KVRECODE: LOSSLESS KV CACHE COMPRESSION WITH LOCAL CODING AND BUFFER REUSE

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context generation places increasing memory demands on the key and value (KV) cache. However, compressing this cache losslessly requires repeated reconstruction before attention, making inference cost depend on both the stored representation and its runtime. To address this challenge, we introduce **KVRecode**, a lossless BF16 cache representation and runtime that combines **complete local exponent palettes** with **reusable reconstruction buffers**. KVRecode encodes exponents with fixed-width indices into page-local tables while preserving sign and mantissa bits, retaining pages in raw form when their exponent sets exceed the palette capacity. This construction enables exact recovery without calibration or per-value exception handling. As the cache grows, the runtime seals completed token segments and reconstructs successive layers into shared buffers, limiting reconstruction memory while preserving the existing attention backend. Across two 8B models and nine natural inputs at 4K, 8K, and 16K contexts, KVRecode reduces request latency for 512-token outputs by **10.45%** relative to SplitZip under matched resident-cache execution. It also uses **15.19%** less resident cache memory than Raw, including tails and reconstruction workspaces. KVRecode preserves cache contents and generated outputs exactly. These results demonstrate the benefits of coupling local coding with buffer reuse for efficient lossless cache access during generation.

1 INTRODUCTION

Large language models increasingly operate on extended documents, conversations, and reasoning traces (Bai et al., 2025). Their key and value cache grows with the retained context, making its memory cost central to efficient inference. PagedAttention improves cache allocation (Kwon et al., 2023), while Strata coordinates cache movement and scheduling across memory tiers (Xie et al., 2026). Compression reduces the space required for cached state, but a compressed cache must still supply usable values to attention throughout generation. This raises a question: **how can a resident KV cache save memory while remaining efficient to reconstruct at every decoding step?**

Recent methods reduce cache cost by adapting precision or retention. KIVI quantizes keys and values asymmetrically (Liu et al., 2024c), TurboQuant uses random rotations and scalar quantization (Zandieh et al., 2026), and PM-KVQ progressively adjusts precision during extended reasoning (Liu et al., 2026a). Token selection provides another route. SnapKV selects prompt positions from observed attention (Li et al., 2024), Ada-KV distributes eviction budgets across heads (Feng et al., 2025), and KVzip scores cached states through context reconstruction (Kim et al., 2025). These methods offer substantial opportunities for compression, but alter cached values or remove retained positions. We study a complementary requirement, *recovering every original BF16 cache word*, which makes the cost of exact reconstruction part of the compression problem.

Lossless coding provides a foundation for this requirement. DFloat11 exploits redundancy in BF16 weights to reduce model storage (Zhang et al., 2025). ECF8 develops exponent coding for FP8 weights (Yang et al., 2026). For KV activations, SplitZip combines a calibrated exponent codebook with sparse exceptions to accelerate transfer in disaggregated serving (Guo & Joshi, 2026). Our target is a cache that remains on the inference device and evolves during generation. Each generated token extends the cache, while subsequent attention calls repeatedly consume earlier entries. The benefits of compression therefore depend on both the cost of recovering cached values and the

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

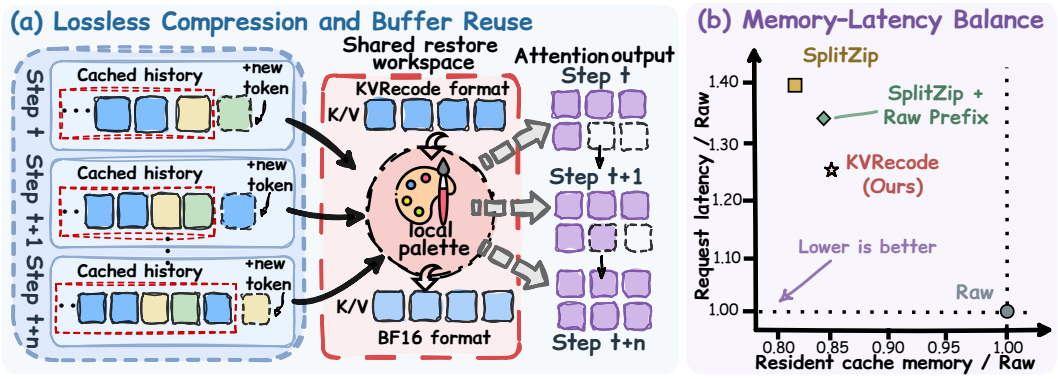


Figure 1: KVRecode overview. (a) Local exponent palettes support exact BF16 reconstruction into buffers reused across layers and decoding steps. (b) Resident cache memory and request latency for 512-token outputs under matched resident-cache execution, normalized to Raw.

memory needed to expose them to attention. This calls for a cache representation that is inexpensive to decode and a runtime that reuses reconstruction storage across layers and decoding steps.

In this paper, we introduce **KVRecode**, a lossless cache representation and runtime built around **complete local exponent palettes** and **reusable reconstruction buffers**. As shown in Figure 1(a), it encodes each eligible page using indices into its own exponent table, leaves other pages unchanged, and restores successive layers into a shared BF16 workspace before attention. This design enables exact recovery without calibration or per-value exception handling and limits the memory retained for reconstruction, while preserving the existing attention backend. Extensive evaluations on two 8B models across 4K, 8K, and 16K contexts demonstrate the efficiency of KVRecode without compromising exactness. As shown in Figure 1(b), KVRecode reduces request latency for 512-token outputs by **10.45%** relative to SplitZip under matched resident-cache execution and uses **15.19%** less resident cache memory than Raw. Compression remains bit-exact, with identical generated tokens in all primary comparisons and unchanged LongBench regression scores.

Our key contributions are as follows:

❶ ***Complete local exponent coding.*** We develop a calibration-free BF16 representation with fixed-width local indices and whole-page raw fallback for exact cache reconstruction.

❷ ***Reusable reconstruction for evolving caches.*** We combine sealed slabs, a raw append tail, and shared layer buffers to support repeated cache access through the existing attention backend.

❸ ***Empirical validation of efficient lossless caching.*** Experiments across two 8B models demonstrate reduced request latency relative to the evaluated SplitZip configurations and lower cache memory than Raw. These gains preserve exact cache recovery and leave outputs unchanged.

2 Related Work

Lossless floating-point representations. Format-aware codecs exploit exponent redundancy while retaining the original numerical representation. ZipNN separates exponents and applies Huffman coding for model storage and distribution (Hershcovitch et al., 2024). DFloat11 Huffman-codes BF16 weight exponents and uses hierarchical GPU lookup tables for online reconstruction (Zhang et al., 2025). These variable-length codes target symbol frequency. KVRecode instead represents every exponent in an eligible KV page with a fixed-width index into its complete local palette, falling back to raw storage for other pages.

Lossless KV coding and transfer. SplitZip combines calibrated four-bit exponent codes with sparse exceptions for KV transfer in disaggregated serving (Guo & Joshi, 2026). ExANS centers exponents within groups and entropy-codes them using block-local ANS models, enabling independent decoding without calibration (OpenLake, 2026). KVRecode shares their bitwise-preservation objective but uses complete palettes and whole-page fallback rather than sparse exceptions or entropy-

coded streams. For a growing device-resident cache reconstructed at every decoding step, the question is how coding interacts with updates and recurring reconstruction. Our matched resident-cache comparisons measure this cost, including tails and workspaces (Section 4), separately from the transfer-oriented evaluations of SplitZip and ExANS.

Reconstruction and compute integration. ECF8 reconstructs Huffman-coded FP8 weights into a shared layer-sized GPU buffer (Yang et al., 2026). ZipServ combines bitmap-based BF16 weight encoding with fused decompression and GEMM during decoding (Fan et al., 2026). Buffer reuse and fusion thus have established weight-side implementations. KVRecode couples reusable reconstruction with an evolving KV state: sealed slabs and a raw append tail supply bitwise-restored K/V to the existing attention backend.

Lossy KV-cache reduction. KIVI quantizes keys per channel and values per token (Liu et al., 2024c); KVQuant derives sensitivity-weighted nonuniform types (Hooper et al., 2024). H_2O retains recent and heavy-hitter tokens (Zhang et al., 2023), while SnapKV selects head-specific prompt positions (Li et al., 2024). CacheGen combines quantization and arithmetic coding for cache streaming (Liu et al., 2024b). These methods change values or retained positions, providing comparisons of quality, storage and latency under different preservation constraints. KVRecode retains every original BF16 word. Appendix G covers broader cache management.

3 METHODS

KVRecode separates the *stored representation* from the *attention input*. Complete page-local palettes encode BF16 words, sealed token slabs absorb cache growth, and reusable buffers reconstruct one layer's K/V for the existing attention backend (Figure 2). These three granularities serve different roles: pages bound the coding domain, slabs determine when to encode, and the workspace supports repeated consumption.

3.1 COMPLETE PAGE-LOCAL EXPONENT CODING

The representation must accommodate every BF16 word while exposing a fixed-width lookup for compact pages. Let $\boldsymbol{U} = (u_i)_{i=0}^{N-1}$ be the unsigned bit patterns of a contiguous K or V tensor. Our word-level invariant is

$$\mathrm{Decode}(\mathrm{Encode}(\boldsymbol{U})) = \boldsymbol{U}, \qquad \boldsymbol{U} \in \{0, \ldots, 2^{16} - 1\}^N. \tag{1}$$

This invariant includes NaN payloads, signed zeros, subnormals and infinities. We partition $\boldsymbol{U}$ into $G = \lceil N/P \rceil$ pages of $P = 256$ words, zero-padding the last page. Following exponent splitting (Guo & Joshi, 2026), integer operations isolate the exponent and sign and mantissa fields:

$$e_{p,i} = (u_{p,i} \gg 7) \,\&\, 255, \qquad a_{p,i} = ((u_{p,i} \gg 8) \,\&\, 128) \mid (u_{p,i} \,\&\, 127). \tag{2}$$

The byte $a_{p,i}$ preserves the sign and seven mantissa bits; only exponents are compressed.

