QEVICT: RETAIN BROADLY, READ SELECTIVELY FOR LONG-CONTEXT DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context LLM inference is increasingly constrained by the memory and computation of the key–value (KV) cache. We find that KV utility exhibits two distinct timescales: long-horizon historical importance is broad and evolves gradually, while relevance to an individual query is much more concentrated. Motivated by this, we propose `QEvict`, a window-based KV-cache architecture that separates historical residency, storage representation, and current-query access. Historical windows are periodically routed under a byte budget among execution-datatype storage, compact INT2 storage, and permanent removal using accumulated attention. A write-once INT2 representation supports stable migration across storage priorities, while a compact query-conditioned representation selects only a fraction of retained low-bit windows for token-level attention. Unread windows remain represented through calibrated window-level completion. A fused mixed-precision kernel realizes selective INT2 access without materializing the full compressed cache. Across LongBench, RULER, and GSM8K with Llama and Mistral, `QEvict` improves the quality–memory trade-off over representative eviction, quantization, and mixed-precision baselines, with particularly strong gains under tight KV-cache budgets.

1 INTRODUCTION

Transformer-based large language models (LLMs) increasingly rely on long contexts for document understanding, retrieval, and multi-step reasoning (Vaswani et al., 2017; Brown et al., 2020; Touvron et al., 2023). During autoregressive decoding, however, every processed token contributes keys and values to the KV cache. Its memory grows linearly with sequence length and batch size, and for a 500B+ model at batch 512/context 2048 it reaches 3TB-3x the model's parameters. (Pope et al., 2023). Hardware-aware attention kernels (Dao, 2023) and memory-management systems (Kwon et al., 2023) improve execution efficiency, but do not remove the underlying cost of storing an increasingly long history.

KV-cache compression has consequently developed along several complementary directions. Eviction methods retain only selected states, from attention sinks and recency (Xiao et al., 2024) to accumulated heavy hitters (Zhang et al., 2023), prompt-informed selection (Li et al., 2024), and adaptive importance estimation (Feng et al., 2024; 2025; 2026). ChunkKV (Liu et al., 2026) further motivates managing coherent chunks rather than independently selected tokens, while LazyEviction (Zhang et al., 2026) accounts for changes in token importance before permanent removal. Quantization reduces the cost of resident states (Liu et al., 2024; Hooper et al., 2024), while mixed-precision methods allocate precision according to importance or quantization sensitivity (Yang et al., 2024; He et al., 2024; Saxena et al., 2024). Separately, Quest and SparQ Attention reduce computation by accessing only cache regions relevant to the current query (Tang et al., 2024; Ribar et al., 2024).

These directions expose a broader tension. The history worth preserving for future decoding need not coincide with the history worth reading for the current query. Aggressive eviction reduces memory and computation but narrows historical coverage and makes deletion irreversible. Low-bit storage preserves more history, but its computational advantage diminishes if the entire compressed cache must still be read at every step. Moreover, historical importance is neither fixed nor arbitrarily volatile: most regions evolve gradually, while some gain or lose priority as generation progresses.

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.

How can a KV cache preserve broad and evolving historical context without paying the cost of reading all retained history for every query?

Our answer is to separate long-horizon retention from current-query access. Section 3 establishes this distinction empirically. Historical utility is broad: covering 80% of cumulative importance requires 64.9% of historical windows in our diagnostic setting. At the same time, importance is persistent but not static. Selection changes gradually, and windows moved to higher-priority storage receive $2.34\times$ the future attention of the average routing candidate, compared with $0.94\times$ for windows remaining at lower priority and $0.66\times$ for windows subsequently removed. In contrast, current-query relevance is substantially more concentrated: reading only 7 of 26 retained low-bit windows captures 59.2% of their attention mass, or 93.9% of a same-cardinality hindsight selector. The unread remainder still accounts for 2.9% of total reference attention, showing that query-time non-selection should not be equated with eviction.

To quantify these behaviors, we introduce Future Missed Mass (FMM), which measures future attention assigned to historical states made inaccessible by a cache decision. We complement it with diagnostics for retention-set churn, future-attention lift after storage transitions, and query-conditioned read recall. Together, these measurements separate three questions often coupled in KV-cache compression: whether history should remain resident, how compactly it should be represented, and whether the current query requires token-level access to it.

Motivated by these findings, we introduce **`QEvict`**, a window-based KV-cache architecture that separates residency, numerical representation, and current-query access. Historical windows are periodically reassigned under a byte budget among execution-datatype storage, compact INT2 storage, and permanent removal using attention accumulated across queries. The low-bit tier preserves intermediate-priority history at substantially lower cost, allowing storage priority to evolve without forcing uncertain states directly into eviction. Each window is quantized only on its first transition to low-bit storage and thereafter reuses the same persistent INT2 record, avoiding repeated-quantization error.

Broad low-bit residency does not imply broad token-level computation. Each compressed window carries a compact representation that estimates its relevance to the current query before its token-level K/V is read. Only a fixed fraction of the low-bit tier is selected for token-level attention at each decode step. Unselected windows remain resident and contribute through a calibrated window-level completion term, which also continues to update their long-horizon importance. Query-time selection therefore changes the resolution of access rather than cache residency.

The implementation follows the same decomposition. A compact scan first selects the low-bit windows required by the current query, after which a fused mixed-precision kernel reads only those INT2 states, reconstructs K/V in registers, reapplies positional rotation, performs grouped-query attention, and incorporates unread-window completion without materializing a dequantized copy of the full low-bit cache. This realizes query-conditioned sparsity as an execution saving rather than only an algorithmic assumption.

We evaluate `QEvict` on LongBench (Bai et al., 2024), RULER (Hsieh et al., 2024), and GSM8K (Cobbe et al., 2021) using Llama-3.1-8B-Instruct (Dubey et al., 2024) and Mistral-7B-Instruct v0.2 (Jiang et al., 2023) across multiple KV-memory budgets. The evaluation spans long-context understanding, controlled retrieval, and multi-step generation, with comparisons against eviction, quantization, and mixed-precision KV-cache methods. Across these settings, `QEvict` improves the quality–memory trade-off, with particularly strong gains under restrictive cache budgets. Moreover, in memory-bound batched decoding, the fused implementation reaches 429.3 tok/s at batch size 32 and 8K decode, achieving $1.33\times$ the decode throughput of Full-KV with FlashAttention-2 while reducing peak GPU memory by 28.5%. At 16K decode, `QEvict` remains feasible at batch size 32 while all evaluated baselines run out of memory.

Our main contributions are:

- An empirical characterization of KV-cache utility across two timescales, showing that long-horizon importance is broad and evolving while current-query relevance is substantially more concentrated, supported by Future Missed Mass and complementary diagnostics for retention stability, transition utility, and query-conditioned access.

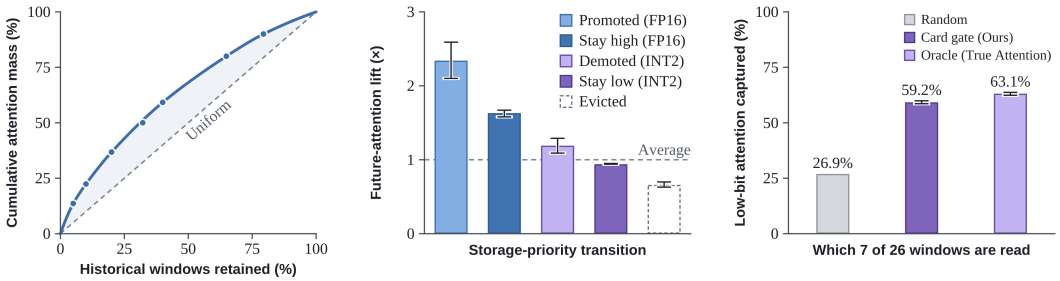
- **QEvict**, a KV-cache architecture that separates historical residency, numerical representation, and current-query access through a byte-constrained window hierarchy, migration-stable INT2 storage, selective token-level reads, and completion of retained but unread history.
- A fused mixed-precision execution path that selectively reads INT2 windows and reconstructs them in registers without materializing the full low-bit cache.
- Evaluation on LongBench, RULER, and GSM8K with Llama and Mistral, showing improved quality–memory trade-offs and higher decode throughput.

2 Related Work

The KV cache grows linearly with sequence length and can become a dominant memory cost in long-context autoregressive inference. Existing methods reduce this cost primarily through eviction, which limits the number of resident states, or quantization, which lowers their per-state footprint. More recent mixed-precision methods combine importance estimation with non-uniform numerical representations, while sparse-attention methods reduce the amount of retained history processed for each query. `QEvict` builds on these directions by separating long-horizon retention, storage representation, and current-query access.

Local Context and Query-Dependent Relevance. Natural language exhibits strong lexical and discourse coherence across neighboring positions (Grosz, 1995; Hearst, 1997; Passonneau, 1997; Barzilay, 2008; Koshorek, 2018), while contextual representations and Transformer attention capture dependencies over related spans (Ethayarajh, 2019; Clark et al., 2019; Joshi et al., 2020). These properties motivate contiguous regions as a natural cache-management unit. ChunkKV (Liu et al., 2026) similarly manages coherent chunks rather than independently selected tokens. Relevance at decode time is also query dependent. Quest (Tang et al., 2024) and SparQ Attention (Ribar et al., 2024) reduce attention computation by selectively accessing cache regions relevant to the current query. `QEvict` combines window-level retention with query-dependent access, but treats these as decisions operating at different timescales.

KV-Cache Eviction. StreamingLLM (Xiao et al., 2024) preserves attention sinks and recent tokens, while H_2O (Zhang et al., 2023) retains heavy hitters using accumulated attention. SnapKV (Li et al., 2024) estimates prompt-token importance from an observation window, AdaKV (Feng et al., 2024) adapts allocation across attention heads, and CriticalKV (Feng et al., 2025) incorporates value-related information into cache selection. DefensiveKV and Layer-DefensiveKV (Feng et al., 2026) further account for uncertainty in future attention, while LazyEviction (Zhang et al., 2026) delays removal when token importance may recur during generation. These methods improve which states remain resident, but eviction ultimately commits states outside the retained cache to removal. `QEvict`



(a) **Broad utility.** Historical importance spans a large fraction of windows.

(b) **Evolving priority.** Priority changes correspond to different future utility.

(c) **Query-time concentration.** Few windows capture most attainable low-bit attention.

Figure 1: **Empirical properties motivating `QEvict`.** Historical utility is broad and changes gradually over decoding, whereas relevance to an individual query is substantially more concentrated than the history worth retaining.

instead preserves an intermediate compressed state so that storage priority can be reconsidered as historical importance evolves.

KV-Cache Quantization. KIVI (Liu et al., 2024) introduces asymmetric low-bit KV quantization with different granularities for keys and values. KVQuant (Hooper et al., 2024) combines pre-RoPE key quantization, non-uniform representations, and outlier handling for sub-4-bit caches. These methods primarily reduce the numerical cost of storing resident KV states. `QEvict` instead uses low-bit storage as part of a broader cache-management policy, where representation is coupled to long-horizon retention but separated from query-time access.

Mixed-Precision KV Compression. Several methods allocate precision non-uniformly according to importance or quantization sensitivity. MiKV (Yang et al., 2024) retains important KV states at higher precision while preserving lower-priority states at reduced precision. ZipCache (He et al., 2024) assigns token-wise bit widths using a saliency estimate, combining higher precision for salient tokens with more aggressive quantization elsewhere. ResQ (Saxena et al., 2024) instead applies mixed precision across representation subspaces, preserving a high-variance low-rank component at higher precision while quantizing its complement more aggressively. These methods demonstrate the value of heterogeneous precision for increasing effective cache capacity. `QEvict` extends this idea by allowing storage priority itself to evolve over decoding and by separating the history worth preserving from the smaller subset that requires token-level access for the current query.

Prior work therefore provides strong mechanisms for deciding which KV states to retain, how compactly to represent them, and which states to access for a particular query. What remains less clear is how these decisions should be coupled over time. History useful over future queries may be broader than a small heavy-hitter set, its relative importance may change as generation progresses, and the subset needed by an individual query may be considerably smaller than the set worth retaining. Section 3 examines these properties empirically and motivates the separation of retention and query-time access used by `QEvict`.

3 Observations and Design Motivation

A KV cache for long-context decoding must decide which history to retain, how its storage priority should evolve, when low-priority history can be removed, and what must be read for the current query. These decisions operate at different timescales and need not be governed by the same notion of importance.

Diagnostic setup. We analyze Llama-3.1-8B-Instruct on eight WikiText-103 test articles, using 512 prompt tokens followed by 512 greedy decoding steps, eight-token historical windows, and a 20% KV-cache budget. Reference statistics are computed from the attention assigned by the same queries to the corresponding uncompressed FullKV states. Appendix I gives the complete protocol and formal definitions of all diagnostics.

Q1: Is historical utility concentrated in a few heavy hitters? No small set dominates long-horizon utility. The top 10% of historical windows account for only 22.4% of cumulative importance, while covering 80% requires 64.9% of the windows (Fig. 1a). This breadth matters for future queries. At a 32-step horizon, Future Missed Mass (Appendix I.2) is 28.9% under mixed-precision retention, compared with 45.9% and 47.9% for same-memory H2O- and StreamingLLM-style FP16 eviction. Making the retained low-bit history inaccessible increases it further to 52.3%. These results favor broad, low-cost retention over keeping only a small high-precision subset.

Q2: How does historical importance evolve over time? Historical importance is persistent but not static. Selection Churn (Appendix I.2) is only 0.044, and 99.6% of low-bit windows remain in the same storage class at the next reassignment. Yet the changes that do occur are informative. Under the future-attention lift defined in Appendix I.2, windows moved to higher-priority storage receive $2.34\times$ the candidate-average future attention, compared with $0.94\times$ for windows remaining at low priority and $0.66\times$ for windows subsequently removed (Fig. 1b). This suggests that storage priority should be revised periodically rather than fixed after a single decision.

Q3: How much retained history must be read for one query? Current-query relevance is substantially more concentrated than long-horizon historical utility. Reading seven of 26 retained low-bit windows per KV head captures 59.2% of their reference attention, compared with 26.9% under uniform selection and 63.1% for a same-cardinality hindsight selector (Appendix I.2). Thus, accessing only 26.9% of the low-bit windows recovers 93.9% of the attention attainable by hindsight (Fig. 1c). The unread windows nevertheless contribute 2.9% of total reference attention, indicating that avoiding token-level access should not make retained history equivalent to eviction.

