FROM MEMORY SAVINGS TO THROUGHPUT GAINS: SERVING-NATIVE KV COMPRESSION FOR SPARSE-ATTENTION LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context LLM serving is increasingly bottlenecked by the growing memory and bandwidth demands of the KV cache. Sparse attention alleviates the KV access cost, but still leaves the full KV cache resident, limiting serving concurrency. Existing KV compression methods primarily target compact storage or optimized kernels, rather than the execution and lifecycle of compressed KV in online serving, where compression timing, efficient attention over compressed states, and cache management directly affect performance. We present Duet, a serving-native KV compression framework that turns KV memory savings into practical serving gains while preserving the underlying sparse-attention selection semantics. Duet first develops an inference-friendly low-rank representation that exploits cross-head redundancy and removes per-token reconstruction from sparse attention. Building on this, a mixed native-latent attention operator directly consumes selected KV blocks in either form without materializing compressed history. Finally, Duet integrates compression into the serving runtime through dual-representation KV management, watermark-triggered batched compression, and phase-specialized execution. Together, these designs reduce KV memory usage by $4\times$ while preserving efficient attention execution, delivering up to $1.8\times$ higher throughput and a $16.8\times$ reduction in TTFT, with less than 1% average accuracy loss.

1 INTRODUCTION

Long-context LLM serving is increasingly constrained by the growing memory and bandwidth demands of the KV cache (Kwon et al., 2023). Dynamic sparse attention (DSA) alleviates the access cost by processing only a query-dependent subset of the history (Tang et al., 2024; Jiang et al., 2024; Yuan et al., 2025). However, the remaining KV states must stay available because future queries may select them, so sparsity reduces the *active* working set without reducing the full-history capacity requirement. Recent systems address this gap through hierarchical KV management, keeping most states in host memory and fetching selected entries on demand (Lee et al., 2024; Xie et al., 2026). This bounds GPU residency, but shifts pressure from HBM capacity to host–device bandwidth. Compression attacks a complementary dimension: rather than relocating the full representation, it reduces the representation itself.

KV compression provides a complementary way to reduce the resident footprint. Prior work has explored low-bit quantization (Liu et al., 2024d; Hooper et al., 2024), low-rank representations (Saxena et al., 2024; Chang et al., 2025), and optimized lossy encodings (Zandieh et al., 2026; Krishnan & Schulz, 2026; Patel, 2026). Several systems further optimize compressed KV for inference through specialized kernels (Cheng et al., 2025; Jiang et al., 2025). Dynamic sparse attention, however, changes the cost balance: once sparsity has already removed most KV accesses, further reducing the bytes per KV state does not necessarily yield proportional operator speedups. Compression for DSA must therefore optimize not only footprint and fidelity, but also *execution under selective access*. This requirement extends to the attention operator itself: selected blocks may reside in different representations in online serving scenarios, so native and compressed KV must be consumed together without sacrificing the efficiency. Existing methods combining KV reduction with sparse access (Behnam et al., 2025) typically couple compression with their own selection or reconstruction mechanisms; our setting instead preserves the semantics of an existing sparse-attention selector.

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.

Beyond efficient compressed inference, realizing the memory benefits of KV compression in serving requires system-level integration. Existing system-oriented approaches mainly use compression to reduce the cost of reusable KV storage (Liu et al., 2024c; Staniszewski & Lancucki, 2026), remote cache fetching (Xiang et al., 2025; Mi et al., 2026), or KV transfer in disaggregated and prefill–decode separated serving (Guo & Joshi, 2026; Liu et al., 2026). In these settings, compression primarily reduces the amount of data stored or transferred across system boundaries. Online serving presents a different lifecycle challenge: KV states are generated incrementally, multiple representations coexist, conversion consumes GPU resources, and memory savings improve concurrency only after uncompressed residency is reclaimed. Serving-aware memory-management schemes such as DiffKV (Zhang et al., 2025b) address parts of this challenge, but do not jointly support low-rank compressed execution under an external dynamic sparse selector. Turning KV compression into serving gains therefore requires treating compressed KV as a *first-class execution-time representation*—one that can be created online, directly consumed by attention, managed alongside native KV, and reclaimed or rematerialized as requests evolve.

In this paper, we present Duet, a serving-native low-rank KV compression framework for dynamic sparse attention that preserves its selection semantics. Duet translates memory savings into serving gains through three complementary designs: (1) an inference-friendly joint-head low-rank representation that exploits cross-head redundancy without per-token reconstruction on the sparse-attention path; (2) a mixed native-latent attention operator that directly processes selected KV blocks in either representation, enabling compressed history to coexist efficiently with writable and protected native KV; and (3) a serving runtime that manages compressed KV as a first-class execution-time representation via dual-representation management, batched online compression, and phase-specialized execution. Together, these designs turn KV memory savings into serving capacity and higher throughput. Across three model families, Duet reduces KV memory usage by $4\times$, improves serving throughput by up to $1.8\times$, and reduces TTFT by up to $16.8\times$, with less than 1% average accuracy loss.

