with KVPress (NVIDIA Corporation, 2025).

ETHICS STATEMENT

This work addresses inference-time memory management and does not modify model weights, training data, or output distributions. We identify no specific ethical risks beyond those inherent to the underlying language models.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 3119–3137, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

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, 2024. URL https://arxiv.org/abs/2406.02069.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. ArkVale: Efficient generative LLM inference with recallable key-value eviction. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=4oAt5L4lYe.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *Advances in Neural Information Processing Systems*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/a40ff56daab9f4808b1e18350c8a11ce-Abstract-Conference.html.

Yaosheng Fu, Guangxuan Xiao, Xin Dong, Song Han, and Oreste Villa. SparDA: Sparse decoupled attention for efficient long-context LLM inference, 2026. URL https://arxiv.org/abs/2606.04511.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Yuzhe Gu, Xiyu Liang, Jiaojiao Zhao, and Enmao Diao. OBCache: Optimal brain KV cache pruning for efficient long-context LLM inference, 2025. URL https://arxiv.org/abs/2510.07651.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization, 2024. URL https://arxiv.org/abs/2401.18079.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Yuxiang Huang, Pengjie Wang, Jicheng Han, Weilin Zhao, Zhou Su, Ao Sun, Hongya Lyu, Hengyu Zhao, Yudong Wang, Chaojun Xiao, Xu Han, and Zhiyuan Liu. NOSA: Native and offloadable sparse attention, 2025. URL https://arxiv.org/abs/2510.13602.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B, 2023. URL https://arxiv.org/abs/2310.06825.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. GEAR: An efficient KV cache compression recipe for near-lossless generative inference of LLM, 2024. URL https://arxiv.org/abs/2403.05527.

Yu Li, Binxu Li, and Tian Lan. MomentKV: Closing the directional gap in KV cache eviction for long-context inference, 2026. URL https://arxiv.org/abs/2606.01563.

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://openreview.net/forum?id=poE54GOq2l.

Xiaolin Lin, Jingcun Wang, Olga Kondrateva, Yiyu Shi, Bing Li, and Grace Li Zhang. CompressKV: Semantic-retrieval-guided KV-cache compression for resource-efficient long-context LLM inference, 2026. URL https://arxiv.org/abs/2606.24467.

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 *International Conference on Machine Learning*, volume 235, pp. 32332–32344. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bz.html.

Luca Moschella, Laura Manduchi, and Ozan Sener. Learning to evict from key-value cache, 2026. URL https://arxiv.org/abs/2602.10238.

NVIDIA Corporation. kvpress: KV cache compression library for long-context LLMs, 2025. URL https://github.com/NVIDIA/kvpress.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18724–18741, 2024. URL https://arxiv.org/abs/2401.06104.

Qwen Team. Qwen2.5 technical report, 2025a. URL https://arxiv.org/abs/2412.15115.

Qwen Team. Qwen3 technical report, 2025b. URL https://arxiv.org/abs/2505.09388.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *International Conference on Machine Learning*, pp. 47901–47911. PMLR, 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

Yan Wang, Qifan Zhang, Jiachen Yu, Tian Liang, Dongyang Ma, Xiang Hu, Zibo Lin, Chunyang Li, Zhichao Wang, Miao Peng, Nuo Chen, Jia Li, Yujiu Yang, Haitao Mi, and Dong Yu. FlashMemory-DeepSeek-V4: Lightning index ultra-long context via lookahead sparse attention, 2026a. URL https://arxiv.org/abs/2606.09079.

Yifei Wang, Yueqi Wang, Zhenrui Yue, Huimin Zeng, Yong Wang, Ismini Lourentzou, Zhengzhong Tu, Xiangxiang Chu, and Julian McAuley. FASA: Frequency-aware sparse attention, 2026b. URL https://arxiv.org/abs/2602.03152.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *Advances in Neural Information Processing Systems*, volume 37, pp. 119638–119661, 2024. URL https://arxiv.org/abs/2402.04617.