Together, these observations reveal two distinct timescales in KV-cache management. Long-horizon retention should remain broad and change gradually as evidence accumulates, with permanent removal reserved for sufficiently persistent low utility. Current-query computation can instead focus on a much smaller subset of retained history while preserving a coarse contribution from the unread remainder. Section 4 translates this separation into the design of `QEvict`.

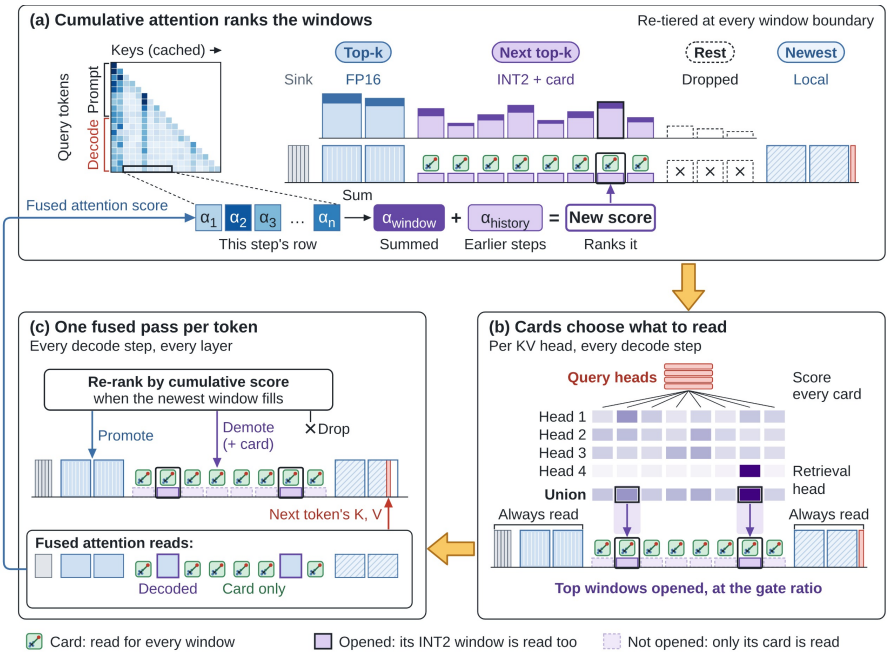


Figure 2: **Overall workflow of the `QEvict` framework.** Stage 1 partitions the input sequence into contiguous KV windows while preserving sink and recent tokens in full precision. Stage 2 accumulates attention scores over each window to produce a stable importance ranking. Stage 3 routes windows under a fixed byte budget into a full-precision tier, a recoverable INT2 tier that remains available for attention and later promotion, or permanent eviction, with dynamic promotion and demotion as importance evolves.

4 METHODOLOGY

Motivated by Section 3, `QEvict` separates two timescales in long-context decoding. Historical importance evolves slowly and therefore governs *retention*, while relevance to the current query is more concentrated and governs *access*. Accordingly, `QEvict` separates historical residency, storage representation, and query-time token-level access. Historical windows are periodically routed under a byte budget, Q-tier windows use persistent INT2 storage, and only query-relevant Q windows are read at token resolution. Unread Q windows remain resident and contribute through a compact completion term.

4.1 HIERARCHICAL KV RETENTION

Consider layer ℓ with H_q query heads, H_{kv} KV heads, and head dimension d_h. We protect an initial sink region $\mathcal{S}$ and recent region $\mathcal{L}_t$ in the model execution datatype. Older states are partitioned into

Table 1: **RULER performance at 32K context length on Llama-3.1-8B-Instruct.** We compare `QEvict` with representative KV-cache eviction methods under a matched 20% cache budget and with quantization and mixed-precision baselines at their corresponding memory footprints. Scores are string-match percentages, and KV memory is reported relative to the full-precision cache. Excluding Full-KV, the best and second-best distinct results within each comparison group are shown in bold and underlined, respectively.

Method	KV memory	Aggregation		Needle-in-a-Haystack								Question Answering		Tracking
		CWE	FWE	MK-1	MK-2	MK-3	MQ	MV	S-1	S-2	S-3	QA-1	QA-2	VT
Llama-3.1-8B-Instruct, RULER 32K														
Full-KV	100%	45.22	94.13	99.60	99.60	99.40	98.75	99.10	100.00	100.00	100.00	79.80	54.80	99.24
EVICTION: 20% KV-CACHE BUDGET														
StreamingLLM	20%	0.04	92.40	23.20	19.20	23.00	20.90	20.10	22.20	17.60	20.80	23.40	39.00	30.76
SnapKV	20%	14.56	70.40	98.40	94.40	72.80	98.80	98.00	**100.00**	97.60	48.00	**82.40**	52.00	97.92
AdaKV	20%	19.20	75.20	99.40	93.60	88.00	99.20	**99.40**	**100.00**	99.20	56.00	77.60	51.20	98.52
CriticalKV	20%	26.80	88.80	91.60	29.40	19.40	95.00	93.60	**100.00**	99.60	42.40	40.80	40.20	97.76
DefensiveKV	20%	22.94	90.00	**99.80**	86.80	97.00	98.65	97.90	**100.00**	**100.00**	97.40	68.80	45.80	98.76
Layer-Def. KV	20%	17.86	90.80	99.60	**99.40**	**99.00**	98.85	98.45	**100.00**	**100.00**	**100.00**	73.00	47.60	98.56
`QEvict`	20%	**44.02**	**93.93**	**99.80**	98.60	94.60	**99.40**	97.85	**100.00**	**100.00**	99.6	78.20	**52.20**	**99.00**
QUANTIZATION AND MIXED PRECISION														
`QEvict`	20%	44.02	**93.93**	**99.80**	**98.60**	**94.60**	**99.40**	**97.85**	**100.00**	**100**	**99.6**	78.20	52.20	**99.00**
KIVI-2b	∼20%	20.76	91.20	98.00	90.20	21.20	92.40	96.35	99.80	98.40	90.40	75.60	50.20	81.68
ZipCache-4b (70%)	∼22%	44.80	93.53	**99.80**	84.20	6.60	97.60	95.25	**100.00**	**100.00**	80.00	76.80	50.40	97.60
MiKV-2b	∼20%	27.68	85.27	91.40	46.20	0.00	84.45	83.90	**100.00**	96.76	77.60	73.00	48.80	92.12
ResQ-2b	∼20.3%	10.68	59.67	64.60	26.20	0.20	57.25	57.75	91.20	81.00	35.00	45.40	31.40	58.28
KVQuant-2b (s1%)	∼15%	64.27	68.21	61.05	52.55	80.88	76.06	75.30	61.82	62.98	89.99	**98.00**	95.00	65.86
KVQuant-3b (s1%)	∼22%	**64.38**	70.49	63.51	63.15	89.83	75.56	75.46	63.64	63.64	90.00	97.80	**96.20**	70.07

windows of W tokens and assigned to an execution-datatype tier $\mathcal{F}_t^\ell$, an INT2 tier $\mathcal{Q}_t^\ell$, or permanent eviction $\mathcal{E}_t^\ell$.

Long-horizon importance. Let $a_t^{\ell,h}(i)$ be the attention probability from query head h to cached token i. For historical window w, we accumulate received attention,

$$m_t^{\ell,h}(w) = \sum_{i \in w} a_t^{\ell,h}(i), \qquad S_t^{\ell,h}(w) = S_{t-1}^{\ell,h}(w) + m_t^{\ell,h}(w). \tag{1}$$

Prompt attention initializes S during prefill. Routing uses the FP32 head-mean score $\bar{S}_t^\ell(w) = H_q^{-1} \sum_h S_t^{\ell,h}(w)$. The first routing event occurs after prefill, before decoding, and subsequent events occur every R steps. We use $R = W$ by default.

Byte-constrained routing. After reserving sink and recent states, let B_{hist} be the historical byte budget. If C_F and C_Q are the charged costs of one F and Q window and α is the fraction assigned to Q, then

$$K_F = \left\lfloor \frac{(1-\alpha)B_{\text{hist}}}{C_F} \right\rfloor, \qquad K_Q = \left\lfloor \frac{\alpha B_{\text{hist}}}{C_Q} \right\rfloor. \tag{2}$$

C_Q includes packed INT2 K/V, quantization metadata, and the compact query-access representation described below. Appendix B.4 gives the complete physical accounting.

At routing step t, the candidate set is $\mathcal{W}_t^\ell = \mathcal{F}_{t-R}^\ell \cup \mathcal{Q}_{t-R}^\ell \cup \mathcal{L}_t^{\text{aged}}$. Ranking by $\bar{S}_t^\ell(w)$ assigns the top K_F windows to F, the next K_Q to Q, and the remainder to E, yielding F $\rightleftarrows$ Q $\rightarrow$ E. Thus, Q provides broad low-cost historical coverage while allowing storage priority to change at later routing events. Algorithm 1 gives the complete procedure.

4.2 MIGRATION-STABLE INT2 REPRESENTATION

A window is quantized only on its first entry into $\mathcal{Q}$. Following KIVI (Liu et al., 2024), keys are quantized per channel across tokens and values per token across channels. Keys are stored pre-RoPE with their original positions so that positional rotation can be reapplied after reconstruction.

Table 2: **LongBench performance on Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.2.** Each cell reports the Llama score followed by the Mistral score. We compare `QEvict` with representative KV-cache eviction methods under matched 20%, 10%, and 5% budgets, and with quantization and mixed-precision baselines at comparable memory footprints. KV memory is reported relative to the full-precision cache. Excluding Full-KV, the best and second-best distinct results for each model within every comparison group are shown in bold and underlined, respectively.

Method	KV memory	Single-Doc QA			Multi-Doc QA			Summarization			Few-Shot Learning		
		Nar.QA	Qasper	Mul.QA	Hot.QA	2Wi.QA	Musique	Gov.Re.	QMSum	M.News	TREC	Tri.QA	SAMSum
Llama-3.1-8B-Instruct / Mistral-7B-Instruct-v0.2													
Full-KV	100%	30.5/21.0	45.5/29.4	55.0/47.1	56.0/36.5	45.7/21.8	31.3/19.1	35.1/32.6	25.6/24.0	27.3/27.1	73.0/71.0	91.7/86.2	43.7/43.0
EVICTION: 20% KV-CACHE BUDGET													
StreamingLLM	20%	19.9/15.3	19.6/13.3	24.4/23.9	40.5/27.8	20.3/14.7	15.1/11.6	27.8/27.4	20.3/20.1	22.3/21.9	53.5/43.0	89.7/76.9	40.0/40.2
SnapKV	20%	20.9/16.7	28.3/17.1	29.4/31.1	45.7/30.1	30.0/16.8	22.8/11.5	27.3/26.2	21.6/21.4	22.6/22.8	50.0/51.0	**92.2**/87.2	**44.6**/42.3
AdaKV	20%	21.7/17.1	28.4/17.6	33.5/32.3	50.8/30.8	30.4/17.2	21.7/13.4	26.4/25.8	21.6/21.1	22.9/22.9	55.0/53.0	91.4/87.0	43.5/43.2
CriticalKV	20%	**29.9**/19.6	31.1/19.5	33.2/33.8	51.3/30.7	34.3/18.2	24.9/13.4	28.5/27.6	22.9/21.7	23.1/23.3	55.0/57.5	91.8/87.2	44.1/41.8
DefensiveKV	20%	27.1/19.2	40.8/23.1	47.8/43.7	55.3/34.3	38.7/21.7	27.9/17.3	29.2/28.6	23.7/21.7	23.9/23.7	66.0/67.0	91.8/86.7	43.7/43.8
Layer-Def. KV	20%	27.7/20.5	**44.3**/26.0	50.7/**47.0**	53.2/35.3	40.7/**23.3**	27.5/18.0	30.6/30.3	24.1/22.7	24.6/24.7	69.0/**71.0**	91.6/86.6	43.7/43.5
`QEvict`	20%	**29.9/22.2**	40.7/**27.1**	**55.5**/46.1	**58.0/37.7**	**48.8**/20.8	**32.6/18.3**	**33.6/30.6**	**25.8/24.5**	**26.9/26.9**	**71.5/71.0**	91.0/**87.9**	42.0/**43.9**
EVICTION: 10% KV-CACHE BUDGET													
StreamingLLM	10%	18.2/15.8	16.6/10.8	22.8/22.6	34.7/23.1	16.6/14.3	11.6/10.3	24.7/24.8	18.8/19.2	19.8/18.8	51.0/32.0	88.8/72.8	38.3/38.6
SnapKV	10%	18.9/14.9	20.8/12.0	23.6/25.2	43.0/26.4	22.6/13.5	18.2/10.8	24.1/24.3	19.6/20.2	20.4/20.9	44.5/44.5	**92.9**/87.3	43.0/40.7
AdaKV	10%	18.3/16.3	22.7/13.7	26.1/27.2	40.8/29.1	22.2/15.4	17.8/11.9	24.3/24.3	20.2/19.9	20.9/20.6	46.5/48.5	92.2/87.1	43.3/42.2
CriticalKV	10%	25.1/16.5	23.4/13.9	26.1/26.4	40.3/27.6	25.6/15.6	19.7/10.5	25.3/25.4	21.0/20.6	20.9/21.5	45.5/45.5	92.5/**87.6**	42.5/42.0
DefensiveKV	10%	20.9/17.0	26.0/14.8	34.0/32.6	48.3/31.7	30.0/16.7	19.6/11.6	25.1/25.1	22.4/20.8	21.5/21.6	54.0/53.0	91.4/87.4	43.7/43.1
Layer-Def. KV	10%	22.6/19.2	29.1/16.7	37.5/38.6	49.6/35.6	30.2/**20.6**	21.7/14.7	25.6/26.8	22.6/21.4	21.4/21.9	52.0/64.0	91.5/87.4	**44.0/43.4**
`QEvict`	10%	**28.9/20.1**	**35.0/23.3**	**50.9/42.6**	**58.2/36.5**	**47.5**/18.7	**32.6/18.0**	**32.2/29.6**	**25.4/24.0**	**24.2/25.1**	**68.0/69.0**	90.6/83.4	40.8/39.5
EVICTION: 5% KV-CACHE BUDGET													
StreamingLLM	5%	14.3/13.1	13.9/9.4	19.7/20.3	30.1/18.5	14.7/12.8	7.9/8.2	22.2/21.8	17.6/18.8	17.3/15.9	38.5/19.5	87.8/68.7	36.2/37.5
SnapKV	5%	18.3/14.4	15.7/11.4	19.8/21.8	36.3/24.0	13.4/14.7	13.2/8.6	21.6/21.8	18.3/19.2	17.8/17.8	34.0/38.5	92.0/86.6	42.1/39.5
AdaKV	5%	17.0/15.1	16.4/10.6	19.6/22.9	36.7/24.8	17.1/14.5	14.4/9.0	21.7/21.6	18.7/19.4	18.4/18.2	35.5/41.0	92.8/87.4	42.0/39.8
CriticalKV	5%	20.2/14.7	17.4/11.0	19.6/22.9	36.0/25.8	15.5/14.3	15.8/9.2	22.7/23.0	18.7/19.4	18.2/18.2	34.0/40.0	92.5/85.7	42.0/40.9
DefensiveKV	5%	20.8/16.1	18.1/11.1	21.6/24.3	39.8/25.7	19.8/15.4	14.2/10.5	23.2/23.1	19.5/20.0	18.9/19.0	43.5/44.0	**93.0**/87.1	**43.2**/40.9
Layer-Def. KV	5%	19.8/15.6	20.1/11.8	21.8/27.0	40.9/28.0	19.8/14.9	17.8/10.3	22.9/23.9	19.7/20.4	18.9/19.2	41.5/47.0	**93.0/87.6**	**43.2/42.6**
`QEvict`	5%	**27.8/20.0**	**30.9/16.8**	**47.4/34.6**	**57.3/34.1**	**45.7/17.1**	**32.5/25.1**	**26.0/27.6**	**23.7/22.7**	**23.1/19.9**	**67.5/63.5**	89.8/84.6	32.3/38.2
QUANTIZATION AND MIXED PRECISION													
`QEvict`	10%	28.9/20.1	35.0/23.3	50.9/42.6	**58.2**/36.5	47.5/18.7	**32.6**/18.0	32.2/29.6	25.4/24.0	24.2/25.1	68.0/69.0	90.6/83.4	40.8/39.5
`QEvict`	20%	**29.9**/22.2	40.7/27.1	**55.5**/46.1	58.0/**37.7**	48.8/20.8	**32.6/18.3**	**33.6**/30.6	**25.8/24.5**	**26.9/26.9**	71.5/**71.0**	91.0/**87.9**	42.0/**43.9**
KIVI-2b	∼20%	23.7/20.6	36.8/**28.7**	41.1/44.9	44.4/35.5	30.0/20.7	21.9/18.0	30.5/**32.6**	24.4/23.7	26.0/26.5	68.0/**71.0**	87.8/86.0	**44.7**/43.3
ZipCache-4b (70%)	∼22%	24.1/20.4	43.3/28.1	53.9/45.4	47.4/33.2	**50.8/20.9**	25.2/16.6	32.6/29.9	22.9/22.9	26.5/26.8	21.0/39.5	91.0/80.3	20.4/37.3
MiKV-2b	∼20%	28.2/**22.8**	38.5/25.0	51.6/44.2	57.1/36.6	47.7/19.7	30.2/17.3	30.1/30.2	24.9/24.1	26.1/26.3	72.0/**71.0**	90.5/84.4	39.9/43.1
ResQ-2b	∼20.3%	20.4/18.2	32.9/24.5	43.2/44.0	38.7/27.1	30.6/20.0	17.3/10.1	31.4/30.3	23.7/23.3	26.8/26.5	65.0/65.5	81.6/80.8	38.3/43.2
KVQuant-2b (s1%)	∼15%	22.6/16.0	36.5/23.4	47.1/41.8	43.8/29.3	34.2/17.3	23.2/11.6	31.5/27.5	22.9/22.6	25.8/26.0	65.0/10.8	86.3/68.0	41.9/25.1
KVQuant-3b (s1%)	∼22%	28.6/20.1	**44.3**/28.6	53.4/**47.1**	53.9/36.7	42.7/19.9	28.0/17.3	33.4/30.4	25.0/23.7	26.5/26.7	**72.5**/10.3	**92.0**/77.1	44.6/37.0