2 Background and Motivation

Selector-preserving joint-head compression. We target KV compression for block-wise dynamic sparse attention (DSA), which partitions the KV cache into fixed-size blocks, the same granularity as paged KV memory (Kwon et al., 2023), and keeps a compact key-derived *summary* per block for query-dependent selection (Jiang et al., 2024; Yuan et al., 2025). We primarily use Quest (Tang et al., 2024), which summarizes block b of each KV head by the element-wise minimum and maximum of its keys, $k_b^{\min}$ and $k_b^{\max}$, and scores the block for a query q as

Table 1: Relative Frobenius error of per-head and joint-head SVD compression on Qwen3-4B. Columns give the compression ratio.

Method	$1/8$	$1/4$	$1/2$
Key, per-head	0.4861	0.3835	0.2658
Key, joint-head	**0.3330**	**0.2692**	**0.1847**
Value, per-head	0.7200	0.6009	0.4266
Value, joint-head	**0.4994**	**0.3717**	**0.2259**

$$s_b(q) = \sum_i \max\left(q_i k_{b,i}^{\min},\, q_i k_{b,i}^{\max}\right). \tag{1}$$

The top-scoring blocks within a budget are then attended; selection is made per KV head and shared by the query heads of its GQA group. Our design does not depend on this scoring rule and applies to any block-wise DSA whose selection metadata can be maintained independently of the KV payload.

DSA reduces the KV read by each query but still requires the full KV to stay resident. Motivated by prior low-rank methods (Staniszewski & Lancucki, 2026; Mu et al., 2025), we study SVD-based compression of this payload. As shown in Table 1, sharing one basis across all KV heads of a layer (joint-head) achieves lower reconstruction error relative to a separate basis per head, revealing substantial cross-head redundancy. A joint-head latent, however, entangles all KV heads of a token, whereas DSA scores each KV head on its own summary. Per-head keys, and hence summaries, can then be obtained only by reconstruction, which feeds compression error directly into the selector: in our experiments, summaries computed from reconstructed keys change up to 60% of the selected blocks and reduce the attention mass they capture by up to 39% relative to summaries from the original keys (see Appendix A for more details). We therefore *decouple sparse selection from KV compression*: block summaries are always computed from the original keys, and only the KV payload read after selection is compressed, thus preserving the sparse selector semantics.

Inference-friendly placement of key compression. A common choice in low-rank KV compression is to compress keys before RoPE (Su et al., 2024), since the position-dependent rotations can weaken the shared low-rank structure (Sun et al., 2025; Chang et al., 2025). For inference, however, reconstruction quality is only part of the design objective: the placement of compression also determines whether the low-rank projection can be efficiently absorbed into attention.

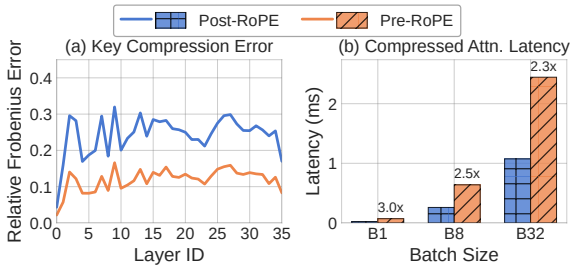

Figure 1: Pre-RoPE keys yield lower compression error, while post-RoPE keys enables lower attention latency, conducted on Qwen3-4B.

This distinction is particularly important for joint-head compression. With pre-RoPE keys, the position-dependent RoPE transformation prevents the SVD basis projection from being absorbed into a single query-side projection shared across keys. With post-RoPE key compression, let $z_{K,t} \in \mathbb{R}^{r_K}$ denote the compressed key at position t, and let $B_{K,h} \in \mathbb{R}^{r_K \times d}$ denote the key compression basis for KV head h. The attention score can be written as

$$q^\top B_{K,h}^\top z_{K,t} = (B_{K,h} q)^\top z_{K,t}. \tag{2}$$

Since $B_{K,h} q$ is independent of the key position t, it can be computed once and reused across all selected keys, enabling efficient direct attention over latent keys. We defer a detailed mathematical derivation of pre- and post-RoPE compression to Appendix B. The resulting trade-off is shown in Figure 1. Besides, because compression is decoupled from sparse selection, the larger post-RoPE reconstruction error does not propagate into the DSA block-selection decisions. We therefore choose post-RoPE keys to optimize the representation for inference efficiency rather than reconstruction fidelity alone.