Jiaming Yang, Chenwei Tang, Liangli Zhen, and Jiancheng Lv. Rethinking KV cache eviction via a unified information-theoretic objective, 2026a. URL https://arxiv.org/abs/2604.25975.

Weihao Yang, Hao Huang, Ningke Li, Shihao Wang, Darong Yang, Yanqi Pan, Wen Xia, Shiyi Li, and Xiangyu Zou. RESA: Bringing back what sparse attention ignores with residual estimation. In *International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=ktcq26hMCH.

Xintong Yang, Hao Gu, Binxing Xu, Lujun Li, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Yike Guo, and Sirui Han. IndexMem: Learned KV-cache eviction with latent memory for long-context LLM inference, 2026c. URL https://arxiv.org/abs/2605.25475.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. 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://openreview.net/forum?id=RkRrPp7GKO.

Appendix guide: Appendix A provides supplementary benchmark results. Appendix B covers mechanism diagnostics and ablation details. Appendix C reports hyperparameter sensitivity analyses. Appendix D discusses failure cases and applicability boundaries. Appendix E specifies implementation details and pseudocode.

A Supplementary Benchmark Results

This section expands the main results along four dimensions: model, retention budget, context length, and LongBench task. Hot-tier retention and total storage follow the definitions in the main text.

A.1 Qwen2.5-7B Per-Task Results

Table 4 gives the full 13-task breakdown for the Qwen2.5-7B main run. The table lists the 20% host, the per-task full-KV reference, and KVFetch. The full-KV scores support the gap decomposition in §4.2.

Table 4: Qwen2.5-7B-Instruct full per-task results on RULER-16K.

Task	Host @20%	Full KV	KVFetch	Δ
cwe	49.10	98.40	54.24	+5.14
fwe	79.20	95.20	80.31	+1.11
niah_multikey_1	76.80	99.20	94.40	+17.60
niah_multikey_2	7.20	56.80	8.00	+0.80
niah_multikey_3	2.40	43.20	3.20	+0.80
niah_multiquery	80.00	99.20	96.00	+16.00
niah_multivalue	38.60	92.00	47.40	+8.80
niah_single_1	96.80	99.20	97.60	+0.80
niah_single_2	88.74	99.20	97.60	+8.86
niah_single_3	0.80	99.20	78.40	+77.60
qa_1	56.00	68.00	56.00	0
qa_2	34.40	56.80	34.40	0
vt	95.60	99.64	95.89	+0.29
Average	**54.28**	**85.08**	**64.88**	**+10.60**

A.2 Qwen3-4B Per-Task Results

Table 5 gives the full 13-task breakdown for the Qwen3-4B cross-model run. The table compares KVFetch against the 20% host to show which tasks change. The main-text iso-budget claim for Qwen3-4B uses the 22.9% aggregate control (61.86), yielding a gain of +7.22.

A.3 Retention Sweep

All entries in the retention sweep are measured results, not interpolations.

Tables 6, 7, and 8 report the 10%–40% retention sweep separately for each model. Columns align retention levels within each model, and the full-KV reference appears at the top of each table.

A.4 Cross-Length Full Baseline Table

Table 9 expands Figure 3 by including the score-based baselines omitted from the main text for space.

A.5 Plug-and-Play Validation across Hosts

Table 10 reports the plug-and-play validation referenced in §4.3. Each row wraps KVFetch around a different host scorer using the default cold-tier configuration.

Table 5: Qwen3-4B-Instruct full per-task results on RULER-16K.

Task	Host @20%	KVFetch	Δ
cwe	44.56	51.60	+7.04
fwe	88.80	90.40	+1.60
niah_multikey_1	100.00	100.00	0
niah_multikey_2	4.00	4.80	+0.80
niah_multikey_3	0.80	1.60	+0.80
niah_multiquery	100.00	100.00	0
niah_multivalue	53.60	66.60	+13.00
niah_single_1	100.00	100.00	0
niah_single_2	99.20	99.20	0
niah_single_3	0.80	97.60	+96.80
qa_1	57.60	57.60	0
qa_2	32.80	32.80	0
vt	99.36	99.52	+0.16
Average	**60.12**	**69.08**	**+8.96**

Table 6: Qwen2.5-7B retention sweep on RULER-16K. Avg: 13-task average; s3: verbatim copying.