For quantization group x,

$$c(x) = \mathrm{clip}_{[0,3]}\left(\mathrm{round}\,\frac{x-\hat{z}}{\hat{s}}\right), \qquad \hat{x} = \hat{z} + \hat{s}\,c(x), \tag{3}$$

where $\hat{s}$ and $\hat{z}$ are the scale and offset after encoding into the persistent quantization grid. Codes are fitted against these stored parameters, four INT2 codes are packed per byte, and the grids are themselves compressed. Appendix J.1 gives the exact format.

The resulting INT2 record is write-once and persists across storage transitions. A Q→F transition reconstructs K/V into the model execution datatype and reapplies RoPE. If the window later returns to Q, its existing INT2 record is reactivated rather than regenerated. F therefore denotes the execution datatype, not recovery of the original pre-quantized values, and repeated F–Q transitions do not accumulate repeated-quantization error.

4.3 QUERY-CONDITIONED ACCESS TO COMPRESSED HISTORY

Broad Q-tier retention need not imply reading all Q windows for every query. On first entry into Q, each window therefore receives a compact write-once representation built from its post-RoPE keys. For one KV head, let $\mathbf{k}_{w,i}, \mathbf{v}_{w,i} \in \mathbb{R}^{d_h}$ be the key and value of token i in window w. We store the key and value centroids $\boldsymbol{\mu}_w = W^{-1}\sum_i \mathbf{k}_{w,i}$ and $\boldsymbol{\nu}_w = W^{-1}\sum_i \mathbf{v}_{w,i}$, together with one direction of within-window key variation. With $j^\star = \arg\max_i \|\mathbf{k}_{w,i} - \boldsymbol{\mu}_w\|_2$,

$$\mathbf{u}_w = \frac{\mathbf{k}_{w,j^\star} - \boldsymbol{\mu}_w}{\|\mathbf{k}_{w,j^\star} - \boldsymbol{\mu}_w\|_2}, \qquad \xi_{w,i} = (\mathbf{k}_{w,i} - \boldsymbol{\mu}_w)^\top \mathbf{u}_w, \tag{4}$$

giving $\widehat{\mathbf{k}}_{w,i} = \boldsymbol{\mu}_w + \xi_{w,i}\mathbf{u}_w$. A query evaluates a window using two d_h-dimensional dot products and W scalar operations. The default encoding stores $\boldsymbol{\mu}_w, \boldsymbol{\nu}_w$ as anchored INT4 residuals and $\mathbf{u}_w, \boldsymbol{\xi}_w$ in INT8. Appendix J.2 gives the representation details.

For query head h, with $\gamma = d_h^{-1/2}$, the predicted log mass of window w is

$$\widehat{\ell}_{t,h}(w) = \mathrm{LSE}_{i=1}^{W}\left[\gamma(\mathbf{q}_t^{\ell,h})^\top \widehat{\mathbf{k}}_{w,i}\right], \tag{5}$$

where $\mathrm{LSE}(x_1, \ldots, x_n) = \log \sum_i \exp(x_i)$.

For GQA, let $\mathcal{G}(k)$ be the query heads sharing KV head k. We normalize each query head as $\pi_{t,h}(w) = \widehat{\ell}_{t,h}(w) - \mathrm{LSE}_{u\in\mathcal{Q}_t^\ell}\widehat{\ell}_{t,h}(u)$ and define the group relevance $g_{t,k}(w) = \max_{h\in\mathcal{G}(k)} \pi_{t,h}(w)$. If $N_Q = |\mathcal{Q}_t^\ell|$ and $\rho \in (0, 1]$ is the read ratio, each KV head selects the top $K_R = \max\{1, \lceil \rho N_Q \rceil\}$ windows. We denote the selected and unread sets by $\mathcal{R}_{t,k}$ and $\mathcal{U}_{t,k}$, respectively, and query head h uses those associated with KV head $k(h)$. Selection therefore changes access resolution, not residency. Algorithm 2 summarizes the procedure.

4.4 Completion of Unread Q Windows

For query head h, token-level attention is computed over $\mathcal{S}$, $\mathcal{L}_t$, $\mathcal{F}_t^\ell$, and $\mathcal{R}_{t,k(h)}$. Let $\bar{\mathbf{o}}_{t,h}$ be the resulting output and $\bar{m}_{t,h}(w)$ the restricted-softmax mass of selected Q window w. With $\mathcal{R}_{t,h}^+ = \{w \in \mathcal{R}_{t,k(h)} : \bar{m}_{t,h}(w) > 0\}$, the selected windows calibrate predicted log masses through

$$\delta_{t,h} = \frac{1}{|\mathcal{R}_{t,h}^+|} \sum_{w\in\mathcal{R}_{t,h}^+} \left[\log \bar{m}_{t,h}(w) - \widehat{\ell}_{t,h}(w)\right]. \tag{6}$$

For unread $u \in \mathcal{U}_{t,k(h)}$, define $\widetilde{m}_{t,h}(u) = \exp(\widehat{\ell}_{t,h}(u) + \delta_{t,h})$ and $Z_{t,h} = 1 + \sum_{u\in\mathcal{U}_{t,k(h)}} \widetilde{m}_{t,h}(u)$. The completed output is

$$\mathbf{o}_{t,h} = \frac{\bar{\mathbf{o}}_{t,h} + \sum_{u\in\mathcal{U}_{t,k(h)}} \widetilde{m}_{t,h}(u)\boldsymbol{\nu}_u}{Z_{t,h}}. \tag{7}$$

The same completed distribution updates long-horizon scores: selected restricted masses are divided by $Z_{t,h}$, while unread window u contributes $\widetilde{m}_{t,h}(u)/Z_{t,h}$. Thus, unread Q windows continue to accumulate routing evidence without requiring token-level K/V access. Appendix J.3 gives the derivation.

System realization. The implementation avoids materializing either the full attention matrix or a dequantized Q tier. Prefill accumulates received-attention statistics through a key-major reduction over FlashAttention-style softmax normalizers. During decoding, compact Q representations are scanned first, after which a fused mixed-precision kernel reads only selected INT2 windows, dequantizes them in registers, reapplies RoPE, performs GQA, and applies unread-window completion in the epilogue. No expanded Q tensor is written to HBM. Appendix J.4 gives the kernel details, and Appendix J.5 specifies the numerical and positional conventions.

5 Experiments

Setup. We evaluate `QEvict` on Llama-3.1-8B-Instruct (Dubey et al., 2024) and Mistral-7B-Instruct-v0.2 (Jiang et al., 2023) using LongBench (Bai et al., 2024), RULER (Hsieh et al., 2024), and GSM8K (Cobbe et al., 2021). We compare against representative eviction, quantization, and mixed-precision KV-cache methods. Eviction methods use matched measured-memory budgets, while quantization and mixed-precision methods are evaluated at their measured operating points. Unless otherwise stated, `QEvict` uses $W = 8$, $R = 8$, INT2 Q-tier storage, $\alpha = 0.70$, and $\rho = 0.25$. For RULER at 32K context, we use $W = R = 32$. Benchmark-specific protected regions, decoding protocols, baseline configurations, and byte-level memory accounting are provided in Appendix B. Code repository is available here.

Long-context understanding. Table 2 reports 12 LongBench tasks at 5%, 10%, and 20% KV-memory budgets. Against matched-memory eviction, QEvict achieves the highest 12-task average at every budget on both models. At 20%, it obtains 46.4 on Llama and 38.1 on Mistral, compared with 44.0 and 37.4 for the strongest matched eviction baseline. The gap grows under tighter memory: at 5%, QEvict reaches 42.0 and 33.7, exceeding the strongest matched eviction baseline by 10.4 and 4.7 points, respectively. The same trend holds against quantization and mixed-precision methods. At 20% memory, QEvict obtains 46.4 on Llama and 38.1 on Mistral, compared with the strongest alternative averages of 45.4 and 37.6. Even at 10%, it retains averages of 44.5 and 35.8. These results show that broad low-bit retention remains effective beyond comparisons with FP16 eviction alone.

Long-range retrieval. We evaluate RULER at 32K context on Llama-3.1-8B-Instruct with a 20% KV-cache budget. As shown in Table 1, QEvict reaches 44.0 on CWE, close to the 45.2 Full-KV result and well above the matched-memory eviction alternatives. It also obtains 93.93 on FWE, 99.8 on MK-1, 98.6 on MK-2, 99.4 on MQ, 100.0 on S-1, and 52.2 on QA-2. Performance varies across RULER tasks, indicating that broader historical retention is particularly useful when relevant evidence is distributed across the context rather than concentrated in a small persistent subset.

Reasoning under compression. GSM8K evaluates cache compression during autoregressive multi-step generation. Complete results are reported in Tables 6 and 7 in Appendix F. At 20% memory, QEvict reaches 74.6% accuracy on Llama and 36.9% on Mistral, compared with 31.2% and 20.1% for the strongest matched-memory eviction baselines. Quantization methods remain strong on this benchmark at similar memory, reaching 80.0% on Llama and 41.8% on Mistral, illustrating a complementary trade-off between broad adaptive retention and uniformly quantized storage.

Ablations. Appendix G varies historical window size W, Q-tier budget fraction α, and query-time read ratio ρ. Performance is stable around the default configuration, with $W = 8$, $\alpha = 0.70$, and $\rho = 0.25$ attaining the highest four-task average in their respective sweeps. In particular, increasing ρ beyond 0.25 provides no consistent gain, supporting the separation between broad historical retention and selective token-level access.

5.1 Efficiency

The systems benefit of query-conditioned access is most pronounced in memory-bound batched decoding. On Llama-3.1-8B-Instruct with a 256-token prefill, batch size 32, and 8K decode, QEvict reaches 429.3 tok/s, corresponding to 3.98× the decode throughput of Full-KV (eager), 1.33× that of Full-KV with FlashAttention-2, 3.97× that of MiKV, and 3.37× that of ResQ.

Table 3: **Batch-32 efficiency at 256 prefill and 8K decode.**

Method	TTFT (s)	TPOT (s)	Decode (tok/s)	Peak (GiB)
Full-KV (eager)	0.750	0.2962	108.0	52.2
Full-KV (flash)	**0.690**	0.0991	322.8	48.5
MiKV	1.020	0.2956	108.2	52.4
ResQ	1.053	0.2512	127.4	48.5
QEvict	1.106	**0.0745**	**429.3**	**34.7**

It also achieves the lowest TPOT, 0.0745 s, while reducing peak memory to 34.7 GiB, 28.5% below Full-KV (flash) and 33.5% below Full-KV (eager). The additional routing work primarily affects TTFT, which is 1.106 s compared with 0.690 s for Full-KV (flash).