Local completeness. A compact page stores its entire exponent set, so every word uses the same index width. Let $\mathcal{E}_p = \{e_{p,i} : 0 \le i < P\}$ include padded words, with sorted distinct values $(v_{p,0}, \ldots, v_{p,k_p-1})$ and $k_p = |\mathcal{E}_p|$. Eligibility and the sixteen-entry palette are defined jointly:

$$c_p = \mathbf{1}[k_p \le 16], \qquad \pi_{p,j} = v_{p,\min(j,k_p-1)}, \quad 0 \le j < 16, \quad c_p = 1. \tag{3}$$

Unused slots repeat the largest exponent. Within an eligible page, the rank $I_{p,i}$ locates the exponent in the palette; two adjacent four-bit indices share one byte:

$$I_{p,i} = \sum_{v \in \mathcal{E}_p} \mathbf{1}[v < e_{p,i}], \qquad q_{p,j} = (I_{p,2j} \ll 4) \mid I_{p,2j+1}, \quad 0 \le j < P/2. \tag{4}$$

The resulting payload contains 16 palette bytes, 256 bytes for sign and mantissa, and 128 index bytes. A page with $c_p = 0$ stores its original 512 bytes. Whole-page fallback preserves Equation 1 when the local exponent set exceeds the palette capacity.

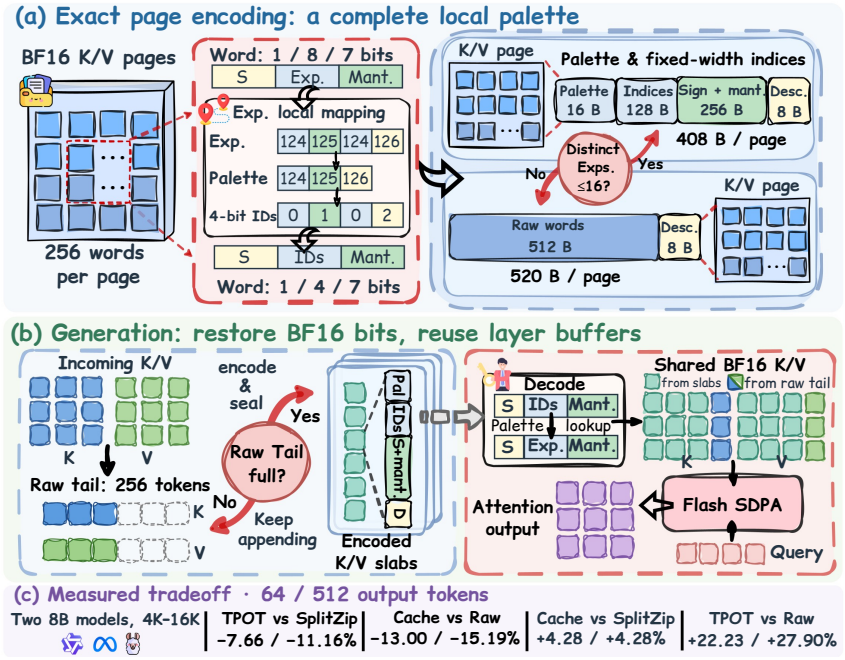


Figure 2: KVRecode representation and runtime. (a) A complete local exponent palette enables compact encoding, with raw fallback preserving every BF16 word. (b) Sealed slabs and the active tail are restored into a shared workspace before attention. Pages contain 256 words, and tails contain 256 tokens. Page totals include descriptors, with object overhead and workspaces charged separately. (c) Measured TPOT and resident-allocation changes.

3.2 Addressable Reconstruction and Storage Cost

An explicit descriptor makes each variable-length payload directly addressable. The encoder computes page sizes s_p and exclusive offsets o_p, then combines the offset and raw-mode flag in an eight-byte descriptor d_p:

$$s_p = 400c_p + 512(1 - c_p), \quad o_p = \sum_{h=0}^{p-1} s_h, \quad d_p = o_p \mid \big((1 - c_p) \ll 63\big). \tag{5}$$

Offsets are relative to the payload region; the descriptor's high bit marks raw mode. Compact reconstruction inverts the local code through nibble extraction, palette lookup and bit-field assembly:

$$\begin{aligned} \hat{I}_{p,i} &= \big(q_{p,\lfloor i/2 \rfloor} \gg [4(1 - (i \bmod 2))]\big) \,\&\, 15, \qquad \hat{e}_{p,i} = \pi_{p,\hat{I}_{p,i}}, \\ \hat{u}_{p,i} &= ((a_{p,i} \,\&\, 128) \ll 8) \mid (\hat{e}_{p,i} \ll 7) \mid (a_{p,i} \,\&\, 127). \end{aligned} \tag{6}$$

Since $I_{p,i} < k_p \leq 16$, unpacking recovers the rank and hence $\hat{e}_{p,i} = e_{p,i}$. Reassembling the three disjoint fields gives $\hat{u}_{p,i} = u_{p,i}$ without floating-point arithmetic. Raw pages copy unchanged; trimming padding establishes Equation 1 for intact objects with valid metadata.

Descriptor-inclusive storage. With $P_c = \sum_p c_p$ compact and $P_r = G - P_c$ raw pages, the allocated blob includes the reserved prefix, aligned descriptors and payloads:

$$B_{\text{blob}} = 64 + \text{align}_{16}(8(P_c + P_r)) + 400P_c + 512P_r, \tag{7}$$

Here $\text{align}_{16}(x) = 16\lceil x/16 \rceil$ (Table 6). For many full pages, compact and raw ratios approach $408/512 = 0.796875$ and $520/512 = 1.015625$, respectively: raw fallback can expand storage. Palettes are page-local; exception values, bitmaps and global codebooks contribute zero bytes.

3.3 Streaming Updates and Reusable Reconstruction

Sealed slabs separate append-time encoding from repeated reconstruction. Layer ℓ maintains $\mathcal{S}_\ell = (\mathcal{B}_\ell, R_\ell, r)$: ordered encoded K/V slab pairs, a raw tail, and its token count $r < T$, where $T = 256$.

Pages contain words, whereas tails contain tokens across heads and channels. Prefill seals the largest prefix divisible by T as one slab pair, copies the remainder into the tail, and supplies incoming K/V directly to prefill attention.

For a new K/V pair x, write $R^+ = R_\ell \| x$, with $\|$ denoting token-axis concatenation. Each append either extends the tail or seals it; Encode acts separately on K and V:

$$(\mathcal{B}'_\ell, R'_\ell, r') = \begin{cases} (\mathcal{B}_\ell, R^+, r+1), & r+1 < T, \\ (\mathcal{B}_\ell \| \operatorname{Encode}(R^+), \varnothing, 0), & r+1 = T. \end{cases} \tag{8}$$

Emptying the logical tail retains its allocation. Reconstruction $\mathcal{R}$ decodes slabs into reusable flat scratch, copies them to their head/token positions in a shared full-layer K/V workspace, and appends the active tail. Attention then consumes the reconstructed views:

$$(\widehat{\mathbf{K}}_\ell, \widehat{\mathbf{V}}_\ell) = \mathcal{R}(\mathcal{S}_\ell), \qquad \mathbf{Y}_\ell = \mathrm{SDPA}(\mathbf{Q}_\ell, \widehat{\mathbf{K}}_\ell, \widehat{\mathbf{V}}_\ell). \tag{9}$$

Here $\mathbf{Q}_\ell$ is the model-native query after normalization and RoPE, and SDPA retains the model's attention settings. Equation 1 and the ordered append transition preserve every stored K/V word. The same workspace is reused across layers and decoding steps.

3.4 Online execution and allocation accounting

In KVRecode, flat scratch is allocated on first decode; tails and the shared K/V workspace are allocated at construction. For L layers, H K/V heads, head dimension d, token capacity C and encoded objects $\mathcal{O}(t)$, persistent device storage is

$$B_{\text{resident}}(t) = \sum_{o \in \mathcal{O}(t)} B_{\text{blob}}(o) + 4LHTd + 4HCd + 2HCd\,\chi_{\text{flat}}(t), \tag{10}$$

The last three terms charge tails, the shared K/V workspace and flat scratch. The indicator χ_{flat} changes from zero to one at scratch allocation. Factors of two account for BF16 bytes and both K and V; Raw capacity is $4LHCd$ bytes.

Online encoding computes page sizes and prefix-sum offsets on the GPU, reads the final payload size to the host, and allocates the blob before packing. This size read synchronizes and remains inside request timing. Size, cumulative-sum and offset arrays, together with contiguous input copies, contribute transient allocations.

Resident objects carry typed device storage and logical metadata. Explicit archive export performs full-body transfer and SHA256 authentication; import validates the archive before constructing a resident object. Thus archive processing is separated from the repeated online path, while payload-allocation synchronization remains timed. Appendix D.1 measures the per-encode archive-processing cost.

4 Experiments

Lossless cache compression must reduce storage while repeatedly supplying exact K/V to attention. We evaluate this tradeoff between storage and latency during generation, then examine whether it persists through cache updates and natural stopping and how runtime choices affect it.

4.1 Experimental Setup

Models and execution. We use Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen3-8B (Yang et al., 2025) with BF16, batch-one greedy decoding and Flash SDPA. They have 32/36 layers and eight K/V heads of dimension 128. Primary methods share the forward path, including normalization and RoPE, under PyTorch 2.8.0, Transformers 4.57.1, Triton 3.4.0 and Tokenizers 0.22.2.

Experiments run on NVIDIA RTX 4090D hardware. We use five paired process sessions with randomized method order.

Comparators. *Raw* supplies the uncompressed BF16 reference in the common cache wrapper. *SplitZip* uses its official codec and model-specific calibration codebook in our resident-cache adapter. *SplitZip + Raw Prefix*, our hybrid control, keeps the first four Llama or six Qwen layers raw and compresses the remainder; prefix lengths are development-selected. *KVRecode* uses the local palettes and lazy flat scratch (Section 3). Compressed methods share the tail and reconstruction policy.

Data and generation lengths. The primary panel contains all nine eligible unused contents from the frozen LongBench interface (Bai et al., 2024): six HotpotQA and three QMSum (Zhong et al., 2021). Selection uses prior-use history and token length before observing performance. Each naturally reaches 16,384 tokens with both tokenizers and supplies fixed 4,096-, 8,192- and 16,384-token views without repetition or concatenation.

Calibration and development data are separate from this panel. Quality and natural-stopping tests reuse consumed identities for regression. The earlier ExANS batch uses six HotpotQA/QMSum/LCC contents; SnapKV/KIVI development uses one Qasper identity per length (Dasigi et al., 2021).