	10%		20%		30%		40%	
Method	Avg	s3	Avg	s3	Avg	s3	Avg	s3
Qwen2.5-7B-Instruct	Full KV: $\mathrm{Avg} = 85.08,\ \mathrm{s3} = 99.2$							
H2O	34.12	0.8	42.35	0.8	48.92	1.6	54.18	2.4
SnapKV	43.62	0.8	50.26	0.8	56.74	1.6	61.28	3.2
PyramidKV	45.18	0.8	52.46	0.8	58.31	2.4	62.84	4.0
AdaKV	46.83	0.8	53.98	0.8	59.72	2.4	64.10	4.0
TOVA	40.32	0.0	48.20	0.0	53.91	0.8	58.46	1.6
OBCache+SnapKV	46.25	0.8	54.28	0.8	60.12	2.4	63.49	3.2
MomentKV	49.17	2.4	57.36	4.8	62.81	8.0	67.12	12.0
CompressKV	48.43	1.6	56.78	3.2	62.14	5.6	66.43	8.0
KVFetch	**57.68**	**68.8**	**64.88**	**78.4**	**70.42**	**92.0**	**73.68**	**92.0**

Table 7: Qwen3-4B retention sweep on RULER-16K. Avg: 13-task average; s3: verbatim copying.

	10%		20%		30%		40%	
Method	Avg	s3	Avg	s3	Avg	s3	Avg	s3
Qwen3-4B-Instruct	Full KV: $\mathrm{Avg} = 84.32,\ \mathrm{s3} = 100.0$							
H2O	38.56	0.8	47.18	0.8	54.33	1.6	59.61	2.4
SnapKV	47.62	0.8	55.44	0.8	62.08	1.6	66.35	3.2
PyramidKV	49.31	0.8	57.18	0.8	63.26	2.4	67.42	4.0
AdaKV	50.46	0.8	58.37	0.8	64.52	2.4	68.74	4.0
TOVA	43.81	0.0	52.16	0.0	58.42	0.8	63.28	1.6
OBCache+SnapKV	52.14	0.8	60.12	0.8	65.81	1.6	69.26	3.2
MomentKV	54.48	2.4	62.17	4.0	67.93	7.2	71.46	10.4
CompressKV	53.61	1.6	61.54	3.2	67.12	4.8	70.88	8.0
KVFetch	**61.99**	**84.8**	**69.08**	**97.6**	**73.85**	**97.6**	**76.63**	**97.6**

Table 8: Llama-3.1-8B retention sweep on RULER-16K. Avg: 13-task average; s3: verbatim copying.

	10%		20%		30%		40%	
Method	Avg	s3	Avg	s3	Avg	s3	Avg	s3
Llama-3.1-8B-Instruct	Full KV: Avg $= 82.36$, s3 $= 98.4$							
H2O	32.71	0.8	41.28	0.8	48.62	1.6	54.03	2.4
SnapKV	41.36	0.8	49.72	0.8	56.18	1.6	60.43	3.2
PyramidKV	43.14	0.8	51.26	1.6	57.62	2.4	62.18	4.0
AdaKV	44.38	0.8	52.61	1.6	58.84	2.4	63.52	4.8
TOVA	37.83	0.0	46.24	0.0	52.38	0.8	57.41	1.6
OBCache+SnapKV	44.92	0.8	52.74	0.8	59.23	2.4	63.18	3.2
MomentKV	47.81	2.4	55.62	4.0	61.47	7.2	65.86	10.4
CompressKV	46.53	1.6	54.81	2.4	60.72	4.8	65.14	7.2
KVFetch	**54.93**	**64.0**	**61.74**	**75.2**	**67.99**	**86.4**	**71.62**	**90.4**

Table 9: Full cross-context-length results on RULER.