At 16K decode and batch size 32, both Full-KV backends, MiKV, and ResQ run out of memory, whereas QEvict remains feasible at 214.3 tok/s with 54.8 GiB peak memory. At batch size 1, the compact Q-tier scan and selection overhead are not fully amortized, so QEvict does not improve latency over optimized Full-KV. Appendix H describes the execution path and measurement protocol, while Appendix H.1 reports the complete measurements.

6 Conclusion

We introduced QEvict, which separates KV-cache retention, representation, and query-time access. Across LongBench, RULER, and GSM8K, it improves quality–memory trade-offs while reducing memory and increasing batched decode throughput. These results support broad low-bit retention with selective query-conditioned access for efficient long-context decoding.

AI USE STATEMENT

We used generative AI tools, including large language models (LLMs), for language-editing assistance such as paraphrasing, grammar correction, improving clarity, and checking notation and terminology consistency across the manuscript. All AI-assisted edits were reviewed by the authors for correctness, consistency, and fidelity to the intended technical content. The authors take full responsibility for the final content of this work, including all text, claims, results, and artifact

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*, 2024.

Barzilay. Modeling local coherence: An entity-based approach. *Computational Linguistics*, 34(1):1–34, 2008. doi: 10.1162/coli.2008.34.1.1. URL https://aclanthology.org/J08-1001/.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of BERT's attention. In Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes (eds.), *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 276–286, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4828. URL https://aclanthology.org/W19-4828/.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Kawin Ethayarajh. How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 55–65, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1006. URL https://aclanthology.org/D19-1006/.

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, 2024.

Yuan Feng, Junlin Lv, Haoyu Guo, Yukun Cao, Xike Xie, and S. Kevin Zhou. Identify critical kv cache in llm inference from an output perturbation perspective. *OpenReview preprint*, 2025.

Yuan Feng, Haoyu Guo, Junlin Lv, S Kevin Zhou, and Xike Xie. Defensivekv: Taming the fragility of kv cache eviction in llm inference. In *The Fourteenth International Conference on Learning Representations, 2026a. URL https://openreview. net/forum*, 2026.

Grosz. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–225, 1995. URL https://aclanthology.org/J95-2003/.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. Zipcache: Accurate and efficient kv cache quantization with salient token identification. *Advances in Neural Information Processing Systems*, 37:68287–68307, 2024.

Marti A. Hearst. Text tiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997. URL https://aclanthology.org/J97-1003/.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. In *NeurIPS*, 2024.

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.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020. doi: 10.1162/tacl_a_00300. URL https://aclanthology.org/2020.tacl-1.5/.

Koshorek. Text segmentation as a supervised learning task. In Marilyn Walker, Heng Ji, and Amanda Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 469–473, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2075. URL https://aclanthology.org/N18-2075/.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *SOSP*, 2023.

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 *NeurIPS*, 2024.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Bo Li, Xuming Hu, and Xiaowen Chu. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *Advances in Neural Information Processing Systems*, 38:28728–28778, 2026.

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, 2024.

Passonneau. Discourse segmentation by human and automated means. *Computational Linguistics*, 23(1):103–139, 1997. URL https://aclanthology.org/J97-1005/.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ attention: Bandwidth-efficient LLM inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42558–42583. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/ribar24a.html.

Utkarsh Saxena, Sayeh Sharify, Kaushik Roy, and Xin Wang. Resq: Mixed-precision quantization of large language models with low-rank residuals. *arXiv preprint arXiv:2412.14363*, 2024.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*, 2024.

Haoyue Zhang, Hualei Zhang, Xiaosong Ma, Jie Zhang, and Song Guo. LazyEviction: Lagged KV eviction with attention pattern observation for efficient long reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36335–36352, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1683. URL https://aclanthology.org/2026.acl-long.1683/.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *NeurIPS*, 2023.

A ALGORITHM

QEvict operates at two time scales. Periodic routing updates the representation of historical windows using their cumulative received-attention scores. At every decoding step, a query-conditioned read policy selects which windows in the low-bit tier are evaluated at token resolution. Algorithms 1 and 2 summarize these two operations.

For each quantized window w, the persistent record Λ_w contains its packed INT2 K/V representation, quantization metadata, original token positions, and compact read representation. This record is created on the first transition into $\mathcal{Q}$ and remains unchanged until the window is permanently evicted.

Algorithm 1 QEvict Periodic Historical-Window Routing

Require: Previous tiers $\mathcal{F}^{\ell}_{t-R}$ and $\mathcal{Q}^{\ell}_{t-R}$
Require: Aged local windows $\mathcal{L}^{\text{aged}}_{t}$
Require: Retention scores $\bar{S}^{\ell}_{t}(w)$
Require: Capacities K_F, K_Q and persistent records Λ
$\mathcal{W}^{\ell}_{t} \leftarrow \mathcal{F}^{\ell}_{t-R} \cup \mathcal{Q}^{\ell}_{t-R} \cup \mathcal{L}^{\text{aged}}_{t}$
$\mathcal{F}^{\ell}_{t} \leftarrow \text{TopK}_{K_F}\left(\mathcal{W}^{\ell}_{t}, \bar{S}^{\ell}_{t}\right)$
$\mathcal{Q}^{\ell}_{t} \leftarrow \text{TopK}_{K_Q}\left(\mathcal{W}^{\ell}_{t} \setminus \mathcal{F}^{\ell}_{t}, \bar{S}^{\ell}_{t}\right)$
$\mathcal{E}^{\ell}_{t} \leftarrow \mathcal{W}^{\ell}_{t} \setminus \left(\mathcal{F}^{\ell}_{t} \cup \mathcal{Q}^{\ell}_{t}\right)$
for $w \in \mathcal{F}^{\ell}_{t}$ **do**
  if $w \in \mathcal{Q}^{\ell}_{t-R}$ **then**
    $(K_w, V_w) \leftarrow \text{Dequantize}(\Lambda_w)$
    reapply RoPE to K_w at the original positions
    mark Λ_w dormant
  end if
end for
for $w \in \mathcal{Q}^{\ell}_{t}$ **do**
  if w has no record in Λ **then**
    $\Lambda_w \leftarrow \text{QuantizeAndEncode}(w)$
    construct and store the read representation of w
  else if Λ_w is dormant **then**
    reactivate Λ_w
  end if
end for
for $w \in \mathcal{E}^{\ell}_{t}$ **do**
  release native K/V state associated with w
  delete Λ_w if present
end for
return $(\mathcal{F}^{\ell}_{t}, \mathcal{Q}^{\ell}_{t}, \mathcal{E}^{\ell}_{t}, \Lambda)$

B COMPLETE EXPERIMENTAL DETAILS

This section specifies the models, inference protocol, baselines, comparison regimes, memory accounting, and default QEvict configurations used throughout the evaluation.

B.1 MODELS AND INFERENCE PROTOCOL

We evaluate QEvict on Llama-3.1-8B-Instruct (Dubey et al., 2024) and Mistral-7B-Instruct-v0.2 (Jiang et al., 2023). QEvict is training-free and modifies only KV-cache management during inference. Within each benchmark, all methods use the same model checkpoint, tokenizer, prompt template, input and generation limits, and answer post-processing. Unless stated otherwise, decoding is greedy.

Algorithm 2 `QEvict` Query-Conditioned Q-Tier Access

Require: Query vectors $\{\mathbf{q}_t^{\ell,h}\}_{h=1}^{H_q}$
Require: Current tiers $\mathcal{F}_t^\ell$ and $\mathcal{Q}_t^\ell$
Require: Persistent records Λ
Require: Read ratio ρ
for each KV head k **do**
  for $w \in \mathcal{Q}_t^\ell$ **do**
    for $h \in \mathcal{G}(k)$ **do**
      $\widehat{\ell}_{t,h}(w) \leftarrow \mathrm{CardLogMass}(\mathbf{q}_t^{\ell,h}, \Lambda_w)$
    end for
  end for
  for $h \in \mathcal{G}(k)$ **do**
    $\pi_{t,h}(w) \leftarrow \widehat{\ell}_{t,h}(w) - \mathrm{LSE}_{u \in \mathcal{Q}_t^\ell} \widehat{\ell}_{t,h}(u)$
  end for
  $g_{t,k}(w) \leftarrow \max_{h \in \mathcal{G}(k)} \pi_{t,h}(w)$
  $K_R \leftarrow \max\left(1, \lceil \rho |\mathcal{Q}_t^\ell| \rceil\right)$
  $\mathcal{R}_{t,k} \leftarrow \mathrm{TopK}_{K_R}\left(\mathcal{Q}_t^\ell, g_{t,k}\right)$
  $\mathcal{U}_{t,k} \leftarrow \mathcal{Q}_t^\ell \setminus \mathcal{R}_{t,k}$
end for
compute exact attention over $\mathcal{S}$, $\mathcal{L}_t$, $\mathcal{F}_t^\ell$, and the selected Q windows $\{\mathcal{R}_{t,k}\}$
for each query head h **do**
  let $k(h)$ be the KV head shared by h
  $\mathcal{R}_{t,h} \leftarrow \mathcal{R}_{t,k(h)}$ and $\mathcal{U}_{t,h} \leftarrow \mathcal{U}_{t,k(h)}$
  obtain restricted output $\bar{\mathbf{o}}_{t,h}$ and selected-window masses $\bar{m}_{t,h}(w)$
  $\delta_{t,h} \leftarrow \dfrac{1}{|\mathcal{R}_{t,h}^+|} \displaystyle\sum_{w \in \mathcal{R}_{t,h}^+} \left[\log \bar{m}_{t,h}(w) - \widehat{\ell}_{t,h}(w)\right]$
  for $u \in \mathcal{U}_{t,h}$ **do**
    $\widetilde{m}_{t,h}(u) \leftarrow \exp\left(\widehat{\ell}_{t,h}(u) + \delta_{t,h}\right)$
  end for
  $Z_{t,h} \leftarrow 1 + \displaystyle\sum_{u \in \mathcal{U}_{t,h}} \widetilde{m}_{t,h}(u)$
  $\mathbf{o}_{t,h} \leftarrow \dfrac{\bar{\mathbf{o}}_{t,h} + \displaystyle\sum_{u \in \mathcal{U}_{t,h}} \widetilde{m}_{t,h}(u) \boldsymbol{\nu}_u}{Z_{t,h}}$
  update $S_t^{\ell,h}$ using the completed window masses
end for
return $\{\mathbf{o}_{t,h}\}_{h=1}^{H_q}$ and updated cumulative scores

We consider three complementary settings. LongBench (Bai et al., 2024) measures long-context understanding across document QA, summarization, and few-shot tasks. RULER (Hsieh et al., 2024) evaluates controlled retrieval, aggregation, question answering, and tracking at 32K context length. GSM8K (Cobbe et al., 2021) evaluates multi-step generation under KV-cache compression. LongBench and GSM8K are evaluated on both Llama and Mistral, while the 32K RULER evaluation uses Llama-3.1-8B-Instruct.

B.2 Baselines

We compare against three families of KV-cache compression methods reviewed in Section 2: eviction, quantization, and mixed-precision caching.

Eviction baselines. We evaluate StreamingLLM (Xiao et al., 2024), SnapKV (Li et al., 2024), AdaKV (Feng et al., 2024), CriticalKV (Feng et al., 2025), DefensiveKV, and Layer-DefensiveKV (Feng et al., 2026). These methods span sink- and recency-based retention, prompt-

time observation, head-adaptive allocation, value-aware importance estimation, and robustness to uncertain future attention. Eviction baselines permanently remove states outside their retained cache.

Quantization baselines. We compare against KIVI (Liu et al., 2024) and KVQuant (Hooper et al., 2024), which reduce the numerical storage cost of resident KV states using low-bit representations.

Mixed-precision baselines. We additionally compare against ZipCache (He et al., 2024), MiKV (Yang et al., 2024), and ResQ (Saxena et al., 2024). These methods allocate precision non-uniformly according to token importance or representation sensitivity. Together with the quantization baselines, they test whether `QEvict` benefits beyond simply storing the cache at reduced or heterogeneous precision.

Uncompressed reference. Full-KV retains the complete key and value cache in the model execution datatype and serves as the uncompressed reference.

B.3 Comparison Regimes

We separate matched-budget eviction comparisons from quantization and mixed-precision comparisons because nominal bit widths and cache fractions do not imply identical physical memory footprints.

Matched-memory eviction. For method $\mathcal{A}$, let $\rho_{\mathcal{A}} = M_{\mathcal{A}}/M_{\mathrm{FullKV}}$ denote its measured persistent KV-memory ratio. LongBench evaluates eviction methods at $\rho_{\mathcal{A}} \in \{0.05, 0.10, 0.20\}$. RULER 32K uses a matched 20% KV-cache budget. GSM8K evaluates the quality–memory trade-off over $\rho_{\mathcal{A}} \in \{0.10, 0.20, 0.40, 0.60, 0.80, 1.0\}$. For these comparisons, each method is configured to the same measured persistent KV-cache budget.

Quantization and mixed precision. Quantization and mixed-precision methods are compared at their corresponding measured performance–memory operating points rather than equating nominal precision with physical cache size. This accounts for method-specific quantization metadata, retained higher-precision states, and other persistent cache allocations. Tables report the resulting KV-memory ratio relative to Full-KV alongside task performance.

B.4 Memory Accounting

For a model with L layers, H_{kv} KV heads, head dimension d_h, sequence length S, and c_f bytes per element, the uncompressed KV-cache footprint is

$$M_{\mathrm{FullKV}} = 2LH_{kv}d_hSc_f, \tag{8}$$

where the factor of two accounts for keys and values.

We include all persistent cache-related allocations when reporting `QEvict` memory. These comprise the protected regions, native-precision historical windows, the allocated low-bit backing store, positional information, and persistent routing metadata. The total persistent cache allocation is

$$M_{\texttt{QEvict}} = M_{\mathrm{sink}} + M_{\mathrm{recent}} + M_{\mathrm{F}} + M_{\text{Q-store}} + M_{\mathrm{position}} + M_{\mathrm{ledger}}. \tag{9}$$

Here, M_{F} contains the K/V tensors currently resident in the native-precision historical tier. $M_{\text{Q-store}}$ is the complete allocated storage for low-bit window records and includes packed INT2 K/V codes, compressed scale and offset grids, and the compact query-read representations described in Appendix J.2. Importantly, this allocation includes both active Q windows and dormant low-bit records retained for windows that have moved from Q to F.