Fixed 64- and 512-token outputs separate generation length from EOS behavior, yielding $9 \times 2 \times 3 \times 2 \times 4 \times 5 = 2{,}160$ timed requests. Each model/method/session process runs two 512-token warmups per input length, totaling 240 warmups across forty processes.

Metrics and paired statistics. Let t_0 be the host timestamp immediately before cache construction and input transfer, and t_j the synchronized timestamp at which generated token j is available to the host. The endpoints are

$$T_{\mathrm{TTFT}} = t_1 - t_0, \qquad T_{\mathrm{TPOT}} = \frac{t_O - t_1}{O - 1}, \qquad T_{\mathrm{request}} = t_O - t_0, \tag{11}$$

where O is the output length. Timing includes online encode/decode, allocation, copies, attention, synchronization and the generation loop. Model loading, tokenization, queueing, detokenization and archive export are excluded; cache clearing and initial synchronization precede t_0. Thus request latency measures engine generation, while TTFT and TPOT distinguish its initial and repeated costs.

The primary estimator weights tasks equally. For metric T and output count O, $T^{(O)}_{a,qimns}$ denotes method a, task q, identity i, model m, input length n and session s. Against reference b, define $z^{(b,O)}_{qimns} = \log(T^{(O)}_{\mathrm{KVRecode},qimns}/T^{(O)}_{b,qimns})$ and compute

$$\bar{z}_{b,O} = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}} \frac{\sum_{i=1}^{N_q}\sum_{m,n,s} z^{(b,O)}_{qimns}}{N_q|\mathcal{M}||\mathcal{N}||\mathcal{S}|}, \qquad \Delta_{b,O} = 100\big(\exp(\bar{z}_{b,O}) - 1\big)\%. \tag{12}$$

Here $|\mathcal{Q}| = 2$, $N_q \in \{6, 3\}$ and $(|\mathcal{M}|, |\mathcal{N}|, |\mathcal{S}|) = (2, 3, 5)$. We analyze output lengths separately and use 10,000 percentile bootstrap replicates (seed 630020). Identity draws remain linked across models and lengths; session draws remain linked across methods and identities.

Memory accounting and exactness. Resident GPU allocation counts all layers simultaneously, including encoded tensors, descriptors, alignment, raw pages, allocated tails, codebooks, shared K/V workspace and flat scratch. The denominator is Raw *capacity* for input plus requested output tokens; live-token bytes are recorded separately. Whole-model allocator allocated/reserved peaks are recorded independently.

The lifecycle audit captures temporary update allocations. With pre-update resident bytes C_e, total allocated bytes A_e and peak allocated bytes P_e at update e, the cache-attributed bound is

$$B_{\mathrm{lifecycle}} = \max_e \big\{C_e + \max(0, P_e - A_e)\big\}. \tag{13}$$

This bound excludes common model activations and shared incoming K/V, with a diagnostic threshold of 0.90 times Raw capacity.

Codec gates test all 65,536 BF16 patterns, special values, boundary lengths, palette cardinalities, and exception-density cases. Model gates compare incoming post-RoPE K/V, output tokens, and final logits to a native raw path. Formal runs compare output hashes within all four-method groups. The separate lifecycle profiles cover all nine identities, two models, three lengths, and two output counts, giving 108 requests.

4.2 Main Results

We compare KVRecode with Raw and SplitZip on resident cache memory and generation latency, and separately evaluate latency against ExANS.

Table 1: Primary latency and cache comparison. (a) Geometric-mean ratios to Raw, including tails and workspaces; bold numbers mark compressed-method minima. (b) Ours minus reference (ms, %); bold numbers mark latency decreases.

(a) Method comparison relative to Raw

Method	Output	TTFT ↓	TPOT ↓	Request ↓	Cache ↓
Raw BF16	64	1.0000	1.0000	1.0000	1.0000
SplitZip	64	1.0124	1.3237	1.1884	**0.8343**
SplitZip + Raw Prefix	64	**1.0116**	1.2787	1.1625	0.8632
KVRecode	64	1.0132	**1.2223**	**1.1272**	0.8700
Raw BF16	512	1.0000	1.0000	1.0000	1.0000
SplitZip	512	1.0114	1.4397	1.3982	**0.8133**
SplitZip + Raw Prefix	512	**1.0108**	1.3762	1.3408	0.8452
KVRecode	512	1.0122	**1.2790**	**1.2521**	0.8481

(b) Absolute and relative latency changes

Method	Reference	Output	TTFT ↓		TPOT ↓		Request ↓	
			Δt (ms)	Δ%	Δt (ms)	Δ%	Δt (ms)	Δ%
Ours	SplitZip	64	+0.86	+0.08	**−2.38**	**−7.66**	**−166.51**	**−5.15**
	SplitZip	512	+0.84	+0.07	**−3.78**	**−11.16**	**−1947.97**	**−10.45**
	SplitZip + Raw Prefix	64	+1.81	+0.16	**−1.32**	**−4.41**	**−96.09**	**−3.04**
	SplitZip + Raw Prefix	512	+1.55	+0.14	**−2.28**	**−7.06**	**−1182.88**	**−6.61**
	Raw	64	+14.72	+1.32	+5.22	+22.23	+346.18	+12.72
	Raw	512	+13.60	+1.22	+6.56	+27.90	+3362.19	+25.21

Memory and latency benefits. As listed in Table 1(a), KVRecode reduces resident cache memory relative to Raw by 13.00% for 64-token outputs, increasing to 15.19% for 512-token outputs. As listed in Table 1(b), the request latency reduction relative to SplitZip grows from 5.15% to 10.45%. At 512 outputs, KVRecode also reduces latency by 6.61% relative to SplitZip + Raw Prefix at a similar resident allocation. This comparison highlights the value of complete local palettes, whose fixed-width indices support efficient reconstruction without per-value exception handling.

Benefits across models and generation lengths. We examine whether the aggregate gains extend across model and context settings. As shown in Figure 3(a,c), KVRecode retains memory savings over Raw throughout the evaluated settings. As shown in Figure 3(b,d), it achieves lower TPOT than SplitZip at all three context lengths on both models for 512-token outputs. As listed in Table 1(b), the TPOT reduction relative to SplitZip increases from 7.66% at 64 outputs to 11.16% at 512 outputs, while TTFT changes by less than 0.1%. This phase contrast locates the benefit in repeated cache access, where cheaper reconstruction contributes throughout generation.

Memory savings through cache updates. We include transient allocations to test whether savings persist while the cache evolves. As listed in Table 2, the maximum measured lifecycle bound is 89.35% of Raw capacity, including transient allocations. The bounds stay close to resident allocations, indicating that tail sealing and reconstruction preserve the storage benefit. This behavior is consistent with retaining tail storage and sharing reconstruction buffers across layers and steps. Resident ratios also decrease from 64 to 512 outputs on both models as fixed-size tails are amortized over larger cache capacities.

Latency relative to entropy coding. A separate batch compares an eager-scratch variant of KVRecode with ExANS under shared-workspace execution. As listed in Table 3, request latency decreases by 68.57% and 75.76% at 64 and 512 outputs, respectively, with matching outputs across all 1,440 timed requests. TPOT decreases by 77.19% to 78.04%, compared with approximately 15% for TTFT. The larger benefit during decoding reinforces the value of a lightweight local lookup path for repeatedly consumed caches, consistent with the component timings in Appendix Table 13.

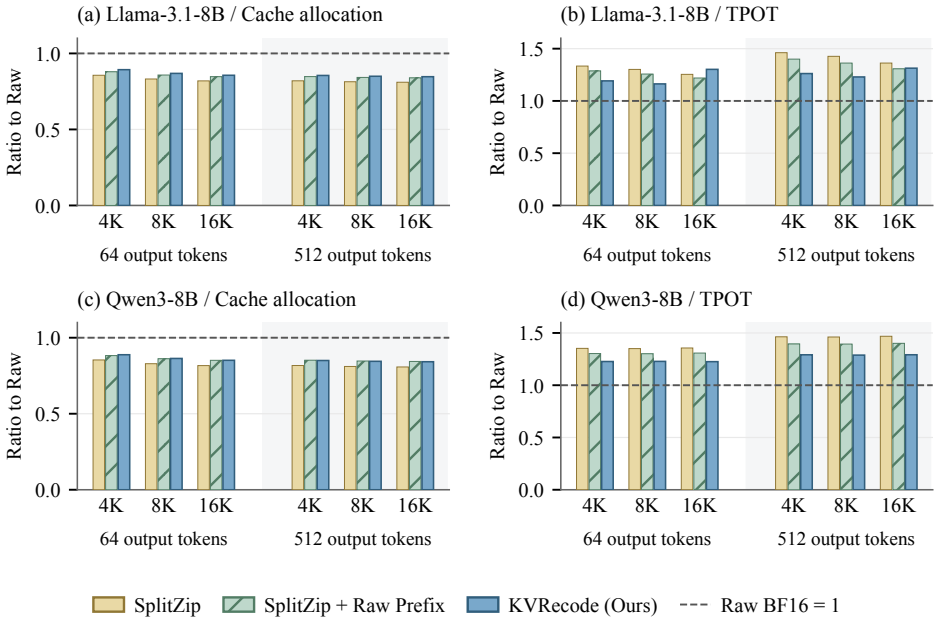


Figure 3: Resident allocation and TPOT relative to Raw (dashed line), by model, input length and output length; lower is better. Grouped bars separate storage from latency, with all axes starting at zero. Storage uses arithmetic means of allocated-cache/capacity ratios; TPOT uses geometric means of paired ratios over nine identities and five sessions per cell. Appendix Table 15 provides primary uncertainty estimates.

Table 2: Cache allocation relative to Raw capacity. Resident ranges span input-length means; lifecycle maxima include tails and workspaces.

Model	Output	Resident ↓	Saving (%) ↑	Lifecycle max. ↓
Llama-3.1-8B	64	0.8562–0.8931	10.69–14.38	0.8935
Llama-3.1-8B	512	0.8468–0.8551	14.49–15.32	0.8551
Qwen3-8B	64	0.8514–0.8881	11.19–14.86	0.8885
Qwen3-8B	512	0.8420–0.8501	14.99–15.80	0.8501

Table 3: Latency changes (%) relative to ExANS for KVRecode with eager scratch.