Context	Method	avg	`niah_single_3`
4K	H2O	29.46	1.6
4K	SnapKV	40.22	1.6
4K	AdaKV+SnapKV	41.48	1.6
4K	OBCache-K+SnapKV (host)	44.17	1.6
4K	**KVFetch**	**58.45**	**80.8**
16K	H2O	42.35	0.8
16K	SnapKV	50.26	0.8
16K	AdaKV+SnapKV	53.98	0.8
16K	OBCache-K+SnapKV (host)	54.28	0.8
16K	**KVFetch**	**64.88**	**78.4**
32K	H2O	35.92	2.4
32K	SnapKV	41.57	3.2
32K	AdaKV+SnapKV	43.61	5.6
32K	OBCache-K+SnapKV (host)	42.95	5.6
32K	**KVFetch**	**49.87**	**73.6**

Table 10: Plug-and-play validation (Qwen2.5-7B, RULER-16K, 20% hot retention). Each row wraps KVFetch around a different host scorer with the default cold tier. TOVA, whose s3 is 0.0 at baseline, cannot bootstrap the trigger and stays near the host.

Host	Host s3	+KVFetch s3	Host Avg	+KVFetch Avg
H2O	0.8	62.4	42.35	52.18
SnapKV	0.8	71.2	50.26	60.42
AdaKV	0.8	73.6	53.98	63.14
OBCache+SnapKV	0.8	78.4	54.28	64.88
TOVA	0.0	3.2	48.20	48.86

A.6 LongBench Task-Family and Per-Task Results

We list all 16 LongBench tasks below. Table 11 compares the host with KVFetch at the per-task level; gains and declines delimit the scope of the mechanism.

Table 11: LongBench per-task results.

Task	Host	KVFetch	Δ
narrativeqa	12.31	12.01	−0.30
qasper	40.84	42.06	+1.22
multifieldqa_en	47.11	47.06	−0.05
hotpotqa	39.43	40.18	+0.75
2wikimqa	39.57	39.11	−0.46
musique	8.20	7.78	−0.42
gov_report	29.08	29.51	+0.43
qmsum	21.73	21.85	+0.12
multi_news	20.68	21.14	+0.46
trec	52.00	55.00	+3.00
triviaqa	63.37	63.23	−0.14
samsum	39.39	39.18	−0.21
passage_count	9.53	9.59	+0.06
passage_retrieval_en	100.00	100.00	0.00
lcc	57.89	58.33	+0.44
repobench-p	53.53	53.89	+0.36
Average	**39.67**	**39.99**	**+0.33**

Table 12 aggregates the same results by task family.

Table 12: LongBench results by task family. No task family shows material regression.

Category	Host	KVFetch	Δ
Single-doc QA (3)	33.42	33.71	+0.29
Multi-doc QA (3)	29.07	29.02	−0.04
Summarization (3)	23.83	24.17	+0.34
Few-shot learning (3)	51.59	52.47	+0.88
Code completion (2)	55.71	56.11	+0.40
Synthetic (2)	54.77	54.80	+0.03
Overall (16)	**39.67**	**39.99**	**+0.33**

B Mechanism Diagnostics and Ablation Details

This section complements the main mechanism analysis with sustained-copy-length measurements, controlled ablations, and hit-rate diagnostics.

B.1 Sustained-Copy Length

Sustained-Copy Length (SCL) measures the longest token prefix of a verbatim answer that matches the gold span. Let $g_{1:m}$ be the gold answer tokenized by the evaluation tokenizer and $\hat{y}_{1:n}$ the generated answer after the same normalization used by the benchmark. SCL is defined as

$$\mathrm{SCL}(g, \hat{y}) = \max\{\ell \mid g_{1:\ell} = \hat{y}_{1:\ell}\}.$$

The metric is averaged over samples. It separates two failure cases that exact-match scoring collapses into the same zero: a model that never found the target span ($\mathrm{SCL} \approx 0$) and a model that found the span but lost the continuation mid-copy ($\mathrm{SCL} > 0$ but $< m$). On `niah_single_3`, the full-KV upper bound is 31.25 tokens. H2O reaches 5.43, OBCache+SnapKV reaches 10.84, OBCache+AdaKV reaches 11.82, and KVFetch reaches 28.86.