Let C_{Q} denote the complete per-window low-bit storage cost, including INT2 K/V codes, quantization metadata, and the read representation. The implementation allocates a bounded low-bit slot table with

$$N_{\mathrm{slot}} = K_F + K_Q + \Delta, \tag{10}$$

where Δ is a small fixed implementation margin. The corresponding low-bit backing allocation is

$$M_{\text{Q-store}} = L\, N_{\mathrm{slot}}\, C_{\mathrm{Q}}, \tag{11}$$

up to batch replication when caches are maintained independently for each sequence.

M_{position} contains the absolute positions required to reconstruct pre-RoPE keys at their original locations. M_{ledger} contains persistent slot identities, activity state, and migration bookkeeping. These terms are counted independently of whether a low-bit slot is currently active or dormant.

For reference, one native-precision window per KV head costs

$$C_F^{\text{head}} = 2Wd_h c_f. \tag{12}$$

The packed INT2 K/V codes require $Wd_h/2$ bytes per KV head. Let $G(n)$ denote the storage required by the compressed affine scale and offset grids over an axis of length n. With the read representation cost $C_{\text{read}}^{\text{head}}$, the complete Q-window cost is

$$C_Q^{\text{head}} = \frac{Wd_h}{2} + G(d_h) + G(W) + C_{\text{read}}^{\text{head}}. \tag{13}$$

For the default $W = 8$ and $d_h = 128$ configuration, the INT2 K/V representation and its quantization grids occupy 804 bytes per KV head, while the query-read representation occupies 272 bytes, giving

$$C_Q^{\text{head}} = 1076 \text{ bytes}. \tag{14}$$

The corresponding FP16 K/V window occupies 4096 bytes per KV head, giving a $3.81\times$ difference at the window level before positional and ledger metadata.

We distinguish the nominal active tier allocation used to determine K_F and K_Q from the physical persistent allocation in Eq. (9). The latter includes the dormant backing records required by bidirectional F–Q migration and is the quantity used for physical cache-memory comparisons.

Temporary tensors used by attention, routing, sorting, and kernel execution are not counted as persistent KV-cache storage. Their contribution is instead captured by measured peak GPU memory.

Finally, the first cache compaction occurs after prompt prefill. The complete prompt KV cache therefore exists in the model execution datatype during prefill before the hierarchical representation is formed. We report steady-state decode-cache memory separately from end-to-end peak GPU memory when evaluating memory capacity.

B.5 QEvict Configuration

Unless otherwise stated, `QEvict` uses contiguous historical windows of $W = 8$ tokens, routing interval $R = 8$, five protected sink tokens, and 32 recent tokens. Historical states outside the protected regions are allocated between an execution-datatype tier and an INT2 tier. After reserving the sink and recent regions, $\alpha = 0.70$ of the active historical byte budget is assigned to the INT2 tier and the remainder to the execution-datatype tier.

Query-conditioned access uses read ratio $\rho = 0.25$. At each decode step, each KV head therefore selects $K_R = \max\{1, \lceil \rho N_Q \rceil\}$ of its N_Q retained INT2 windows for token-level access. The remaining Q windows stay resident and contribute through the completion mechanism described in Section 4.4.

For the 32K RULER evaluation, we use $W = 32$ and $R = 32$ while retaining the same byte-budgeted hierarchy and query-conditioned access mechanism. LongBench and GSM8K use the default $W = R = 8$ configuration. Sensitivity to the routing interval, tier allocation, compressed precision, and query-read ratio is reported in Appendix G.

C LongBench Evaluation: Task Families and Metrics

We evaluate 12 LongBench tasks spanning the four families summarized in Table 4. For LongBench, we retain 128 recent tokens in the local cache.

We use the official metric for each dataset and report task-level scores. All baselines, configurations, memory-accounting rules, and comparison protocols follow Appendix B, particularly Appendix B.3. LongBench results for Llama-3.1-8B-Instruct (Dubey et al., 2024) and Mistral-7B-Instruct-v0.2 (Jiang et al., 2023) are reported in Table 2.

Table 4: **LongBench task families used in our evaluation.**

Task Family	Datasets	Evaluation Focus
Single-Document QA	NarrativeQA, Qasper, MultiFieldQA-en	Retrieving evidence from one long document.
Multi-Document QA	HotpotQA, 2WikiMultihopQA, MuSiQue	Retrieving and combining evidence across documents.
Summarization	GovReport, QMSum, MultiNews	Preserving information distributed throughout the input.
Few-Shot Learning	TREC, TriviaQA, SAMSum	Retaining demonstrations and task structure.

D LongBench Performance–Memory Pareto Analysis

Fixed-budget comparisons capture performance at only a small number of operating points and may obscure the overall trade-off between task quality and KV-cache memory. We therefore analyze the performance–memory Pareto frontier for each LongBench task family, model, and comparison class.

Let (ρ_i, S_i) denote an operating point, where ρ_i is the measured KV-memory ratio and S_i is the corresponding task-family score. Operating point i dominates j when

$$\rho_i \leq \rho_j, \qquad S_i \geq S_j, \tag{15}$$

with at least one strict inequality. The Pareto frontier is the set of non-dominated operating points. Points closer to the upper-left corner achieve higher task performance at lower KV-memory usage.

We construct separate comparisons against eviction baselines and quantization/mixed-precision baselines for Single-Document QA, Multi-Document QA, Summarization, and Few-Shot Learning. This yields eight panels per model and 16 panels across Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.2. Figures 3 and 4 report the corresponding Pareto plots. Within each figure, the left column compares `QEvict` with eviction methods, while the right column compares it with quantization and mixed-precision methods. Horizontal dashed lines denote the performance of the uncompressed FullKV reference.

These plots complement the fixed-budget results in Table 2 by showing whether performance advantages persist across multiple memory regimes rather than at a single selected budget. They also distinguish methods that improve task quality at similar memory usage from those that reduce memory while maintaining comparable performance.

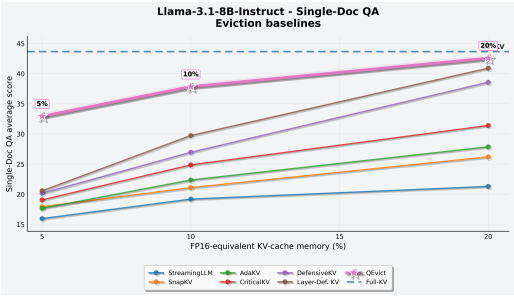
E RULER Evaluation Details

RULER (Hsieh et al., 2024) provides a controlled evaluation of long-context retrieval, aggregation, and reasoning by varying the location, multiplicity, and dependency structure of relevant evidence. We evaluate Llama-3.1-8B-Instruct (Dubey et al., 2024) at a 32K context length under a 20% KV-cache budget. For this longer context, we use historical windows of $W = 32$ tokens, route every $R = 32$ decode steps, and protect a recent region of 64 tokens. All other configurations, memory-accounting rules, and comparison protocols follow Appendix B. Results are reported in Table 1.

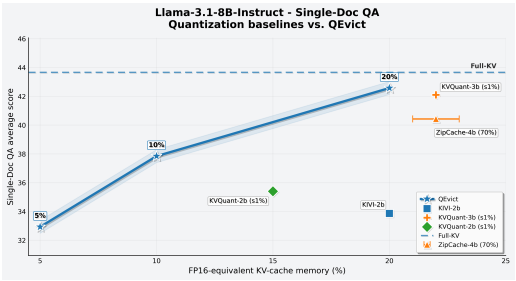
RULER contains 13 tasks spanning four complementary capabilities, summarized in Table 5. Needle-in-a-haystack tasks test whether sparse evidence remains accessible when its relevance is revealed by a later query. Aggregation tasks require information distributed throughout the context to remain available, while variable tracking tests preservation of dependent intermediate states. The question-answering tasks provide a more naturalistic evaluation of long-range evidence retrieval and combination.

F GSM8K Evaluation Protocol

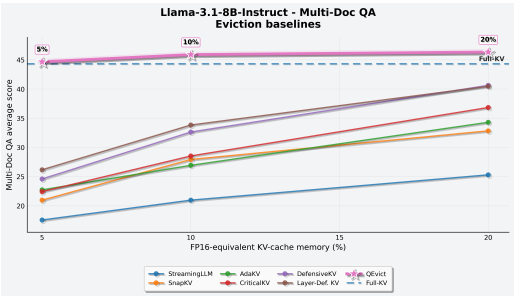
GSM8K (Cobbe et al., 2021) evaluates KV-cache compression during autoregressive multi-step reasoning, complementing long-context benchmarks that primarily test retrieval from the input se-



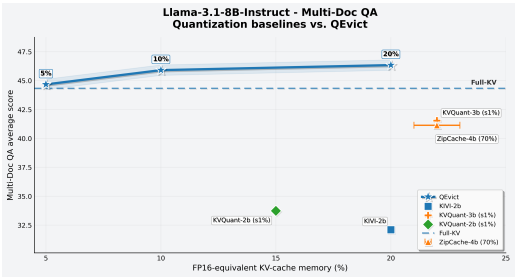
(a) Single-Document QA: eviction.



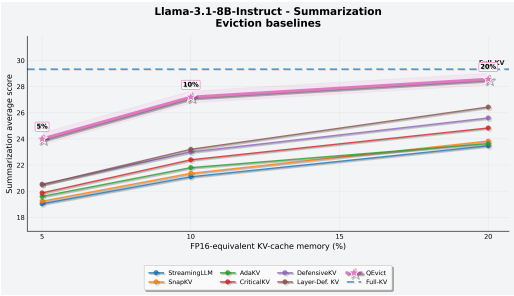
(b) Single-Document QA: quantization / mixed precision.



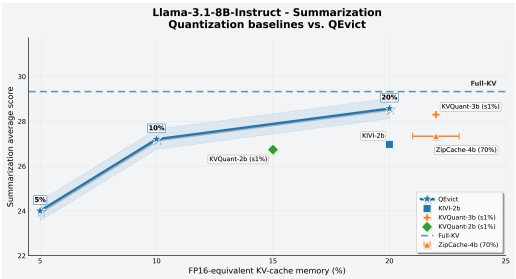
(c) Multi-Document QA: eviction.



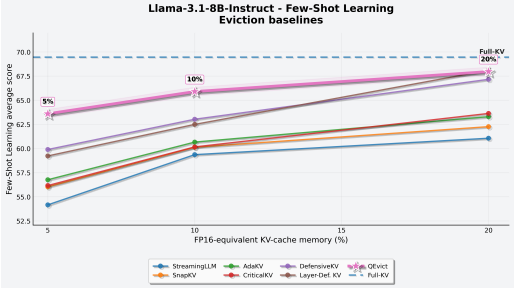
(d) Multi-Document QA: quantization / mixed precision.



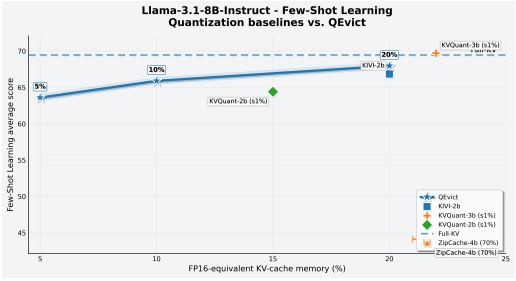
(e) Summarization: eviction.



(f) Summarization: quantization / mixed precision.

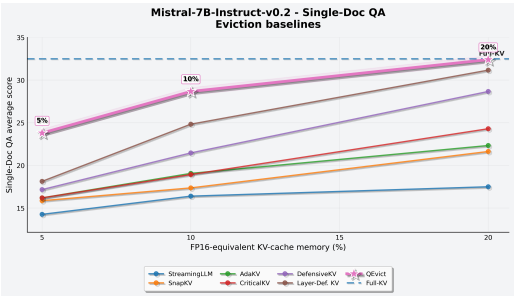


(g) Few-Shot Learning: eviction.

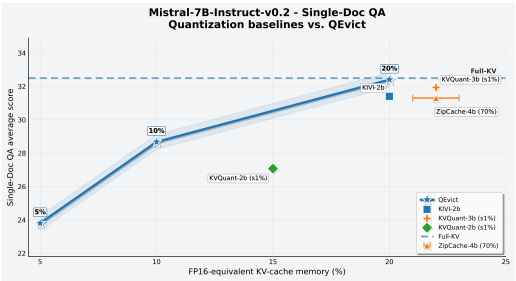


(h) Few-Shot Learning: quantization / mixed precision.

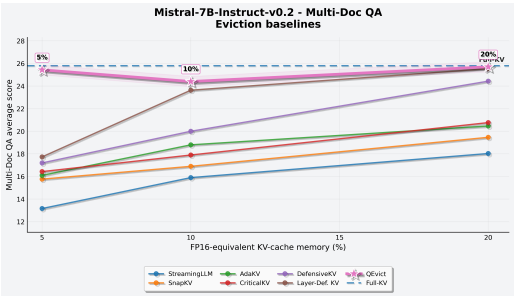
Figure 3: **LongBench performance–memory Pareto analysis for Llama-3.1-8B-Instruct.** Rows correspond to Single-Document QA, Multi-Document QA, Summarization, and Few-Shot Learning. The left column compares `QEvict` with eviction baselines, while the right column compares it with quantization and mixed-precision baselines. Dashed horizontal lines denote FullKV performance.



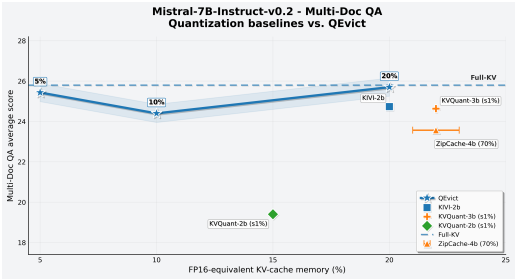
(a) Single-Document QA: eviction.



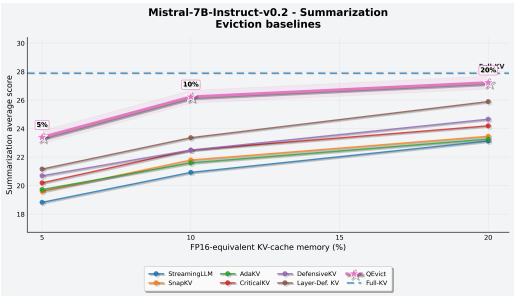
(b) Single-Document QA: quantization / mixed precision.



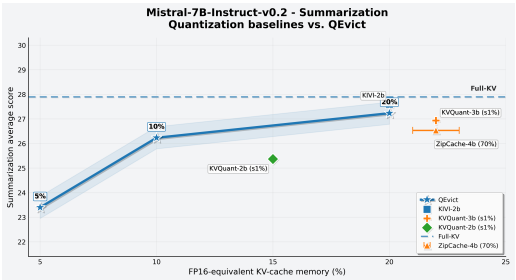
(c) Multi-Document QA: eviction.