Output	TTFT ↓	TPOT ↓	Request ↓
64	**−15.60** [-15.90, -15.39]	**−78.04** [-78.12, -77.96]	**−68.57** [-68.64, -68.47]
512	**−15.37** [-15.49, -15.29]	**−77.19** [-77.24, -77.12]	**−75.76** [-75.82, -75.69]

4.3 Output Preservation and Natural Stopping

We evaluate whether the efficiency gains preserve generation and task performance. As listed in Table 4, KVRecode matches Raw and SplitZip on all eight LongBench tasks for both models, covering question answering, summarization and code completion. All 2,160 primary requests also agree token for token within their matched comparison groups. These results show that exact cache recovery preserves both generated sequences and measured task quality.

Natural-stopping evaluation further confirms unchanged token sequences, output lengths and stop reasons, while retaining the latency benefit relative to SplitZip (Appendix Table 7).

4.4 Runtime Analysis

We isolate how archive processing and page size affect inference cost and storage (Table 5 and Figure 4).

Table 4: LongBench quality (0–100; higher is better). Ours and SplitZip match Raw in every cell; R-L denotes ROUGE-L and Sim. denotes code similarity.

Task	Metric ↑	Llama-3.1-8B			Qwen3-8B		
		Raw	SplitZip	**Ours**	Raw	SplitZip	**Ours**
Qasper	F1	44.75	44.75	44.75	54.83	54.83	54.83
HotpotQA	F1	54.29	54.29	54.29	62.13	62.13	62.13
TriviaQA	F1	90.00	90.00	90.00	90.00	90.00	90.00
MultiNews	R-L	26.00	26.00	26.00	24.07	24.07	24.07
QMSum	R-L	24.25	24.25	24.25	23.69	23.69	23.69
SAMSum	R-L	44.73	44.73	44.73	46.36	46.36	46.36
LCC	Sim.	74.80	74.80	74.80	75.12	75.12	75.12
RepoBench-P	Sim.	62.80	62.80	62.80	61.00	61.00	61.00
Macro average	Mixed	52.70	52.70	52.70	54.65	54.65	54.65

Archive processing. We move per-encode archive transfers and authentication to explicit export while holding the codec, resident allocation and eager-scratch policy fixed. Table 5 shows a **64.42% to 66.75%** decrease in TTFT and a 6.77% to 50.63% decrease in request latency, while TPOT changes by at most 0.25%. Separating archive processing from online encoding primarily improves first-token latency, complementing the codec's recurring decode savings.

Table 5: Archive-processing ablation with identical codec and storage. Percentage changes compare explicit export with per-encode authentication and data transfers.

Input	Output	TTFT Δ%	TPOT Δ%	Request Δ%
4K	64	**−66.75** [-66.91, -66.58]	+0.25	**−32.94**
4K	512	**−66.73** [-67.08, -66.45]	+0.09	**−6.77**
16K	64	**−64.42** [-65.99, -63.02]	+0.00	**−50.63**
16K	512	**−65.13** [-66.93, -63.30]	**−0.25**	**−20.60**

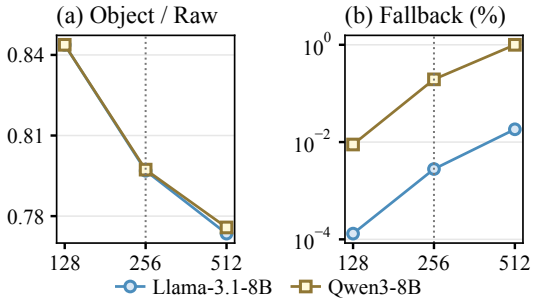


Figure 4: Page-size diagnostic (BF16 words per page) on one frozen 16K input per model. Allocation includes metadata relative to logical Raw, excluding runtime tails and decode workspace. Runtime measurements use 256-word pages.

Page size. Figure 4 compares 128-, 256- and 512-word pages on one frozen 16K input per model. Larger pages reduce object allocation despite increasing raw fallback: lower per-word palette and descriptor costs outweigh the additional fallback in these two cases. All runtime results use 256-word pages; latency at the other sizes is unmeasured. Appendix B gives the input-selection rule and exact allocation ratios.

5 Conclusion & Limitation

KVRecode combines complete page-local exponent palettes with reusable reconstruction buffers. It reduces resident cache allocation while incurring less generation overhead than the evaluated SplitZip configurations; the eager-scratch configuration also reduces latency relative to the ExANS adapter. The two-model system, quality and stopping measurements preserve generated outputs. Local representation and runtime reuse together make the tradeoff between storage and latency of lossless caches explicit and measurable. The current representation is specialized for BF16 caches. Extending it to caches that natively use other floating-point formats would require adapting the field decomposition, palette layout and bit-packing rules. Supporting mixed-format caches would additionally require format-aware reconstruction and correctness checks. A unified lossless representation across numerical formats is left for future work.

Reproducibility Statement

The artifact retains model/input versions, calibration, source, request/token timestamps, memory ledgers and analysis scripts. Tables are generated from the retained result files. Appendices A and E specify correctness coverage and failed/unrun conditions. Public hosting and anonymized packaging remain preparation steps.

AI Use Statement

Generative AI tools assisted language editing and LaTeX typesetting to improve the clarity and readability of the manuscript. The authors reviewed the AI-assisted material, verified the scientific content, and take full responsibility for the manuscript.

Ethics Statement

This work evaluates bitwise-lossless KV-cache compression using existing pretrained language models and benchmark datasets. The method preserves cached values and does not address biases or harmful outputs of the underlying models. Deployment and redistribution should follow the applicable model and dataset licenses and privacy requirements.

References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, 2024. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

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 (Volume 1: Long Papers)*, pp. 3119–3137. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025. doi: 10.18653/v1/2025.acl-long.183. URL https://aclanthology.org/2025.acl-long.183/.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A Dataset of Information-Seeking Questions and Answers Anchored in Research Papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, 2021. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365/.

Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get More with LESS: Synthesizing Recurrence with KV Cache Compression for Efficient LLM Inference. In *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11437–11452. PMLR, 2024. URL https://proceedings.mlr.press/v235/dong24f.html.

Ruibo Fan, Xiangrui Yu, Xinglin Pan, Zeyu Li, Weile Luo, Qiang Wang, Wei Wang, and Xiaowen Chu. ZipServ: Fast and memory-efficient LLM inference with hardware-aware lossless compression. In *Proceedings of the 31st ACM International Conference on Architectural*

Support for Programming Languages and Operating Systems, Volume 2, pp. 2264–2280. Association for Computing Machinery, 2026. doi: 10.1145/3779212.3790250. URL https://doi.org/10.1145/3779212.3790250.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113152–113188, 2025. doi: 10.52202/085713-3773. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a40ff56daab9f4808b1e18350c8a11ce-Abstract-Conference.html.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783v3. Version 3, November 23, 2024.

Yipin Guo and Siddharth Joshi. SplitZip: Ultra fast lossless KV compression for disaggregated LLM serving, 2026. URL https://arxiv.org/abs/2605.01708v3. Version 3, June 23, 2026.

Moshik Hershcovitch, Andrew Wood, Leshem Choshen, Guy Girmonsky, Roy Leibovitz, Ilias Ennmouri, Michal Malka, Peter Chin, Swaminathan Sundararaman, and Danny Harnik. ZipNN: Lossless compression for AI models, 2024. URL https://arxiv.org/abs/2411.05239v2. Version 2, revised June 2025.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun S. Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 1270–1303, 2024. doi: 10.52202/079017-0040. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/028fcbcf85435d39a40c4d61b42c99a4-Abstract-Conference.html.

Shipeng Hu, Guangyan Zhang, Yuqi Zhou, Yaya Wei, Ziyan Zhong, and Jike Chen. Bidaw: Enhancing Key-Value Caching for Interactive LLM Serving via Bidirectional Computation–Storage Awareness. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 101–116, 2026. URL https://www.usenix.org/conference/fast26/presentation/hu-shipeng.

Jushi Kai, Yixuan Wang, Boyi Zeng, Haoli Bai, Bo Jiang, Ziwei He, and Zhouhan Lin. FreqKV: Key-Value Compression in Frequency Domain for Context Window Extension. In *International Conference on Learning Representations*, pp. 105568–105587, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ac4c13a5c6ec1bba68c9ce1d908321d2-Abstract-Conference.html.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-Agnostic KV Cache Compression with Context Reconstruction. In *Advances in Neural Information Processing Systems*, volume 38, pp. 167563–167591, 2025. doi: 10.52202/085713-5585. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f4eaa4b8f2d08edb3f0af990d56134ea-Abstract-Conference.html.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172, 2024. URL https://www.usenix.org/conference/osdi24/presentation/lee.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV Cache-Centric Analysis of Long-Context Methods. In *International Conference on Learning Representations*, pp. 66063–66093, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a540b17fb2295c736d5afd6c507acf66-Abstract-Conference.html.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM Knows What You are Looking for Before Generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970, 2024. doi: 10.52202/079017-0722. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. MiniCache: KV Cache Compression in Depth Dimension for Large Language Models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 139997–140031, 2024a. doi: 10.52202/079017-4443. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/fd0705710bf01b88a60a3d479ea341d9-Abstract-Conference.html.

Tengxuan Liu, Shiyao Li, Jiayi Yang, Tianchen Zhao, Feng Zhou, Xiaohui Song, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. PM-KVQ: Progressive Mixed-precision KV Cache Quantization for Long-CoT LLMs. In *International Conference on Learning Representations*, pp. 107881–107900, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/afb8caec018d3c8f6ef8b81fa52386fe-Abstract-Conference.html.

Yang Liu, Yunfei Gu, Liqiang Zhang, Chentao Wu, Guangtao Xue, Jie Li, Minyi Guo, Junhao Hu, and Jie Meng. CacheSlide: Unlocking Cross Position-Aware KV Cache Reuse for Accelerating LLM Serving. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 83–99, 2026b. URL https://www.usenix.org/conference/fast26/presentation/liu-yang.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024b. doi: 10.1145/3651890.3672274. URL https://doi.org/10.1145/3651890.3672274.

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

OpenLake. ExANS: Achieving 600GB/s+ decode throughput at 1.51x lossless compression. Official technical blog, August 2026. URL https://www.theopenlake.com/blog/exans-lossless-gpu-compression-for-bf16-kv-cache. Published August 4, 2026. Software snapshot: commit 010493bb7865cb35cccd4c4c8ca9966fac26efa9.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132, 2024. doi: 10.1109/ISCA59077.2024.00019. URL https://doi.org/10.1109/ISCA59077.2024.00019.

Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. ThinKV: Thought-Adaptive KV Cache Compression for Efficient Reasoning Models. In *International Conference on Learning Representations*, pp. 110072–110106, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/b2b502c3629beadda06311386d2c6f73-Abstract-Conference.html.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision. In *Advances in Neural Information Processing Systems*, volume 37, pp. 68658–68685, 2024. doi: 10.52202/079017-2193. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/7ede97c3e082c6df10a8d6103a2eebd2-Abstract-Conference.html.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31094–31116. PMLR, 2023. URL https://proceedings.mlr.press/v202/sheng23a.html.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference. In *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911. PMLR, 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

Haoyi Wu and Kewei Tu. Layer-Condensed KV Cache for Efficient Inference of Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11175–11188, 2024. doi: 10.18653/v1/2024.acl-long.602. URL https://aclanthology.org/2024.acl-long.602/.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory. In *Advances in Neural Information Processing Systems*, volume 37, pp. 119638–119661, 2024a. doi: 10.52202/079017-3801. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. In *International Conference on Learning Representations*, pp. 21875–21895, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5e5fd18f863cbe6d8ae392a93fd271c9-Abstract-Conference.html.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads. In *International Conference on Learning Representations*, pp. 37228–37253, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5c1ddd2e59df46fd2aa85c833b1b36ed-Abstract-Conference.html.

Zhiqiang Xie, Ziyi Xu, Mark Zhao, Yuwei An, Vikram Sharma Mailthody, Scott Mahlke, Michael Garland, and Christos Kozyrakis. Strata: Hierarchical Context Caching for Long Context Language Model Serving. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pp. 1–16, 2026. URL https://www.usenix.org/conference/osdi26/presentation/xie-zhiqiang.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388v1.

Zeyu Yang, Tianyi Zhang, Jianwen Xie, Chuan Li, Zhaozhuo Xu, and Anshumali Shrivastava. To Compress or Not? Pushing the Frontier of Lossless GenAI Model Weights Compression with Exponent Concentration. In *International Conference on Learning Representations*, pp. 53467–53487, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/57c3fef21638eb9d1d67d32ba0a6d1af-Abstract-Conference.html.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. TurboQuant: Online Vector Quantization with Near-optimal Distortion Rate. In *International Conference on Learning Representations*, pp. 56418–56439, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5c802ef38ab6e366c2ea06eee554c088-Abstract-Conference.html.

Tianyi Zhang, Mohsen Hariri, Shaochen Henry Zhong, Vipin Chaudhary, Yang Sui, Xia Hu, and Anshumali Shrivastava. 70% size, 100% accuracy: Lossless LLM compression for efficient GPU inference via dynamic-length float (DFloat11). In *Advances in Neural Information Processing Systems*, volume 38, pp. 98966–98994, 2025. doi: 10.52202/085713-3306. URL https://papers.nips.cc/paper_files/paper/2025/hash/8e8b4449c80879215ff2ee47b6dd01a6-Abstract-Conference.html.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.814. URL https://aclanthology.org/2024.acl-long.814/.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710, 2023. doi: 10.52202/075280-1506. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract.html.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H. Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient Execution of Structured Language Model Programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. QMSum: A New Benchmark for Query-based Multi-domain Meeting Summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5905–5921, 2021. doi: 10.18653/v1/2021.naacl-main.472. URL https://aclanthology.org/2021.naacl-main.472/.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

APPENDIX CONTENTS

Appendix A. Protocol and Correctness Details 15
Appendix B. Page-Size Diagnostic Details 16
Appendix C. Task-Level Quality Regression 16
Appendix D. External Comparators and Admission Boundaries 16
Appendix E. Additional Evaluation Details 18
Appendix F. Paired and Identity-Level Timing and Allocation 18
Appendix G. Extended Related Work 25

A PROTOCOL AND CORRECTNESS DETAILS

Artifact identifiers. The archived method identifiers `E63` and `EH63` correspond to SplitZip and SplitZip + Raw Prefix, respectively. The latter is our hybrid configuration of the official codec, not a separately published method. These identifiers remain unchanged in the raw records.

Units of replication. The primary request count factors into nine identities, two models, three input lengths, two output counts, four methods, and five sessions. Each method has 540 requests and each four-method comparison group contains four matched requests. Within one output regime, each reference comparison has 270 pairs. The primary estimator weights HotpotQA and QMSum equally despite their six-versus-three identity counts. It then weights identities equally within each task, with balanced models, lengths, and sessions. The bootstrap keeps a sampled identity linked across models and lengths and uses a shared sampled session index across methods. The narrow timing intervals describe this panel and process variation, not the diversity of arbitrary long-context workloads.

Table conventions. Table 1(a) uses task-balanced geometric means for latency and descriptive geometric means for resident allocation. Panel (b) reports Ours minus reference: Δt is the difference of task-balanced geometric-mean latencies in milliseconds, and $\Delta\%$ follows Equation 12. Bold marks compressed-method minima in panel (a) and latency decreases in effect tables, with intervals wholly below zero when shown. Positive changes denote increases. Appendix Table 15 retains all primary intervals; the two SplitZip TTFT intervals cross zero, and equivalence was not tested.

Aggregation sensitivity. Weighting all 540 common-runtime KVRecode/SplitZip pairs equally gives request ratios of 0.9482/0.8952 and TPOT ratios of 0.9229/0.8881 at 64/512 outputs, preserving the direction of the task-equal primary comparison. This descriptive check retains both codec behavior and allocation timing.

Source versions. The paper description of SplitZip follows arXiv version 3, while the reproduced codec is frozen at repository commit `caac0872` (full identifier in the artifact). The codec snapshot predates that paper version, and the two are recorded separately. ExANS's inspected software snapshot begins `010493bb`; its reference is an official technical blog rather than a verified peer-reviewed publication. Model identities are established by the frozen model-file inventories, independently of their model-family citations. The LongBench scorer is the retained official source snapshot used by the experiment runner.

Model-native queries. The runner invokes the complete pretrained model with caching enabled and the Flash SDPA backend selected. It selects the next token from the model logits and synchronizes before recording its timestamp. Validation intercepts K/V at cache update, after model positional transformations, and compares their bitwise hashes and positions to a native raw run. The model gates generate 513 tokens so that aligned prefixes exercise two full 256-token tail seals. The formal model forward path uses the same normalization and rotary transformations.

Codec coverage. The frozen local-palette codec gate reports 22 cases, including the exhaustive sixteen-bit pattern set and special/boundary cases. The SplitZip gate applies each model's calibration to the exhaustive set, special words, lengths 0/1/2/3/255/256/257/511/512/513/1025, and no/single/all-exception arrays. It records thirty passing nonempty official-codec cases and two unsupported empty-input cases. The adapter is separately checked for its supported aligned shapes;

an untested adapter shape is not inferred from official-codec success. Codec gates, native-model comparisons, formal output agreement, and quality regression provide separate levels of coverage.

Resident allocation denominator. For L layers, H_{kv} K/V heads, head dimension d, and allocated token capacity C, Raw reserves

$$B_{\text{raw,capacity}} = L \cdot 2 \cdot H_{\text{kv}} \cdot C \cdot d \cdot 2 \text{ bytes}. \tag{14}$$

The first factor of two represents K and V, and the second represents BF16 bytes per word. Capacity is input length plus the requested output count. The last output token need not yet have a cached K/V entry, so capacity and logical live-token bytes are not identical. Resident ratios include full allocated tails and workspaces even when their active slices are smaller. Descriptive storage means are labeled separately from the task-equal latency estimand.

Codec sizing versus runtime peaks. Equation 7 describes one blob's physical storage. It includes descriptor padding and the reserved prefix, but excludes input tensors and transient encode arrays. During an update, the profiler records allocator-visible peaks around allocation, seal, decode, and copies, with a separate whole-model allocated/reserved peak. This is why the approximately 0.797 page-object ratio at 16K cannot be used as the final resident or model-memory ratio.

B Page-Size Diagnostic Details

The diagnostic in Figure 4 selects the lexicographically first frozen 16K identity per model and collects real post-RoPE K/V. For 128-, 256- and 512-word pages, the object-allocation ratios relative to logical Raw are 0.8438/0.7969/0.7735 for Llama and 0.8438/0.7973/0.7758 for Qwen. These ratios include codec metadata but exclude runtime tails and decode workspace. The 128- and 512-word CUDA kernels are absent; runtime measurements use the 256-word format.

C Task-Level Quality Regression

The same task-specific preprocessing and frozen official scorer are used for Raw, KVRecode, and SplitZip. Qasper, HotpotQA, and TriviaQA use QA F1; MultiNews, QMSum, and SAMSum use ROUGE-L; LCC and RepoBench-P use code similarity. The macro average gives each of the eight task interfaces equal weight even though their scores have different meanings. The unchanged per-task outputs are the basis for the equality conclusion. No quality confidence interval is interpreted as a numerical-equivalence margin.

The natural-stopping panel uses eight consumed identities per model and five paired process sessions. The RULER subset uses four records per model/task/length and four methods, totaling 128 requests. Raw, RT62-v1, SplitZip and the raw-prefix hybrid share every reported RULER score; actual token lengths are distinguished from nominal budgets.

D External Comparators and Admission Boundaries

SplitZip uses its official codec with model-specific calibration in the shared resident runtime; the full disaggregated system is outside this evaluation. SplitZip + Raw Prefix is our adapter control using a development-selected raw layer prefix. ExANS uses the separate shared-workspace adapter. SnapKV retains its per-query-head selection and associated K/V head expansion; KIVI uses its supported FP16 Llama path with matched FP16 references.

ExANS probes account for 4,158,509 dynamic native device bytes for one object and 124,829,642 for a batch. Static-device ownership, physical residency and host-pinned memory remain partly unresolved, so these probes do not supply a full-model cache-allocation ratio. The following tables state each comparator's measured dtype and interface.

The ExANS timing batch spans six contents, two models, three lengths, 64/512 outputs and five paired sessions. Its 1,440 timed requests and 400 quality outputs use the eager-scratch configuration described in Section 4.2.

Table 6: Physical layout of a 256-word page. Object prefix and alignment are additional charges.