B.2 Visualizing the Ablation Results

Figure 4 reports the trigger mechanism and recall-selection ablations under the plotted configuration.

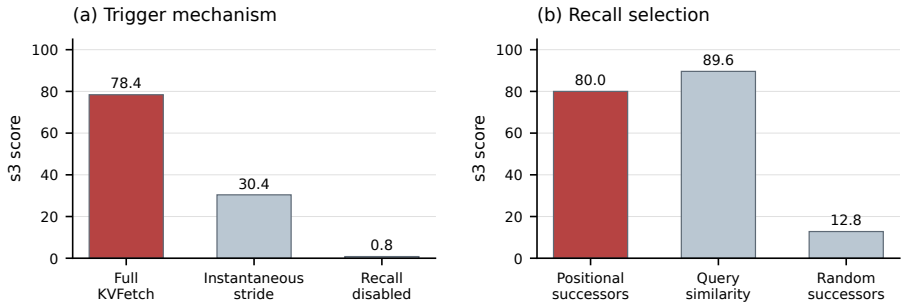

Figure 4: Ablation results for the trigger mechanism (a) and recall selection (b) on Qwen2.5-7B, RULER-16K, 20% hot retention. Panel (a) uses the default unigram admission signal; panel (b) uses the literal-predicate signal for consistency with Table 13. The two panels are not directly comparable across admission configurations; each is internally consistent.

B.3 Hit-Rate–Gain Correlation

We report two complementary diagnostics: the step-level trigger rate (fraction of decoding steps where $\mathrm{Copy}_t = 1$) and the cold-tier recall hit rate (fraction of cold-tier entries recalled at least once during generation). On LongBench, trigger rates fall in the range 0.000–0.029 and recall hit rates in 0.042–0.130; both occupy the zero-gain band.

The hit-rate analysis is computed from the full RULER-16K prediction and telemetry files. In the final run, task-level cold-tier hit rate and gain over the 22.9% iso-budget control have Pearson correlation $r = 0.924$ across 13 tasks. After removing `niah_single_3`, Spearman's ρ remains 0.818. An independent earlier full run yields Pearson $r = 0.940$ and post-removal Spearman $\rho = 0.846$. LongBench hit rates fall in the 0.042–0.130 zero-gain band, consistent with the observed +0.33 average gain and no material regression.

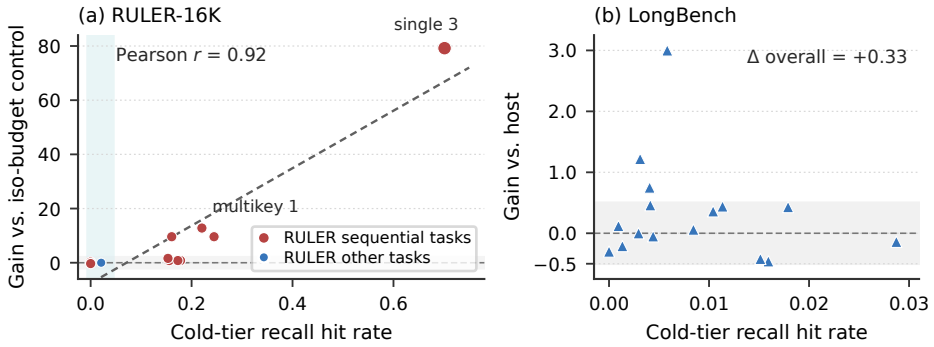

Figure 5: Hit-rate–gain prediction and verification. **(a)** RULER gains grow with cold-tier recall hit rate (Pearson $r = 0.924$ in the plotted run). **(b)** A dedicated zoom shows that all LongBench tasks occupy the low-hit, near-zero-gain regime (overall $+0.33$), rather than being visually compressed into panel (a).