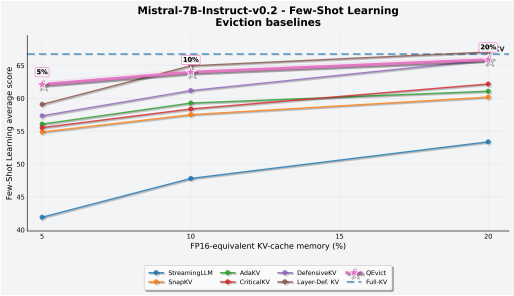
(d) Multi-Document QA: quantization / mixed precision.



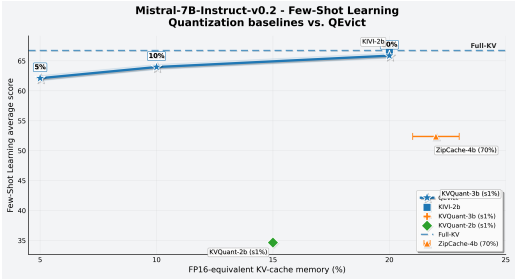
(e) Summarization: eviction.



(f) Summarization: quantization / mixed precision.



(g) Few-Shot Learning: eviction.



(h) Few-Shot Learning: quantization / mixed precision.

Figure 4: **LongBench performance–memory Pareto analysis for Mistral-7B-Instruct-v0.2.** Rows correspond to Single-Document QA, Multi-Document QA, Summarization, and Few-Shot Learning. The left column compares `QEvict` with eviction baselines, while the right column compares it with quantization and mixed-precision baselines. Dashed horizontal lines denote FullKV performance.

Table 5: **RULER task families used in our evaluation.**

Task Family	Tasks	Evaluation Focus
Aggregation	CWE, FWE	Aggregating information distributed throughout the context.
Needle-in-a-Haystack	MK-1–MK-3, MQ, MV, S-1–S-3	Retrieving sparse evidence under varying numbers of keys, values, and queries.
Question Answering	QA-1, QA-2	Retrieving and combining evidence from relevant passages.
Variable Tracking	VT	Preserving and following dependent variable assignments across the sequence.

quence. Successful generation requires preserving intermediate decoding states throughout a multi-step solution as subsequent tokens are generated.

We evaluate Llama-3.1-8B-Instruct (Dubey et al., 2024) and Mistral-7B-Instruct-v0.2 (Jiang et al., 2023) across multiple measured KV-memory budgets. All methods use the same prompts, greedy decoding procedure, generation limits, and answer-extraction protocol. Performance is reported as exact-match accuracy, and memory is measured relative to the FullKV cache. Eviction methods are compared at matched memory budgets, while quantization and mixed-precision methods are reported at their measured operating points following Appendix B.3.

For GSM8K, `QEvict` uses historical windows of $W = 8$ tokens, routes every $R = 8$ decoding steps, preserves five sink tokens, and protects the 32 most recent tokens. Other method-specific configurations and memory-accounting rules follow Appendix B.

Tables 6 and 7 report the corresponding numerical results across all evaluated operating points.

G Ablation Studies

We study the sensitivity of `QEvict` to three parameters governing its retention and query-time access behavior: the historical window size W, the fraction α of the historical byte budget assigned to the compressed Q tier, and the query-time Q-tier read ratio ρ. All ablations use Llama-3.1-8B-Instruct under a 20% KV-cache budget and follow the LongBench evaluation protocol in Appendix C. To limit evaluation cost while covering the major LongBench capabilities, we select one task from each task family: NarrativeQA for Single-Document QA, MuSiQue for Multi-Document QA, QMSum for Summarization, and TriviaQA for Few-Shot Learning.

Table 6: **GSM8K performance on Llama-3.1-8B-Instruct.** The left subtable compares KV-cache eviction methods at different KV-cache budgets. The right subtable compares quantization and mixed-precision methods at approximately matched memory footprints. Full-KV is excluded from the best and second-best rankings.

(a) KV-cache eviction

Method	KV-Cache Budget				
	10%	**20%**	**40%**	**60%**	**80%**
Full-KV			86.9		
DefensiveKV	3.1	11.5	76.0	81.7	<u>84.8</u>
Layer-Def. KV	3.2	<u>31.2</u>	<u>82.6</u>	<u>84.1</u>	84.2
CriticalKV	<u>3.3</u>	7.0	31.2	59.1	81.9
SnapKV	1.9	4.1	21.6	51.0	77.0
AdaKV	2.2	5.2	24.0	52.8	72.8
StreamingLLM	2.0	3.6	66.8	82.9	84.5
`QEvict`	**18.9**	**74.6**	**85.4**	**85.8**	**85.8**

(b) Quantization & Mixed precision

Method	Equivalent KV Memory	Score
Full-KV	100%	86.9
`QEvict`	20%	74.6
KIVI-2b	∼20%	**80.0**
ZipCache-4b	∼22%	76.5
MiKV-2b	∼20%	72.9
ResQ-2b	∼20.3%	52.6
KVQuant-2b	∼15%	76.6
KVQuant-3b	∼22%	<u>78.2</u>

Table 7: **GSM8K performance on Mistral-7B-Instruct-v0.2.** The left subtable compares KV-cache eviction methods at different KV-cache budgets. The right subtable compares quantization and mixed-precision methods at approximately matched memory footprints. Full-KV is excluded from the best and second-best rankings.

(a) KV-cache eviction

Method	KV-Cache Budget				
	10%	20%	40%	60%	80%
Full-KV			46.2		
DefensiveKV	2.4	10.9	39.9	42.7	43.9
Layer-Def. KV	1.9	20.1	39.8	43.8	43.7
CriticalKV	2.1	5.2	11.2	22.0	36.2
SnapKV	1.5	2.7	7.9	18.5	32.8
AdaKV	1.5	3.4	10.8	19.9	30.4
StreamingLLM	1.6	2.0	29.6	40.3	42.2
QEvict	**5.2**	**36.9**	**44.4**	**45.9**	**45.7**

(b) Quantization & Mixed precision

Method	Equivalent KV Memory	Score
Full-KV	100%	46.2
QEvict	20%	36.9
KIVI-2b	∼20%	40.9
ZipCache-4b	∼22%	40.6
MiKV-2b	∼20%	36.0
ResQ-2b	∼20.3%	22.8
KVQuant-2b	∼15%	38.1
KVQuant-3b	∼22%	**41.8**

Unless otherwise stated, we use the default configuration

$$W = 8, \qquad R = 8, \qquad \alpha = 0.70, \qquad \rho = 0.25,$$

with INT2 storage for the Q tier. Each ablation varies one parameter while holding the remaining settings fixed. Overall, performance changes modestly around the default configuration, indicating that `QEvict` does not depend on a narrowly tuned operating point.

G.1 Historical Window Size

We vary the historical window size over

$$W \in \{4, 8, 16, 32\},$$

with the routing interval set to $R = W$, while fixing $\alpha = 0.70$ and $\rho = 0.25$. Smaller windows permit finer-grained retention and query-time selection, whereas larger windows group more neighboring states under a shared routing decision.

Table 8: **Effect of historical window size on LongBench.** Results use Llama-3.1-8B-Instruct under a 20% KV-cache budget with $\alpha = 0.70$, $\rho = 0.25$, and INT2 Q-tier storage. The routing interval is set to $R = W$. One representative task is selected from each LongBench family. The best and second-best results in each column are shown in bold and underlined, respectively. The highlighted row denotes the default configuration used in the main experiments.

W	Nar.QA	MuSiQue	QMSum	Tri.QA	Avg.
4	28.73	31.10	25.61	90.86	44.08
8	**29.92**	**32.64**	**25.78**	91.04	**44.84**
16	28.99	30.94	25.56	91.02	44.13
32	28.81	32.01	25.27	**91.88**	44.49

Performance is stable across the tested window sizes. The four-task average ranges from 44.38 to 44.84, with the default $W = 8$ achieving the highest average. The preferred granularity nevertheless varies by task: $W = 8$ performs best on NarrativeQA and QMSum, $W = 4$ performs best on MuSiQue, and $W = 32$ achieves the highest TriviaQA score. These results suggest that a moderate window size provides a useful compromise between fine-grained cache decisions and coherent window-level routing, motivating $W = 8$ as our default.

G.2 Q-Tier Budget Allocation

We next vary the fraction of the historical byte budget assigned to the compressed Q tier,

$$\alpha \in \{0.10, 0.30, 0.50, 0.70, 0.90\},$$

while fixing $W = 8$, $R = 8$, and $\rho = 0.25$. Increasing α allocates more of the fixed historical byte budget to compact INT2 storage, increasing low-bit historical coverage while reducing the capacity available to the execution-datatype tier.

Table 9: **Effect of Q-tier budget allocation on LongBench.** Results use Llama-3.1-8B-Instruct under a 20% KV-cache budget with $W = 8$, $R = 8$, $\rho = 0.25$, and INT2 Q-tier storage. One representative task is selected from each LongBench family. The best and second-best distinct results in each column are shown in bold and underlined, respectively. The highlighted row denotes the default configuration used in the main experiments.

α	Nar.QA	MuSiQue	QMSum	Tri.QA	Avg.
0.10	28.55	32.22	24.66	90.34	43.94
0.30	27.95	32.42	24.91	89.66	43.73
0.50	<u>29.75</u>	<u>32.69</u>	25.31	90.48	44.56
0.70	**29.92**	32.64	**25.78**	**91.04**	**44.84**
0.90	**29.92**	**32.91**	<u>25.49</u>	<u>90.64</u>	<u>44.74</u>

The lowest allocations yield four-task averages of 43.94 at $\alpha = 0.10$ and 43.73 at $\alpha = 0.30$. Performance improves to 44.81 at $\alpha = 0.50$ and peaks at 44.84 for the default $\alpha = 0.70$, before decreasing slightly to 44.74 at $\alpha = 0.90$. Individual tasks exhibit different preferred allocations: the largest Q tier performs best on MuSiQue and TriviaQA, whereas $\alpha = 0.70$ achieves the best QMSum score and ties the best NarrativeQA score. These results support allocating a substantial fraction of historical memory to compact storage, while also preserving sufficient execution-datatype capacity. We therefore use $\alpha = 0.70$ as the default.

G.3 Query-Time Q-Tier Read Ratio

Finally, we vary the fraction of retained Q-tier windows selected for token-level access at each decoding step,

$$\rho \in \{0.10, 0.25, 0.50, 0.75\},$$

while fixing $W = 8$, $R = 8$, and $\alpha = 0.70$. Decreasing ρ reduces the amount of compressed history whose token-level K/V must be read and dequantized for the current query. The remaining Q-tier windows stay resident and continue to contribute through the window-level completion mechanism.

Table 10: **Effect of query-time Q-tier read ratio on LongBench.** Results use Llama-3.1-8B-Instruct under a 20% KV-cache budget with $W = 8$, $R = 8$, $\alpha = 0.70$, and INT2 Q-tier storage. One representative task is selected from each LongBench family. The best and second-best distinct results in each column are shown in bold and underlined, respectively. The highlighted row denotes the default configuration used in the main experiments.

ρ	Nar.QA	MuSiQue	QMSum	Tri.QA	Avg.
0.10	**30.34**	<u>32.43</u>	25.18	90.69	44.66
0.25	<u>29.92</u>	**32.64**	**25.78**	<u>91.04</u>	**44.84**
0.50	29.73	31.24	<u>25.53</u>	**91.25**	<u>44.43</u>
0.75	29.67	31.24	25.34	90.80	<u>44.26</u>

Performance varies little across the tested read ratios. The four-task average is 44.66 at $\rho = 0.10$, rises to 44.84 at the default $\rho = 0.25$, and is 44.68 at both $\rho = 0.50$ and $\rho = 0.75$. The task-level results also show that more token-level access does not consistently improve quality: $\rho = 0.10$ gives the highest NarrativeQA score, $\rho = 0.25$ performs best on MuSiQue and QMSum, and $\rho = 0.50$

Table 11: **Complete end-to-end efficiency measurements on Llama-3.1-8B-Instruct.** All runs use a 256-token prefill. Decode throughput is aggregated across the batch. OOM denotes configurations that exceed available GPU memory.

Decode	Batch	Method	TTFT (s)	TPOT (s)	Decode (tok/s)	Peak (GiB)
8K	1	Full-KV (eager)	0.080	0.0489	20.5	16.15
		Full-KV (flash)	0.076	0.0451	22.2	16.04
		MiKV	0.069	0.0581	17.2	16.14
		ResQ	0.368	0.1500	6.7	16.03
		QEvict	0.078	0.0678	14.8	15.61
	32	Full-KV (eager)	0.750	0.2962	108.0	52.19
		Full-KV (flash)	**0.690**	0.0991	322.8	48.51
		MiKV	1.020	0.2956	108.2	52.41
		ResQ	1.053	0.2512	127.4	48.49
		QEvict	1.106	**0.0745**	**429.3**	**34.70**
16K	1	Full-KV (eager)	0.080	0.0483	20.7	17.28
		Full-KV (flash)	0.076	0.0449	22.3	17.05
		MiKV	0.084	0.0666	15.0	17.27
		ResQ	0.368	0.1499	6.7	17.04
		QEvict	0.079	0.0676	14.8	16.24
	32	Full-KV (eager)		OOM		
		Full-KV (flash)		OOM		
		MiKV		OOM		
		ResQ		OOM		
		QEvict	0.712	0.1493	214.3	54.78

performs best on TriviaQA. Thus, reading substantially more of the retained Q tier provides no consistent benefit in this sweep. This supports the central separation in `QEvict`: broad historical retention can be maintained while token-level access remains selective for each query.

H Efficiency and Systems Analysis

H.1 Complete End-to-End Results

Table 11 reports the complete measurements underlying Table 3. We evaluate batch sizes 1 and 32 with a 256-token prefill and decode lengths of 8,192 and 16,384 tokens. TTFT denotes time to first token, TPOT denotes time per output token, and decode throughput is aggregated across all generated tokens in the batch.

At batch size 1, the additional compact Q-tier scan and selection work is not sufficiently amortized to improve decode latency over optimized Full-KV. The regime changes at batch size 32, where KV traffic becomes substantially more memory intensive. At 8K decode, `QEvict` reaches 429.3 tok/s with 34.70 GiB peak memory, compared with 322.8 tok/s and 48.51 GiB for Full-KV with FlashAttention-2. At 16K decode, `QEvict` is the only evaluated method that fits at batch size 32.