Component	Compact page (bytes)	Raw page (bytes)
Sign and mantissa bytes	256	–
Packed exponent indices	128	–
Complete exponent palette	16	–
Original BF16 words	–	512
Descriptor	8	8
Page total before object alignment	408	520

Table 7: Natural-stopping request latency. Entries are method/reference percentage changes; brackets give 95% linked paired intervals, and bold estimates have intervals below zero.

Method	Reference	Request $\Delta\%$
KVRecode	Raw	$+15.06$ [+13.37, +16.78]
SplitZip	Raw	$+22.04$ [+18.65, +25.52]
KVRecode	SplitZip	$\mathbf{-5.72}$ [-6.98, -4.43]

Lossy development settings. Each input length uses one exposed Qasper identity and five paired process sessions; quality is one EOS-aware score per identity/configuration. Latency changes use matched Raw SDPA, with 95% process intervals. Storage is final tensor bytes divided by logical Raw KV bytes, excluding separate workspaces; F1 differences are points. Table 9 reports ranges over measured 4K/16K, 64/512 cells and retention settings, rather than uncertainty intervals. FP16-N denotes the native full-logit interface and FP16-L the final-logit-only interface.

SnapKV uses a 32-token observation window, kernel size 7 and max pooling; the retention settings are 0.25, 0.5, 0.75 and 0.9 where admitted. Tables 10 and 11 include every measured native-panel contrast against the matched SDPA reference. KIVI uses the official 2/4-bit paths with its full-precision residual region. Native FP16 16K gates ran out of memory while materializing full FP32 logits. The separate last-logit protocol applies the same final-logit-only conversion to all four FP16 methods, enabling its 4K/16K measurements (Table 12). This protocol change is retained as a separate panel. SnapKV 16K retention 0.75 ran out of memory; 0.9 was not run after the abort. KIVI BF16/Qwen3 paths remain unsupported.

D.1 Online archive processing and component costs

Table 5 compares per-encode archive processing with explicit export using the same codec and storage. On the same Llama development inputs, explicit export decreases TTFT by 64.42–66.75%, while TPOT changes from -0.25% to $+0.25\%$. Resident bytes are identical. Table 13 separates encode, decode, SHA256 and transfers on 24 fixed K/V tensors. The ablation uses eager scratch in both configurations and three paired sessions over 13 combinations of input identity and length; 4K and 16K contents differ. Component timings use five processes with 200 inner calls per operation/tensor. Their ranges span tensor-level process means, not confidence intervals or whole-request sums; ExANS native scratch is outside tensor-owned output storage.

General-purpose lossless codecs. Table 14 gives the earlier D0 layer-level comparison with official nvCOMP LZ4, Cascaded and Bitcomp, separating encode, decode and latency of decoding followed by SDPA. The same post-normalization/RoPE queries and K/V are used within each layer. Cascaded/Bitcomp expand all three encoded buffers, while LZ4 compresses layer 0 and expands layers 15/31. Native scratch is unknown; the table describes layer buffers and hot-working-set timing. This diagnostic uses one 4K Qasper input at Llama layers 0/15/31, with five processes and 50 warmup/200 measured calls. Raw K/V occupies 16,777,216 bytes per layer. D0 and SplitZip use an additional 16,777,216-byte preallocated decode workspace; nvCOMP allocates decode output inside timing. Encoded buffer sizes are distinct from full-cache residency.

Table 8: Quality on the fixed RULER subset. All four methods share each score (0–100); actual token lengths accompany nominal budgets.

Model	Task	Budget	Actual tokens	Shared score
Llama	niah_single_1	8K	8029 to 8033	100.0
Llama	niah_single_1	16K	16235 to 16241	100.0
Llama	vt	8K	8119 to 8126	0.0
Llama	vt	16K	16303 to 16310	0.0
Qwen	niah_single_1	8K	8032 to 8036	100.0
Qwen	niah_single_1	16K	15638 to 16240	100.0
Qwen	vt	8K	8113 to 8119	0.0
Qwen	vt	16K	16080 to 16087	0.0

Table 9: Lossy development comparisons with matched Raw SDPA. Ranges summarize measured conditions, not uncertainty intervals; F1 differences are points.

Model / method	Dtype/path	TPOT $\Delta\%$	Storage ratio	F1 Δ
Llama / KIVI 2-bit	FP16-N	+54.42 to +56.28	0.206 to 0.210	+1.67
Llama / KIVI 4-bit	FP16-N	+54.19 to +56.20	0.328 to 0.332	+0.00
Llama / SnapKV	BF16	+3.90 to +54.57	1.011 to 3.644	-73.50 to +2.56
Qwen / SnapKV	BF16	-3.01 to +40.00	1.011 to 3.644	-9.27 to +0.00
Llama / KIVI 2-bit	FP16-L	-17.47 to +56.43	0.192 to 0.210	+0.00 to +1.67
Llama / KIVI 4-bit	FP16-L	-16.81 to +56.26	0.316 to 0.332	+0.00

E Additional Evaluation Details

Evaluation counts and exclusions. Forty model/method/session processes complete 2,160 measured requests and 240 warmups. Supplements comprise 108 memory profiles, 800 quality outputs and 160 natural-stopping requests; these are not new independent confirmation identities. An additional 200-request stopping run used an incorrect input filter and is excluded from timing analysis. Its records remain in the artifact. Formal timing uses server-exclusive execution after the concurrency pilot failed five of 48 interference checks.

Natural-stopping criterion. The preregistered request-latency noninferiority margin is 1.05 relative to Raw. Both compressed methods exceed it. KVRecode/Raw is 1.15061, with 95% interval [1.13368,1.16785]; KVRecode/SplitZip is 0.94283 [0.93017,0.95572]. These results describe the measured request overhead and the reduction relative to SplitZip.

Unevaluated conditions. The frozen natural contents do not reach 24K/32K, so those resource cells are unmeasured. The 128/512-word page diagnostic supplies space and fallback measurements without corresponding runtime kernels. ExANS full-model memory accounting remains unresolved as described in Appendix D.

F Paired and Identity-Level Timing and Allocation

Table 15 gives all 18 primary percentage changes and their confidence intervals. The millisecond differences in Table 1(b) use the difference of geometric means, each computed by applying the task, identity, model, length and session weights of Equation 12 to log latency; they are not arithmetic means of per-request differences.

Primary-effect visualization. Figure 5 displays the primary estimates and their intervals without introducing another aggregation or comparison population. Figure 7 displays every primary identity at each model, input length, and output count. Each cell summarizes five session-paired request-latency ratios geometrically; it does not pool lengths or models. Identities are ordered by task and then by their frozen hash, independent of the observed latency. This post-confirmation descriptive display introduces no new samples or subgroup inference and does not replace the task-balanced

Table 10: Native lossy comparisons on Llama: BF16 SnapKV and FP16 KIVI. Latency changes are relative to matched Raw SDPA; brackets give TPOT 95% process intervals.

Model	Method	In/out	TPOT Δ%	Request Δ%	Storage	F1 Δ
L	SnapKV 0.25	4K/64	+3.90 [+3.79, +3.99]	+3.29	1.045	+1.67
L	SnapKV 0.25	4K/512	+4.75 [+4.67, +4.84]	+4.61	1.333	+1.67
L	SnapKV 0.25	16K/64	+17.77 [+17.35, +18.08]	+7.62	1.011	-73.50
L	SnapKV 0.25	16K/512	+19.60 [+19.55, +19.64]	+16.50	1.091	-73.50
L	SnapKV 0.5	4K/64	+10.41 [+10.36, +10.47]	+8.20	2.030	+2.56
L	SnapKV 0.5	4K/512	+11.28 [+11.20, +11.38]	+10.88	2.222	+2.56
L	SnapKV 0.5	16K/64	+52.70 [+52.12, +53.09]	+21.02	2.008	+0.00
L	SnapKV 0.5	16K/512	+54.57 [+54.46, +54.65]	+45.55	2.060	+0.00
L	SnapKV 0.75	4K/64	+17.34 [+17.21, +17.50]	+13.46	3.015	+0.00
L	SnapKV 0.75	4K/512	+19.36 [+19.28, +19.46]	+18.63	3.111	+0.00
L	SnapKV 0.9	4K/64	+23.07 [+22.93, +23.17]	+17.76	3.606	+0.00
L	SnapKV 0.9	4K/512	+25.42 [+25.32, +25.55]	+24.44	3.644	+0.00
L	KIVI 4-bit	4K/64	+56.20 [+55.21, +57.50]	+41.54	0.328	+0.00
L	KIVI 4-bit	4K/512	+54.19 [+53.79, +54.49]	+51.97	0.332	+0.00
L	KIVI 2-bit	4K/64	+56.28 [+55.14, +57.42]	+41.59	0.206	+1.67
L	KIVI 2-bit	4K/512	+54.42 [+53.40, +55.72]	+52.19	0.210	+1.67

Table 11: BF16 SnapKV comparisons on Qwen. Changes are relative to matched Raw SDPA; brackets give TPOT 95% process intervals, and bold marks latency decreases (TPOT intervals below zero).

Model	Method	In/out	TPOT Δ%	Request Δ%	Storage	F1 Δ
Q	SnapKV 0.25	4K/64	**−3.01** [-4.04, -2.02]	**−1.85**	1.045	-5.88
Q	SnapKV 0.25	4K/512	**−2.54** [-3.35, -1.73]	**−2.38**	1.333	-5.88
Q	SnapKV 0.25	16K/64	+6.15 [+5.40, +6.77]	+3.49	1.011	-9.27
Q	SnapKV 0.25	16K/512	+7.62 [+6.74, +8.37]	+6.71	1.091	-9.27
Q	SnapKV 0.5	4K/64	**−2.41** [-4.23, -0.24]	**−1.22**	2.030	+0.00
Q	SnapKV 0.5	4K/512	**−2.85** [-4.21, -1.48]	**−2.65**	2.222	+0.00
Q	SnapKV 0.5	16K/64	+38.59 [+37.61, +39.32]	+16.90	2.008	+0.00
Q	SnapKV 0.5	16K/512	+40.00 [+38.87, +40.94]	+34.14	2.060	+0.00
Q	SnapKV 0.75	4K/64	**−1.78** [-2.70, -0.71]	**−0.61**	3.015	+0.00
Q	SnapKV 0.75	4K/512	−0.27 [-0.90, +0.31]	**−0.15**	3.111	+0.00
Q	SnapKV 0.9	4K/64	+2.57 [+2.08, +2.94]	+2.87	3.606	+0.00
Q	SnapKV 0.9	4K/512	+4.59 [+3.80, +5.11]	+4.55	3.644	+0.00

primary estimator. Table 16 gives the resident and lifecycle quantities underlying the allocation audit, keeping each denominator explicit.