B.4 Fetch Criterion

Once the temporal channel is open, the addressing mode becomes an implementation choice. Replacing the default positional-successor criterion with query–key similarity changes the four-task average by only 0.76 (from 92.96 to 93.52) and improves s3 by +9.6 (from 80.0 to 89.6); random successors degrade substantially (s3 = 12.8). The gap between the two meaningful criteria is dwarfed by the gap between having the channel and not having it. We default to positional successors because they require only an $O(k)$ position-index lookup with no additional dot products. This ablation uses the literal-predicate admission signal; as a configuration-matched variant, it leaves the conclusion unchanged under the default unigram signal.

Table 13: Fetch criterion ablation (Qwen2.5-7B, RULER-16K, 20% hot retention).

Criterion	s3	avg4
Positional successors	80.0	92.96
Query–key similarity	89.6	93.52
Random successors	12.8	–

C Hyperparameter Sensitivity

We report the sensitivity analyses referenced in Section 4.4. All sweeps below were conducted under the literal-predicate admission signal for configuration consistency with the ablations in Appendix B.4; the relative ordering is unchanged under the default unigram signal (Table 14).

Cold-tier quantization bit-width: s3 = 79.2 for 8-bit, 80.8 for 4-bit (default), and 24.0 for 2-bit. Prefetch distance k: s3 = 67.2 for $k = 4$, 80.0 for $k = 8$ (default), 89.6 for $k = 16$, and 93.6 for $k = 32$. Admission run threshold: s3 = 83.2 for threshold 4, 80.0 for threshold 6, and 80.0 for threshold 8.

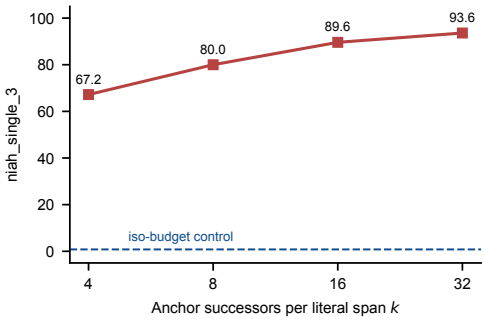

Figure 6: `niah_single_3` s3 as a function of the prefetch distance k.

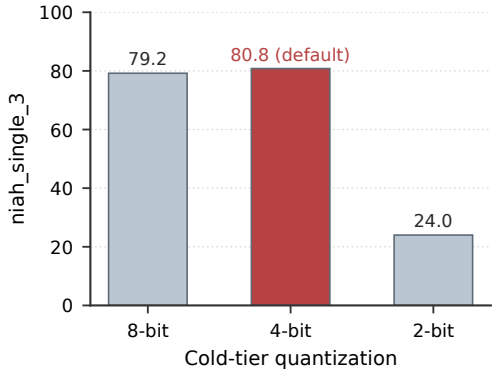

Figure 7: `niah_single_3` s3 as a function of the cold-tier quantization bit-width. 4-bit matches 8-bit; 2-bit collapses.

Table 14 compares candidate admission signal variants with all other configuration fields held identical.

D Failure Cases and Applicability

We distinguish two failure modes: locating a span but losing its continuation, and failing to locate the target after compression. The former motivates recall; the latter limits applicability.

Table 14: Candidate signal variants (all other configuration fields identical). The three instantiations yield comparable performance on their covered tasks; unigram frequency is the default because it requires no format priors.

Signal	Description	Avg	s3	cwe	fwe	vt
Unigram (default)	Low-frequency runs	64.88	78.4	54.24	80.31	98.16
Literal predicate	Hex/digit regex	65.31	80.0	54.80	80.53	98.24
Quotation span	Quote-marked segments	62.06	42.4	54.48	80.53	97.92

D.1 Failure-Sample Gallery

Figure 1 schematically illustrates the failure mode that motivates SCL: copying begins with a correct prefix but cannot continue when successor KV entries are unavailable. The ID and outputs shown are illustrative, not recorded failure samples. The diagnostic category targeted by KVFetch is a correctly found span whose successor tokens are no longer reachable under static prefill eviction.