H.2 Measurement Protocol

All efficiency measurements use identical hardware, model precision, batch size, sequence lengths, and execution backend within each comparison. We run warm-up iterations before measurement to initialize kernels and memory pools, synchronize the GPU around timed regions, and reset peak-memory statistics immediately before each run.

We report time to first token (TTFT), time per output token (TPOT), decode throughput, and peak GPU memory. For a run producing N_{out} output tokens, with timestamps t_{start}, t_{first}, and t_{end}, we

compute

$$\mathrm{TTFT} = t_{\mathrm{first}} - t_{\mathrm{start}}, \qquad \mathrm{TPOT} = \frac{t_{\mathrm{end}} - t_{\mathrm{first}}}{N_{\mathrm{out}} - 1}, \tag{16}$$

and define single-sequence decode throughput as

$$\mathrm{Throughput}_{\mathrm{decode}} = \frac{N_{\mathrm{out}} - 1}{t_{\mathrm{end}} - t_{\mathrm{first}}}. \tag{17}$$

For batched inference, decode throughput is computed from the total number of generated post-first tokens across the batch divided by the corresponding decode wall-clock time. Peak GPU memory includes model parameters, persistent KV-cache storage, quantization metadata, compact Q-tier query-access representations, attention workspaces, routing state, and temporary migration buffers. All methods are measured using their actual persistent cache representations rather than nominal bit-width estimates. The principal batch-32 comparison is summarized in Table 3 of the main paper, while complete batch-1 and batch-32 measurements at both decode lengths are reported in Appendix H.1.

H.3 Runtime Components

The runtime of `QEvict` follows the separation between slow cache routing and fast query-conditioned access introduced in Section 4. Routing is performed once every R decoding steps. A routing event aggregates received-attention statistics, ranks historical windows, determines their storage assignments under the byte budget, and performs any required migrations. Its amortized per-token overhead is

$$T_{\mathrm{route/token}} = \frac{T_{\mathrm{score}} + T_{\mathrm{rank}} + T_{\mathrm{assign}} + T_{\mathrm{migrate}}}{R}. \tag{18}$$

Quantization is incurred only when a window first enters the Q tier. Its persistent INT2 record is subsequently reused, so later transitions do not re-quantize the same window. Migrations are batched across layers that share the same routing geometry to reduce launch and bookkeeping overhead.

At every decode step, `QEvict` first scans the compact representation of the retained Q tier to estimate query-conditioned window relevance. For each KV head, only the selected fraction ρ of Q-tier windows proceeds to token-level attention. The fused mixed-precision attention path reads the corresponding packed INT2 keys and values directly from persistent storage, dequantizes them in registers, reapplies positional rotation to the keys, and performs grouped-query attention together with the execution-datatype cache. The complete Q tier is therefore never expanded into an execution-datatype tensor in HBM.

Unread Q-tier windows are handled by the completion mechanism rather than by token-level K/V reads. The kernel accumulates the quantities required for selected-window attention mass during the attention computation and applies the calibrated window-level completion in the epilogue. This preserves a contribution from retained but unread history without paying the bandwidth cost of loading its full K/V states.

The compact Q-tier scan introduces additional arithmetic, but its input is substantially smaller than the packed K/V representation and avoids loading most Q-tier token states. Consequently, increasing historical coverage does not require proportional growth in token-level memory traffic. The principal execution benefit of `QEvict` therefore comes from replacing a full scan of retained low-bit K/V with a compact relevance scan followed by selective mixed-precision reads.

H.4 Attention-Score Accumulation

Long-horizon routing requires the attention received by each historical window. During prefill, we compute these statistics using a key-major reduction that reuses the row-wise softmax normalization quantities already produced by the FlashAttention-style attention computation. This avoids materializing the full attention matrix solely for cache scoring.

During decoding, the fused attention path accumulates selected-window mass while computing the attention output. For Q-tier windows that are not read at token level, the completion masses defined

in Section 4 are incorporated into the same long-horizon statistics. Thus, cache scoring remains consistent with the completed attention distribution without materializing dense attention probabilities over the entire retained history.

H.5 Persistent Representation and Memory Traffic

The Q tier is stored in a migration-stable packed representation. Four INT2 codes are packed per byte, with the associated quantization grids and metadata kept in their compressed persistent form. Each Q window additionally stores the compact query-access representation used by the relevance scan. These components are included in the persistent-memory accounting described in Appendix B.4.

When a Q-tier window is selected for token-level access, its packed K/V values are loaded once by the mixed-precision kernel and reconstructed in registers. No dequantized copy of the entire Q tier is written to global memory. When a window returns to execution-datatype storage, its K/V are reconstructed from the same immutable INT2 record rather than from a newly quantized copy. This both bounds migration error and avoids redundant persistent representations during repeated storage-priority changes.

H.6 Hardware and Software Settings

All experiments were run on a node with 8 NVIDIA A100 80GB GPUs. Larger evaluation sweeps were parallelized by sharding examples round-robin across GPUs, with one complete model replica per GPU. Unless otherwise stated, individual inference runs use a single GPU. The software stack consists of PyTorch 2.6.0 with CUDA 12.4, Transformers 4.47.1, and flash-attn 2.8.3. All compared methods within an efficiency experiment use the same software environment and execution precision.

I Additional Observation Details

This section provides the diagnostic protocol, formal metric definitions, and supporting measurements for Section 3. The diagnostics characterize three properties of long-context KV-cache behavior: the breadth of long-horizon historical utility, the temporal evolution of storage priority, and the concentration of relevance for the current query.

I.1 Diagnostic Protocol

We collect reference attention traces from an uncompressed FullKV execution because attention assigned to states that would be inaccessible under compression cannot be measured from the compressed cache alone. The analysis uses Llama-3.1-8B-Instruct (Dubey et al., 2024) on eight WikiText-103 test articles. Each trace contains a 512-token prompt followed by 512 greedy decoding steps.

We use five protected sink tokens, a 128-token recent region, historical window size $W = 8$, routing interval $R = 8$, and a 20% persistent KV-cache budget. Under this budget, the historical allocation contains $K_F = 2$ execution-datatype windows and $K_Q = 26$ INT2 windows. The query-conditioned read ratio is $\rho = 0.25$, yielding $K_R = 7$ selected Q windows per KV head at each decode step.

Let $A_t^{\ell,h}(i)$ denote the attention probability assigned by the FullKV reference at decode step t, layer ℓ, and query head h to cached token i. For historical window w, its reference attention mass at step t is

$$m_{t,\mathrm{ref}}^{\ell,h}(w) = \sum_{i \in w} A_t^{\ell,h}(i), \tag{19}$$

and its cumulative reference score is

$$S_{t,\mathrm{ref}}^{\ell,h}(w) = \sum_{\tau \le t} m_{\tau,\mathrm{ref}}^{\ell,h}(w). \tag{20}$$

Unless stated otherwise, diagnostics are first computed over their eligible layer, head, routing-event, or decode-step instances and are then aggregated over the full set of eight traces.

For same-memory eviction comparisons, we simulate H_2O-style and StreamingLLM-style FP16 policies on the same reference attention traces. The H_2O-style policy is given exact attention-derived scores, making this a favorable comparison for FP16-only retention.

I.2 Diagnostic Metrics

Historical importance coverage. For each reference trace, historical windows are ranked by cumulative FullKV attention. We report the fraction of windows required to account for a specified fraction of the cumulative historical attention mass. This diagnostic measures whether long-horizon utility is concentrated in a small heavy-hitter set or distributed across a broader fraction of the history.

Future Missed Mass. Let routing event r occur at decode step t_r, and let $\mathcal{E}_r^\ell$ denote historical windows made inaccessible by the policy at that event. Let $\mathcal{P}_r^\ell$ contain the non-sink historical positions that already exist at t_r. For future horizon H, we define

$$\mathrm{FMM}_H^{\ell,h}(r) = \frac{\sum_{\tau=t_r+1}^{\min(t_r+H,T)} \sum_{\substack{i\in\mathcal{E}_r^\ell \\ i\le t_r}} A_\tau^{\ell,h}(i)}{\sum_{\tau=t_r+1}^{\min(t_r+H,T)} \sum_{i\in\mathcal{P}_r^\ell} A_\tau^{\ell,h}(i)}. \tag{21}$$

Only states already present when the routing decision is made contribute to either term. Lower FMM indicates that less future attention is assigned to history made permanently inaccessible by the cache policy. The main paper uses $H = 32$, and we additionally report $H \in \{8, 64\}$.

Selection Churn. Let

$$\mathcal{H}_r^\ell = \mathcal{F}_{t_r}^\ell \cup \mathcal{Q}_{t_r}^\ell$$

denote the historical windows retained after routing event r. Selection Churn is the Jaccard distance between consecutive retained sets,

$$\mathrm{Churn}(r) = 1 - \frac{|\mathcal{H}_r^\ell \cap \mathcal{H}_{r+1}^\ell|}{|\mathcal{H}_r^\ell \cup \mathcal{H}_{r+1}^\ell|}. \tag{22}$$

A value of zero indicates an unchanged retained set. Protected sink and recent regions are excluded because their residency is fixed by construction.

Future-attention lift. For candidate window w at routing event r, define its reference utility over the next H decode steps as

$$U_H^\ell(r,w) = \frac{1}{H_q} \sum_{h=1}^{H_q} \sum_{\tau=t_r+1}^{\min(t_r+H,T)} \sum_{i\in w} A_\tau^{\ell,h}(i). \tag{23}$$

Let $\mathcal{W}_r^\ell$ denote the candidate windows considered at the routing event, and define their mean future utility as

$$\overline{U}_H^\ell(r) = \frac{1}{|\mathcal{W}_r^\ell|} \sum_{u\in\mathcal{W}_r^\ell} U_H^\ell(r,u). \tag{24}$$

For transition class c, its future-attention lift is

$$\mathrm{Lift}_H(c) = \frac{1}{|\mathcal{T}_c|} \sum_{(r,\ell,w)\in\mathcal{T}_c} \frac{U_H^\ell(r,w)}{\overline{U}_H^\ell(r)}, \tag{25}$$

where $\mathcal{T}_c$ contains the windows undergoing transition c. A value above one indicates greater future attention than the average candidate considered by the same routing decisions. We use $H = 32$.

Table 12: **Future Missed Mass across horizons.** All comparison policies use the same memory budget. The F-only condition makes the Q tier inaccessible without reallocating its bytes and is therefore a mechanistic ablation rather than a matched-memory competitor.

H	QEvict	H_2O	StreamingLLM	F only
8	24.2%	37.9%	39.2%	42.9%
32	28.9%	45.9%	47.9%	52.3%
64	30.7%	49.7%	52.4%	56.9%

Query-conditioned read recall. Let $k(h)$ denote the KV head shared by query head h, and let $\mathcal{R}_{t,k(h)}$ denote the Q windows selected for token-level access. The reference Q-tier attention captured by the selected set is

$$\mathrm{Recall}^{Q}_{t,h} = \frac{\sum_{w \in \mathcal{R}_{t,k(h)}} \sum_{i \in w} A^{\ell,h}_{t}(i)}{\sum_{w \in \mathcal{Q}^{\ell}_{t}} \sum_{i \in w} A^{\ell,h}_{t}(i)}. \tag{26}$$

We evaluate recall on query-head steps for which Q accounts for at least 1% of the reference attention. *Uniform* selects the same number of Q windows uniformly, while *hindsight* selects the same number using their true current-query reference attention. Hindsight therefore represents the best same-cardinality selection available after observing the reference attention for the current query.

For completeness, the reference attention assigned to unread Q history is

$$M^{\mathrm{unread}}_{t,h} = \sum_{u \in \mathcal{Q}^{\ell}_{t} \setminus \mathcal{R}_{t,k(h)}} \sum_{i \in u} A^{\ell,h}_{t}(i). \tag{27}$$

This quantity measures the reference contribution that would be discarded if query-time non-selection were treated as cache eviction.

I.3 Broad Historical Utility

The top 10% of historical windows contain only 22.4% of cumulative reference attention. Covering 50%, 80%, and 90% of cumulative attention requires 32.2%, 64.9%, and 79.4% of historical windows, respectively. Thus, long-horizon utility is distributed across a substantial fraction of the history rather than concentrated in a small heavy-hitter set.

Under the diagnostic budget, QEvict retains 28 evictable historical windows, with two in F and 26 in Q, while the same bytes hold eight FP16 historical windows. Table 12 shows that this broader retention substantially reduces attention assigned to history that becomes unavailable to subsequent queries.

At the $H = 32$ horizon used in the main paper, QEvict reduces FMM from 45.9% for H_2O and 47.9% for StreamingLLM to 28.9%. Making the retained Q history inaccessible raises FMM to 52.3%, showing that the gain comes from maintaining broader historical availability rather than only from the execution-datatype subset.

I.4 Temporal Evolution of Historical Importance

Historical retention changes gradually across routing events. The retained historical set has mean Selection Churn 0.044, compared with 0.080 for the same-byte FP16 eviction reference. Moreover, 99.6% of Q windows remain in Q at the next routing event, and approximately one window per layer changes assignment at each event. These measurements indicate substantial persistence in long-horizon importance while still allowing the storage priority of individual windows to evolve.

The windows whose storage assignments change exhibit substantially different subsequent utility, as shown in Table 13.

The run contains 221 Q→F transitions, 221 F→Q transitions, and 1,336 Q→E transitions. Windows moved from Q to F receive $2.34\times$ the candidate-average future attention, while windows remaining

Table 13: **Future-attention lift of routing outcomes.** A value of $1.0\times$ corresponds to the mean future utility of all candidate windows considered by the same routing events.

Routing outcome	Future-attention lift
$\mathrm{Q} \to \mathrm{F}$	$2.34\times$
$\mathrm{F} \to \mathrm{F}$	$1.63\times$
$\mathrm{F} \to \mathrm{Q}$	$1.19\times$
$\mathrm{Q} \to \mathrm{Q}$	$0.94\times$
$\mathrm{Q} \to \mathrm{E}$	$0.66\times$

Table 14: **Query-conditioned Q-tier access.**

Measure	Value
Windows selected	$7/26$ (26.9%)
Reference Q attention captured	59.2%
Uniform same-cardinality selection	26.9%
Hindsight same-cardinality selection	63.1%
Recall relative to hindsight	93.9%
Highest-mass Q window selected	92.0%

in Q receive $0.94\times$ and windows removed from Q receive $0.66\times$. The transition classes therefore correlate with meaningful differences in subsequent attention despite the low overall retention churn. This supports periodic reassignment using evidence accumulated across multiple queries rather than fixing storage priority after a single decision.