Raw-relative cell changes range from +8.89% to +29.12%. SplitZip-relative changes range from −13.49% to +1.83%: all nine Llama 16K/64 cells increase by 1.59–1.83%, while the other SplitZip-relative cells decrease. Thus the aggregate decrease relative to SplitZip does not imply a decrease at every length and output setting. Table 16 reports nine-identity arithmetic resident means and maximum lifecycle bounds for each cell. Workspace includes shared K/V and flat scratch; tails are charged at allocated capacity. Table 17 reuses 45 primary requests per cell at 8K/16K and 64 outputs, reporting arithmetic means for request time and resident bytes and maxima for whole-model allocated/reserved peaks. Memory units are MiB (2^{20} bytes); L/Q denote Llama/Qwen. The 16 planned 24K/32K cells lack eligible natural inputs.

Table 12: FP16 KIVI comparisons on Llama with final-logit-only FP32 conversion. Changes use matched Raw SDPA; brackets give TPOT 95% process intervals, and bold marks latency decreases (TPOT intervals below zero).

Model	Method	In/out	TPOT $\Delta\%$	Request $\Delta\%$	Storage	F1 Δ
L	KIVI 4-bit	4K/64	$+56.26$ [+55.63, +56.89]	$+41.60$	0.328	+0.00
L	KIVI 4-bit	4K/512	$+54.71$ [+54.20, +55.13]	$+52.47$	0.332	+0.00
L	KIVI 4-bit	16K/64	$\mathbf{-16.36}$ [-16.62, -16.07]	$\mathbf{-8.60}$	0.316	+0.00
L	KIVI 4-bit	16K/512	$\mathbf{-16.81}$ [-17.19, -16.27]	$\mathbf{-15.15}$	0.318	+0.00
L	KIVI 2-bit	4K/64	$+56.43$ [+55.67, +57.56]	$+41.73$	0.206	+1.67
L	KIVI 2-bit	4K/512	$+53.94$ [+52.73, +55.08]	$+51.73$	0.210	+1.67
L	KIVI 2-bit	16K/64	$\mathbf{-16.22}$ [-16.87, -15.56]	$\mathbf{-9.10}$	0.192	+0.00
L	KIVI 2-bit	16K/512	$\mathbf{-17.47}$ [-18.04, -17.05]	$\mathbf{-15.88}$	0.194	+0.00

Table 13: Component latency across 24 K/V tensors. Ranges span five-process tensor means in milliseconds.

Operation	Tensors	Latency range (ms)
D0 decode into buffer	24	0.0325 to 0.0583
D0 encode	24	15.4160 to 76.8262
SplitZip adapter decode into buffer	24	0.0659 to 0.0685
SplitZip encode	24	0.1390 to 0.1836
SplitZip official decode + allocation	24	0.0662 to 0.0688
ExANS decode into buffer	24	7.7580 to 29.9340
ExANS encode	24	9.0708 to 36.0166
RT62-v1 archive export	24	454.4092 to 1919.6695
RT62-v1 decode into buffer	24	0.0335 to 0.0631
RT62-v1 encode	24	0.1413 to 0.3351
Body device-to-host transfer	24	1.7664 to 13.1670
Host SHA256	24	4.8664 to 44.3753
Raw clone	24	0.0128 to 0.0253

Table 14: Layer-level lossless codec comparison. Encoded bytes exclude decode buffers and native scratch (size unknown); latency is in milliseconds.

Layer	Method	Encoded bytes	Encode	Decode	Decode+SDPA
0	Raw Flash SDPA	16,777,216	–	–	0.0514
0	KVRecode D0	13,370,704	33.7068	0.0589	0.1180
0	SplitZip	13,102,049	0.2876	0.1327	0.2011
0	LZ4	12,807,058	3.9141	0.7571	0.8726
0	Cascaded	16,783,536	0.9785	0.7021	0.9864
0	Bitcomp	16,806,032	0.6448	0.6769	0.8986
15	Raw Flash SDPA	16,777,216	–	–	0.0512
15	KVRecode D0	13,369,920	31.5052	0.0582	0.1181
15	SplitZip	12,714,697	0.2833	0.1322	0.2006
15	LZ4	16,809,261	4.3936	0.6089	0.7195
15	Cascaded	16,783,536	1.0013	0.7119	0.9393
15	Bitcomp	16,806,032	0.6375	0.6833	0.9140
31	Raw Flash SDPA	16,777,216	–	–	0.0514
31	KVRecode D0	13,370,032	31.8336	0.0601	0.1176
31	SplitZip	12,706,227	0.2977	0.1340	0.2040
31	LZ4	16,807,535	4.3720	0.6269	0.7512
31	Cascaded	16,783,536	0.9948	0.7101	1.0274
31	Bitcomp	16,806,032	0.6374	0.6692	0.9150

Table 15: Uncertainty estimates for Table 1(b). Ours/reference changes and 95% interval bounds are percentages; negative values indicate lower latency.

Reference	Metric	Output	Change (%)	95% interval (%)	
				Lower	Upper
SplitZip	TTFT	64	+0.076	−0.060	+0.191
SplitZip	TPOT	64	−7.660	−8.142	−7.250
SplitZip	Request	64	−5.149	−5.493	−4.861
SplitZip	TTFT	512	+0.074	−0.003	+0.150
SplitZip	TPOT	512	−11.158	−11.625	−10.774
SplitZip	Request	512	−10.447	−10.882	−10.088
SplitZip + Raw Prefix	TTFT	64	+0.161	+0.075	+0.279
SplitZip + Raw Prefix	TPOT	64	−4.409	−4.835	−3.919
SplitZip + Raw Prefix	Request	64	−3.037	−3.335	−2.710
SplitZip + Raw Prefix	TTFT	512	+0.138	+0.084	+0.191
SplitZip + Raw Prefix	TPOT	512	−7.057	−7.454	−6.597
SplitZip + Raw Prefix	Request	512	−6.615	−6.984	−6.189
Raw	TTFT	64	+1.320	+1.169	+1.467
Raw	TPOT	64	+22.233	+21.911	+22.661
Raw	Request	64	+12.722	+12.510	+12.977
Raw	TTFT	512	+1.219	+1.097	+1.330
Raw	TPOT	512	+27.903	+27.501	+28.383
Raw	Request	512	+25.210	+24.853	+25.631

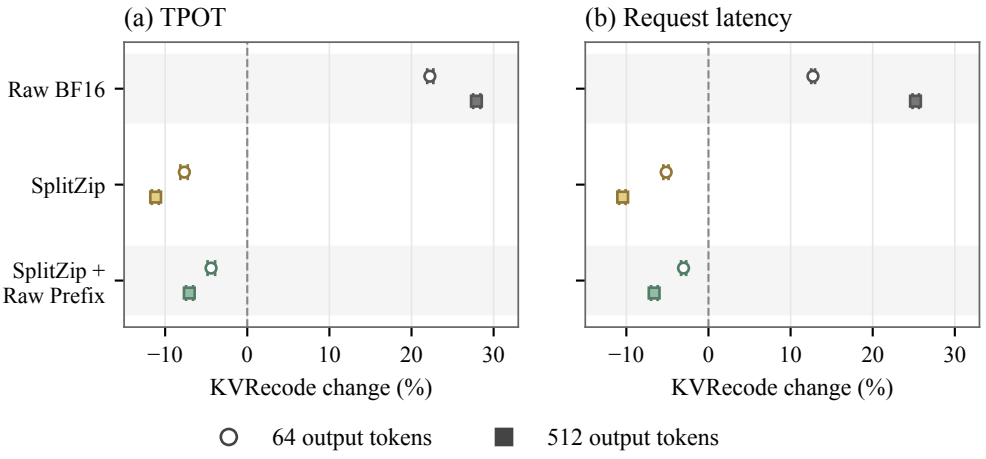


Figure 5: Paired TPOT and request-latency changes for KVRecode relative to Raw, SplitZip, and SplitZip + Raw Prefix. Negative values indicate lower latency. Error bars are the frozen linked identity/session bootstrap 95% intervals. Both output regimes retain the same nine independent contents.

Table 16: Cache allocation across 108 profiles. Resident/capacity reports arithmetic means; lifecycle/capacity reports maximum update bounds, with workspace and tail sizes in MiB.

Input tokens	Output tokens	Resident/ capacity	Lifecycle/ capacity	Workspace (MiB)	Tail (MiB)
Llama-3.1-8B					
4K	64	0.8931	0.8935	24.375	32
4K	512	0.8551	0.8551	27.000	32
8K	64	0.8686	0.8692	48.375	32
8K	512	0.8497	0.8498	51.000	32
16K	64	0.8562	0.8569	96.375	32
16K	512	0.8468	0.8468	99.000	32
Qwen3-8B					
4K	64	0.8881	0.8885	24.375	36
4K	512	0.8501	0.8501	27.000	36
8K	64	0.8636	0.8644	48.375	36
8K	512	0.8448	0.8448	51.000	36
16K	64	0.8514	0.8520	96.375	36
16K	512	0.8420	0.8421	99.000	36

Table 17: Resource use at 8K/16K with 64 output tokens. Request time is seconds; cache means and whole-model allocator peaks are MiB.