D.2 Mistral Full-KV Probe and Trigger-Rate Telemetry

For Mistral-7B-Instruct, the 13-task compressed host average is 26.39 and KVFetch reaches 28.05. A 10% full-KV probe on `niah_single_2` reaches 100.0, whereas the compressed host reaches only 0.8 on the same task. The KVFetch trigger rate is 0.0018, compared with 0.068 on Qwen2.5 copy tasks. This supports the main-text boundary claim: KVFetch repairs continuation once a copy trajectory is present but does not perform global re-localization after the host compression has destroyed target localization.

E Implementation Details of KVFetch

E.1 Candidate Identification and Cold-Tier Ordering

The default admission signal is unigram frequency rank: tokens in the bottom-τ percentile of unigram frequency form candidate runs when they occur consecutively. For the configuration-matched ablations in Appendices B.4 and C, we use an alternative literal predicate, described below for reproducibility.

The literal predicate first strips tokenizer prefixes and whitespace. A token is considered literal if the remaining string consists entirely of `[0-9a-fA-F_-]`, or if it is a `0x`-prefixed hexadecimal string with at least two hexadecimal digits after the prefix. A run requires at least M_{run} consecutive literal tokens. Cold-tier membership is decided by the three-tier priority score

$$\sigma(p) = \begin{cases} c_{\text{cand}} + \text{rank}(p), & p \in \mathcal{S}_{\text{cand}} \\ s_a - \lambda\,(p - a), & p \in (a,\, a + W_a],\ a \in \mathcal{A} \\ \varepsilon_0\, s_p, & \text{otherwise} \end{cases} \qquad \mathcal{C} = \text{top-}n_{\text{cold}}(\sigma) \tag{5}$$

where $\mathcal{S}_{\text{cand}}$ is the union of candidate runs (extended by δ positions on each end), restricted to positions outside the hot tier, and $\mathcal{A}$ is the set of high-scoring hot-tier anchors. In Eq. (5), we set $c_{\text{cand}} = 2.0$, use position rank as the within-candidate tie breaker, $\lambda = 0.01$, anchor window $W_a = 64$, and $\varepsilon_0 = 10^{-3}$, then retain the top-n_{cold} positions.

E.2 4-bit Affine Quantization

The cold tier stores keys and values using per-head-dimension affine quantization. For each channel, we compute

$$s = \max\left((x_{\max} - x_{\min})/15,\ \varepsilon\right), \qquad z = x_{\min},$$

and store $\text{clip}(\text{round}((x - z)/s), 0, 15)$ as a 4-bit integer. Two quantized values are packed into one byte. Dequantization applies $\hat{x} = sq + z$ before a cold entry is swapped back into the hot tier.

E.3 Victim Scoring and the Protection Period

Victim slots are chosen by the composite score

$$S_{\text{vict}}(j) = 0.5\,\bar{s}_j^{(\text{prefill})} + 0.5\,\text{Norm}(\mathbf{E}_t[j]),$$

where $\mathbf{E}_t$ is an exponential moving average of hot-tier attention with decay factor $\beta = 0.9$. This EMA is used only for victim scoring; peak extraction uses the instantaneous attention at the current decoding step. Candidate victims exclude attention sinks, the prompt trailing window, and slots under protection. Newly recalled slots receive $T_{\text{prot}} = 12$ protected steps; the counter is decremented at every step and floored at zero.