I.5 Query-Conditioned Access

Table 14 evaluates query-conditioned selection within the 26-window Q tier. The configured read ratio selects seven Q windows per KV head, corresponding to 26.9% of the retained low-bit windows.

The compact query representation captures 59.2% of Q-tier reference attention while reading only 26.9% of the Q windows. This corresponds to 93.9% of the attention captured by a same-cardinality hindsight selector and substantially exceeds uniform selection. The highest-mass Q window is included in the selected set for 92.0% of eligible query-head steps.

The unread portion nevertheless remains relevant. Across eligible queries, unread Q windows account for 2.9% of total reference attention. A complementary attention ledger assigns 82.8% of reference attention to sink, recent, and F states, 4.4% to selected Q windows, 2.9% to unread Q windows, and 9.9% to permanently evicted history. These measurements motivate separating query-time access from residency: most Q-tier token-level reads can be avoided for a given query without treating the unread retained history as absent.

J Additional Methodological Details

This section provides implementation and representation details omitted from Section 4. The complete cache transitions and query-conditioned read procedure are summarized in Algorithms 1 and 2. We first describe the persistent INT2 representation, followed by the compact representation used for query-dependent Q-tier selection and the completion of windows that are not read at token resolution. We then describe the GPU execution path and the numerical conventions required to keep these representations consistent across tier transitions.

J.1 INT2 Representation and Quantization Metadata

When a window first enters $\mathcal{Q}$, `QEvict` constructs a persistent INT2 representation of its keys and values. Following the asymmetric K/V quantization geometry of KIVI (Liu et al., 2024), keys are quantized per channel across the W tokens in the window, while values are quantized per token

across the d_h channels. This asymmetry reflects the different distributional structure of keys and values and is maintained throughout all subsequent tier transitions.

For a quantization group x, let $x_{\min}$ and $x_{\max}$ denote its extrema. The nominal affine grid has scale $s = (x_{\max} - x_{\min})/3$ and offset $z = x_{\min}$. The scale and offset are first encoded into their persistent metadata representation and decoded to $\hat{s}$ and $\hat{z}$. INT2 codes are then computed using

$$c(x) = \mathrm{clip}_{[0,3]}\left(\mathrm{round}\,\frac{x - \hat{z}}{\hat{s}}\right), \qquad \hat{x} = \hat{z} + \hat{s}\,c(x). \tag{28}$$

Fitting codes to the stored grid is important because every subsequent reconstruction uses $\hat{s}$ and $\hat{z}$, rather than the higher-precision intermediate values from which the grid was obtained. A window is therefore decoded from the same quantizer that determined its original codes.

Four INT2 values are packed into one byte. For a window $K, V \in \mathbb{R}^{H_{kv} \times W \times d_h}$, packed keys are stored in channel-major form with shape $[H_{kv}, d_h, W/4]$, while packed values use the token-major layout $[H_{kv}, W, d_h/4]$. The former matches per-channel key quantization and the latter matches per-token value quantization.

The affine metadata are compressed as well. Scale and offset arrays are stored using one-byte entries together with a shared FP16 multiplier for each small group of entries. We use groups of 32 along standard wide axes and a single group when the axis is shorter than 32. This substantially reduces the metadata cost relative to storing every affine parameter directly in FP16. The resulting storage contribution is included in the complete accounting in Appendix B.4.

Keys require an additional positional convention. The persistent Q record stores them before RoPE, together with their original absolute token positions. Whenever a Q window is evaluated or materialized in $\mathcal{F}$, the key is reconstructed and RoPE is reapplied at those original positions. Eviction therefore changes neither the positional identity of a window nor the position used by subsequent attention.

The packed representation, affine metadata, and positions are created only on the first transition into $\mathcal{Q}$. A later transition back to $\mathcal{Q}$ reactivates the same record. The low-bit representation is therefore invariant to the number of F–Q transitions experienced by the window.

J.2 Compact Representation for Query-Conditioned Reads

The INT2 record is optimized for storage and exact token-level reads. Scanning the complete record of every Q window merely to decide which windows should be read would negate much of this advantage. Each Q window therefore stores a second, substantially smaller representation used only for query-dependent selection.

The representation is constructed from the post-RoPE keys of the window at its first demotion. For one KV head, let $\{\mathbf{k}_{w,i}\}_{i=1}^{W}$ denote these keys. We compute the centroid $\boldsymbol{\mu}_w = W^{-1}\sum_i \mathbf{k}_{w,i}$ and choose the largest within-window residual,

$$j^\star = \arg\max_i \left\|\mathbf{k}_{w,i} - \boldsymbol{\mu}_w\right\|_2. \tag{29}$$

The associated unit direction and token coefficients are

$$\mathbf{u}_w = \frac{\mathbf{k}_{w,j^\star} - \boldsymbol{\mu}_w}{\left\|\mathbf{k}_{w,j^\star} - \boldsymbol{\mu}_w\right\|_2}, \qquad \xi_{w,i} = \left(\mathbf{k}_{w,i} - \boldsymbol{\mu}_w\right)^\top \mathbf{u}_w. \tag{30}$$

Each key is then represented by $\widehat{\mathbf{k}}_{w,i} = \boldsymbol{\mu}_w + \xi_{w,i}\mathbf{u}_w$.

This rank-1 form retains both a window-level common component and one dominant direction of within-window variation. More importantly for decoding, it reduces the query interaction with a window to two vector products, $\mathbf{q}^\top\boldsymbol{\mu}_w$ and $\mathbf{q}^\top\mathbf{u}_w$. Approximate logits for the W tokens are then formed using only the scalar coefficients $\xi_{w,i}$.

The same representation stores the value centroid $\boldsymbol{\nu}_w = W^{-1}\sum_i \mathbf{v}_{w,i}$. The centroid is not used to rank windows. It provides a window-level value representation for the completion mechanism in Appendix J.3 when a Q window is retained but not read at token resolution.

To keep the scan bandwidth small, the card itself uses mixed-width quantization. The key centroid $\boldsymbol{\mu}_w$ and value centroid $\boldsymbol{\nu}_w$ are represented as INT4 residuals from fixed per-layer, per-KV-head

anchors. The direction $\mathbf{u}_w$ and token coefficients $\boldsymbol{\xi}_w$ are stored in INT8. Each field uses a per-window, per-head FP16 scale. The anchors are initialized from the first set of windows entering $\mathcal{Q}$ and remain fixed thereafter so that an existing card never changes interpretation.

For query head h, the compact representation gives the approximate token logits

$$\widehat{z}_{t,h,w,i} = \frac{(\mathbf{q}_t^{\ell,h})^\top \left(\widehat{\boldsymbol{\mu}}_w + \widehat{\xi}_{w,i}\widehat{\mathbf{u}}_w\right)}{\sqrt{d_h}}. \tag{31}$$

Rather than ranking a window by one representative token, we estimate its aggregate log attention mass as

$$\widehat{\ell}_{t,h}(w) = \mathrm{LSE}_{i=1}^{W}\, \widehat{z}_{t,h,w,i}. \tag{32}$$

This aligns the routing quantity with the total mass that the window would contribute to a softmax.

Grouped-query attention requires an additional normalization. Query heads that share a KV head can have different logit offsets and scales, making their raw window scores unsuitable for direct comparison. We therefore normalize $\widehat{\ell}_{t,h}(w)$ over the Q tier separately for each query head and only then take the maximum over the heads sharing a KV head. Selection is therefore based on whether a window accounts for a large fraction of the compressed-tier relevance for at least one head in the group.

The resulting representation is used only to allocate read bandwidth. The underlying INT2 K/V record remains the representation used whenever the window is selected for token-level attention.

J.3 Calibration and Completion of Unread Windows

Query-conditioned selection partitions the active Q tier into windows that are read at token resolution and windows that remain represented only through their compact summaries. Simply omitting the latter would alter both the current attention output and the future retention scores. `QEvict` instead estimates their contribution relative to the attention computation already performed on the selected cache.

Consider one query head and let $\mathcal{A}$ contain the sink and recent regions, the F tier, and the selected Q windows. If $z_i = \mathbf{q}^\top \mathbf{k}_i/\sqrt{d_h}$, exact attention over $\mathcal{A}$ uses $Z_{\mathcal{A}} = \sum_{i\in\mathcal{A}} \exp(z_i)$ and produces the restricted output $\bar{\mathbf{o}} = \sum_{i\in\mathcal{A}} \exp(z_i)\mathbf{v}_i/Z_{\mathcal{A}}$.

For a selected Q window w, its measured restricted mass is

$$\bar{m}(w) = \frac{\sum_{i\in w} \exp(z_i)}{Z_{\mathcal{A}}}. \tag{33}$$

The compact representation independently predicts $\widehat{\ell}(w) \approx \log \sum_{i\in w} \exp(z_i)$. Their difference is therefore approximately a common normalization term,

$$\log \bar{m}(w) - \widehat{\ell}(w) \approx -\log Z_{\mathcal{A}} + \epsilon_w, \tag{34}$$

where ϵ_w captures approximation error of the compact representation.

We estimate this query-specific correction by averaging the log differences over selected windows with positive measured mass,

$$\delta = \frac{1}{|\mathcal{R}^+|} \sum_{w\in\mathcal{R}^+} \left[\log \bar{m}(w) - \widehat{\ell}(w)\right]. \tag{35}$$

For each unread Q window u, the resulting provisional mass is $\widetilde{m}(u) = \exp(\widehat{\ell}(u) + \delta)$.

Because the restricted softmax over $\mathcal{A}$ already sums to one, the completed normalization is $Z = 1 + \sum_{u\in\mathcal{U}} \widetilde{m}(u)$. Using the stored value centroid $\boldsymbol{\nu}_u$, the completed attention output becomes

$$\mathbf{o} = \frac{\bar{\mathbf{o}} + \sum_{u\in\mathcal{U}} \widetilde{m}(u)\boldsymbol{\nu}_u}{1 + \sum_{u\in\mathcal{U}} \widetilde{m}(u)}. \tag{36}$$

The same normalization is applied to the window masses used by the cumulative retention score. A window evaluated exactly receives its restricted mass divided by Z, while an unread Q window

receives $\widetilde{m}(u)/Z$. Thus read selection changes how a retained window is evaluated on the current query, but does not remove it from the signal used by future cache routing.

We use the mean log correction in Eq. (35) rather than a ratio of summed predicted and measured masses. The log-domain average corresponds to a multiplicative correction while reducing the influence of a single selected window with unusually concentrated attention. The quality of the resulting completed masses is evaluated directly in our diagnostic experiments.

J.4 GPU Execution

The algorithmic savings of the hierarchical cache depend on avoiding intermediate representations that reconstruct the traffic removed by compression. We therefore implement the scoring, Q-tier access, and cache migration paths directly around their window-level structure.

Prefill received-attention scoring. During prefill, the retention score requires the total attention received by each cached key. Let $x_{is} = \mathbf{q}_i^\top \mathbf{k}_s/\sqrt{d_h}$ denote the logit between query row i and key s, and let $L_i = \log \sum_u \exp(x_{iu})$ be the row-wise log-normalizer produced by the attention forward pass. The received attention of key s is

$$r_s = \sum_i \exp\left(x_{is} - L_i\right). \tag{37}$$

Equation (37) has the same key-wise reduction structure that appears in attention backward. We exploit this structure with a key-major Triton kernel that reuses the forward log-normalizers. Each program owns a disjoint block of keys and accumulates contributions only from query blocks that can attend to those keys under the causal mask. The $T \times S$ attention-probability matrix is never materialized. The resulting per-token values are reduced to the window scores used by the hierarchy.

Compact Q-tier scan. At each decode step, the compact representations are scanned before token-level Q attention. For every active Q window, the scan evaluates the two vector products required by its rank-1 representation and forms $\widehat{\ell}_{t,h}(w)$. A subsequent reduction performs the per-head normalization and GQA aggregation described in Section 4.3, producing the selected Q-slot indices for each KV head. Selection is represented by indirection into the existing Q storage. The selected windows are not first gathered into a new contiguous K/V tensor.

Fused mixed-precision decode. The decode kernel processes the native-precision states together with the selected Q windows. Packed INT2 codes and their quantization metadata are loaded directly from the Q store. K/V values are reconstructed in registers, and RoPE is applied to the reconstructed keys using their stored absolute positions. The expanded representation is consumed immediately by the online attention computation and is not written back to global memory.

The same attention pass also produces the received mass required for long-horizon scoring. Since decoding contains a single query row per step, per-window attention mass can be accumulated directly while the attention weights are produced. The kernel epilogue then applies the calibration and completion described in Appendix J.3, combining selected token-level contributions with the window-level contributions of the unread Q tier. Attention output and updated window statistics are therefore produced without a second attention or dequantization pass.

Layer-major cache migration. Routing occurs simultaneously across transformer layers with the same resolved window size and tier capacities. Rather than issuing a separate sequence of small operations for every layer, the implementation folds the layer dimension into the row dimension. For L layers and batch size B, the migration kernels operate over LB independent rows with common tensor geometry.

This batched path performs compaction of retained native windows, first-demotion quantization, reactivation of existing Q records, Q-to-F reconstruction, and release of permanently evicted records. The ranking remains layer-specific through $\bar{S}_t^\ell(w)$. Only the execution geometry is shared across layers.

J.5 Numerical and Positional Conventions

Several conventions are required to ensure that a cached state has a consistent numerical interpretation as it moves through the hierarchy.

Absolute positions. Window identity is tied to the original token positions. Surviving historical states are never renumbered after cache compaction. Pre-RoPE keys in the Q store are therefore always reconstructed using the same absolute positions at which they were first produced.

RoPE consistency. The native and low-bit paths use the same numerical convention when applying rotary embeddings. In particular, the rotary tables used after Q dequantization are constructed at the same KV-store datatype used by the native path. This prevents the represented key from changing solely because a window changes tier.

FP32 retention scores. Cumulative received-attention scores are stored in FP32. Prompt-time contributions can be substantially larger than individual decode-step updates, and lower-precision accumulation can otherwise suppress small later contributions through rounding.

Immutable persistent representations. The packed INT2 record, its quantization grid, and its compact read representation are written only on the first transition into $\mathcal{Q}$. Subsequent re-entry activates the same record. The anchors used to encode the compact representation are fixed as well, ensuring that previously stored residual codes retain the same interpretation throughout decoding.