Model	Input	Method	Request (s)	Cache	Allocated	Reserved
L	8K	Raw	2.372	1032.0	17288.8	17512.0
L	8K	KVRecode	2.598	896.4	17137.5	17666.0
L	8K	SplitZip	2.779	858.4	17116.1	17470.0
L	8K	SplitZip + Raw Prefix	2.720	885.3	17144.3	17596.0
L	16K	Raw	3.756	2056.0	19244.9	19676.0
L	16K	KVRecode	4.192	1760.4	18917.7	20216.0
L	16K	SplitZip	4.120	1684.6	18875.3	19372.0
L	16K	SplitZip + Raw Prefix	4.071	1741.5	18933.0	19816.0
Q	8K	Raw	2.794	1161.0	17630.9	17854.0
Q	8K	KVRecode	3.185	1002.7	17456.4	17914.0
Q	8K	SplitZip	3.387	962.5	17432.6	17756.0
Q	8K	SplitZip + Raw Prefix	3.305	1000.5	17471.0	17746.0
Q	16K	Raw	4.214	2313.0	19619.0	20050.0
Q	16K	KVRecode	4.606	1969.3	19242.3	20398.0
Q	16K	SplitZip	4.812	1888.8	19195.3	19690.0
Q	16K	SplitZip + Raw Prefix	4.731	1969.2	19275.9	20188.0

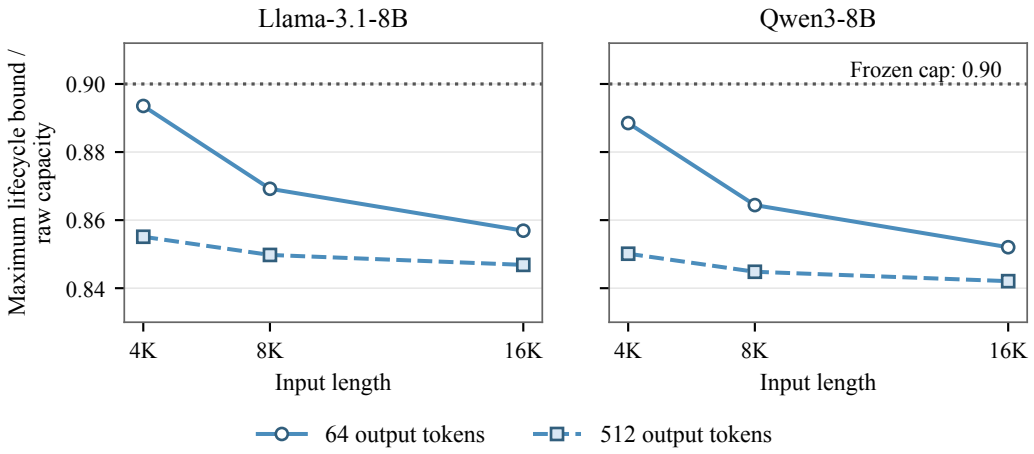


Figure 6: Maximum cache-attributed lifecycle bound relative to Raw capacity across nine profiles per model/length/output cell. The horizontal line is 0.90. The bound includes cache-update allocation and excludes common model activations; whole-model allocator peaks are recorded separately.

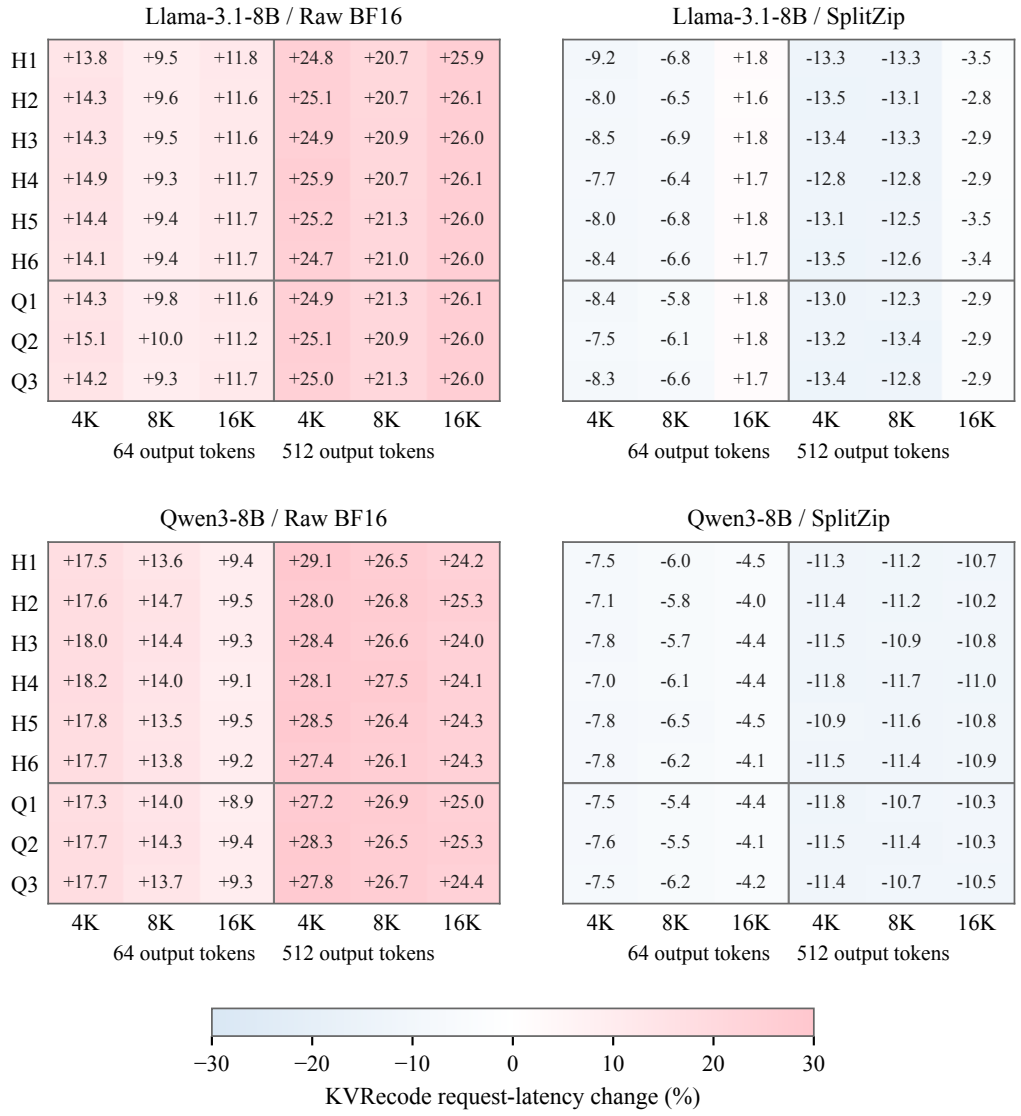


Figure 7: KVRecode request-latency changes (percent) relative to Raw and SplitZip for all nine primary contents: H1–H6 are HotpotQA and Q1–Q3 are QMSum. Each cell is $100(\mathrm{GM}_{\mathrm{session}}(T_{\mathrm{KVRecode}}/T_{\mathrm{reference}}) - 1)$ over five matched sessions. Negative values indicate a decrease and positive values an increase. All panels use the same zero-centered color scale; signed values remain explicit. These are descriptive cell summaries without confidence intervals, not independent confirmation results.

G Extended Related Work

Serving and cache management. PagedAttention manages allocation and sharing through logical cache blocks (Kwon et al., 2023), while SGLang reuses cached prefixes in structured language-model programs (Zheng et al., 2024). These mechanisms concern where reusable state resides and how it is shared; KVRecode changes the bit representation of that state. Orca schedules at the iteration level and selectively batches model operations (Yu et al., 2022). Sarathi-Serve combines chunked prefill with stall-free batching (Agrawal et al., 2024). DistServe and Splitwise separate prefill and decoding to accommodate their different resource demands (Zhong et al., 2024; Patel et al., 2024). Our batch-one resident-cache measurements isolate codec and reconstruction costs rather than evaluating these multi-request scheduling policies.

Offloading and transfer compression. FlexGen schedules tensors across GPU, CPU and disk for throughput-oriented inference under limited GPU memory (Sheng et al., 2023). InfiniGen speculates which cache entries will matter and prefetches selected entries from host memory (Lee et al., 2024). CacheGen combines quantization with change-based encoding and arithmetic coding to reduce cache-transfer cost (Liu et al., 2024b). Its quantization changes numerical values, whereas our word-level contract preserves them. Transfer compression, selective fetching and repeated resident-cache reconstruction address different costs; our request measurements charge online encoding and reconstruction on the inference device.

Selection, sparsity and alternative representations. Ada-KV assigns eviction budgets adaptively across attention heads (Feng et al., 2025); KVzip scores cache entries by reconstructing their source context and evicts lower-importance entries (Kim et al., 2025). StreamingLLM retains attention sinks alongside a recent window (Xiao et al., 2024b). Quest selects cache pages according to the current query (Tang et al., 2024), while DuoAttention separates full-context retrieval heads from bounded-cache streaming heads (Xiao et al., 2025). InfLLM retrieves relevant units from an external context memory (Xiao et al., 2024a). These policies reduce retained or attended context. LESS augments sparse caches with a fixed-size recurrent state (Dong et al., 2024); MiniCache merges similar states across depth (Liu et al., 2024a); Layer-Condensed KV Cache computes and stores K/V at a subset of layers (Wu & Tu, 2024). They alter attention state or its use, rather than reversibly encoding every original BF16 word.

Attention kernels and evaluation coverage. FlashAttention-3 targets Hopper GPUs using asynchronous execution and includes low-precision attention variants (Shah et al., 2024); it is not the backend of our RTX 4090D measurements. On evaluation, InfiniteBench extends tasks beyond 100K tokens (Zhang et al., 2024), LongBench v2 emphasizes understanding and reasoning over long-context multitasks (Bai et al., 2025), and SCBench examines cache generation, compression, retrieval and loading across shared-context workloads (Li et al., 2025). They define complementary evaluation settings. Our measured task interfaces remain LongBench and the reported RULER subset; we do not report results on these three additional benchmarks.

Recent compression and serving directions. TurboQuant uses random rotation and scalar quantization, with a residual correction for inner-product estimation (Zandieh et al., 2026). PM-KVQ progressively lowers cache precision and assigns bit-widths by block sensitivity (Liu et al., 2026a); ThinKV combines thought-dependent precision with token eviction (Ramachandran et al., 2026). FreqKV removes high-frequency components of K/V representations for context extension (Kai et al., 2026). These methods change cached values or retained information, whereas KVRecode reconstructs the original BF16 words. ECF8 Huffman-codes weight exponents and reconstructs FP8 weights into a shared layer-sized buffer (Yang et al., 2026). KVRecode instead uses complete fixed-width local palettes for evolving BF16 K/V activations, with online appends and repeated reconstruction before attention.

At the serving level, Strata couples hierarchical cache layouts with cache-aware scheduling (Xie et al., 2026), while Bidaw coordinates request scheduling and two-tier KV storage (Hu et al., 2026). CacheSlide reuses position-shifted cache segments through positional encoding and weighted attention correction (Liu et al., 2026b). They address movement, scheduling or reuse of cache state, rather than the page-wise reversible representation evaluated here.