E.4 Hyperparameters

The default configuration uses the following hyperparameters: minimum run length $M_{\text{run}} = 6$, literal-run extension $\delta = 4$, slack $\eta_s = 1.25$, refund ratio $\eta_r = 0.58$, cold-tier budget cap $\gamma_{\text{cold}} = 0.05$, prefetch distance $k = 8$, trigger window $W = 8$, advance threshold $n_{\text{adv}} = 2$, maximum stride $\Delta_{\max} = 4$, pointer band $= 4$, protection period $T_{\text{prot}} = 12$, sink size $= 4$, trailing window $= 64$, and EMA decay $\beta = 0.9$. In the default configuration, the pointer-band constraint dominates the back-tolerance parameter.

E.5 Complete Pseudocode

Algorithm 1 gives the complete procedure of KVFetch. Line-level comments indicate the correspondence to the equations in the main text.

Algorithm 1 KVFetch (host scorer: any ScorerPress; we use OBCache-K+SnapKV)

— Prefill —
$s \leftarrow \textsc{HostScorer}(K, V)$ ▷ scores left untouched
$\mathcal{H} \leftarrow \text{top-}(n_{\text{base}} + n_{\text{ret}})(s)$ ▷ n_{ret}: on-demand budget refund (§3.4)
▷ *The baseline ends here (frozen after top-n_{base}); everything below is added by KVFetch.*
record the position map $\mathcal{M}_{\text{pos}}$
$\mathcal{C} \leftarrow \text{top-}n_{\text{cold}}(\sigma)$, 4-bit quantized; build index $\mathcal{M}_{\text{cold}}$ ▷ three-tier admission, Eq. (5)
— Each decoding step t (per layer) —
$\mathbf{a}_t \leftarrow$ hot-tier attention ▷ same forward as the baseline; no extra compute
$p^* \leftarrow \mathcal{M}_{\text{pos}}\big[\arg\max_{j \in \mathcal{V}} \mathbf{a}_t[j]\big]$; $\Delta \leftarrow p^* - \text{ptr}$ ▷ Eq. (1)
update ptr, history $\mathbf{h}$, and Copy_t by Eqs. (2)–(3)
if $\text{Copy}_t = 1$ **then**
  $\mathcal{R} \leftarrow [\text{ptr}{+}1,\ \text{ptr}{+}k] \cap \text{dom}(\mathcal{M}_{\text{cold}}) \setminus \text{range}(\mathcal{M}_{\text{pos}})$ ▷ Eq. (4), $|\mathcal{R}| \le k$
  $\mathcal{U} \leftarrow$ the $|\mathcal{R}|$ unprotected prompt slots with the lowest composite scores
  $\Phi_{\text{swap}}(\mathcal{H}, \mathcal{C};\ \mathcal{R}, \mathcal{U})$; update $\mathcal{M}_{\text{pos}}$, $\mathcal{M}_{\text{cold}}$ ▷ one-for-one; $|\mathcal{H}|$ unchanged (Properties 1–2)
  $\text{protect}[\text{slots of } \mathcal{R}] \leftarrow T_{\text{prot}}$
end if
$\text{protect} \leftarrow \max(\text{protect} - 1,\ 0)$

The implementation additionally aggregates query heads into their corresponding GQA KV head before choosing the layer-local observation peak, computes the cold-tier quota on demand from the detected literal-run set, and writes telemetry for cold-tier hit rate, trigger rate, storage, and swap counts. The executable order is as follows: decrement the protection counter first and floor it at zero, construct the hot mask explicitly from hot indices, compute valid recall targets against $\mathcal{M}_{\text{cold}}$, and only then perform symmetric swaps.

Attention peak extraction under SDPA. PyTorch's scaled dot-product attention does not return attention weights by default. We extract the attention peak by computing $\arg\max(QK^\top)$ for the single most confident KV head per layer at each decoding step. This partial computation covers one head rather than the full multi-head attention and is included in the reported latency (Table 3). No additional full forward pass is required.

E.6 Per-Layer and Per-Head Semantics

Each layer maintains one read pointer per sample. Within a layer, the observation comes from the highest-confidence KV head after GQA aggregation. Prefetching and swapping are executed independently per KV head, so a successful trigger in one layer does not force other layers to enter copy mode.