Compression is not free after sparsification. Online sparse-attention serving naturally produces a mixture of native and compressed KV cache. Some blocks must remain uncompressed to support sink or local-window policies to maintain accuracy (Xiao et al., 2024b), while recently generated blocks may not yet have been compressed. Consequently, each selected token under sparse attention must be routed to the appropriate representation, and the partial attention results from the two sources must be merged. The left panel of Figure 2 shows that this routing and output merging introduce non-negligible overhead, even when compressed KV is consumed directly without materialization.

More importantly, sparse attention already removes most full-history KV traffic, leaving limited additional bandwidth savings from compression. After accounting for its extra overheads, sparse attention with compression is slightly slower than raw sparse attention, as shown in Figure 2 (b). Under DSA, KV compression should therefore be viewed primarily as a *capacity optimization*, rather than an attention-kernel speedup. Realizing end-to-end gains requires converting the reduced KV footprint into higher concurrency while minimizing the overhead of compressed execution.

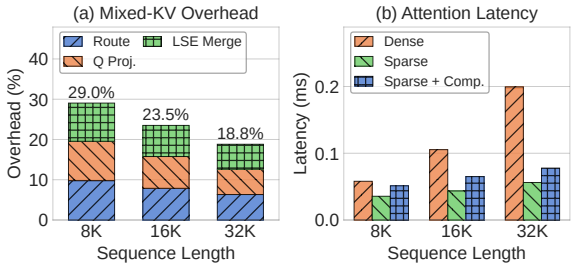

Figure 2: Left: additional overhead from mixed native/compressed representations. Right: adding compression increases sparse-attention latency.

Online serving introduces new requirements for compression. Transforming KV memory savings into serving gains requires more than optimizing a single attention kernel. As shown in Figure 3(a), the preferred execution path depends on the inference phase. Specifically, during extend prefill, many queries re-select the same prefix. As the extend chunk grows, materializing the compressed prefix once and reusing the native prefill path becomes increasingly favorable (see Appendix C for further analysis). Compression therefore requires phase-specialized execution rather than one uniform path. Additionally, compression timing introduces a second trade-off. Compressing every completed block immediately incurs frequent fine-grained GPU work, whereas delaying compression leaves more completed blocks resident in

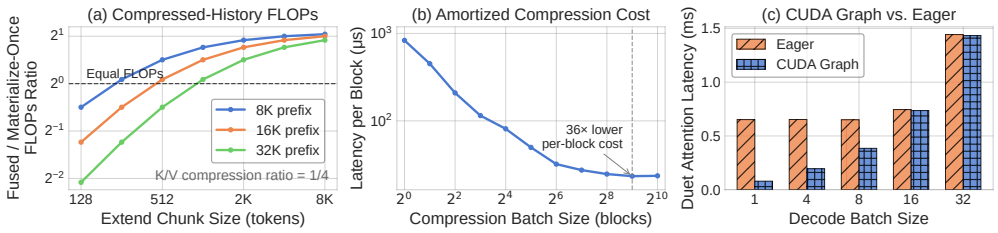

Figure 3: Serving-specific trade-offs of KV compression. (a) Compressed-history FLOPs of direct latent versus materialize-once extend prefill. (b) Amortized per-block compression latency versus batch size. (c) Decode attention latency with eager execution and CUDA Graph replay.

uncompressed form and weakens the memory benefit. Figure 3(b) shows that batching conversion reduces the amortized per-block compression cost by up to $36\times$, motivating batched online compression that balances conversion overhead against native KV pressure.

Compression must also coexist with the mechanisms used by modern serving systems. Representation changes should not disrupt paged KV allocation, prefix caching, or other stateful runtime abstractions that rely on stable logical block identities and ownership Kwon et al. (2023). Execution optimizations impose additional constraints: CUDA Graph replay, for example, requires stable memory addresses and execution structure even though KV residency may change across iterations. Figure 3(c) shows that preserving graph replay remains important for decode performance. A serving-oriented compression runtime must therefore make representation changes transparent to these existing cache-management and execution mechanisms. Together, these observations show that KV compression in online serving is a runtime problem as much as a representation problem. An effective design must support phase-specialized execution, amortize online conversion without accumulating excessive uncompressed KV, and integrate with existing serving abstractions.

3 Method

Duet keeps each KV block in two representations: the *native* uncompressed KV produced by the model and a joint-head low-rank compressed *latent*. The latent is persistent and defines KV capacity, whereas native residency is transient (Figure 4).

3.1 Inference-Friendly Joint-Head Latent KV

Calibration and encoding. For each layer, we concatenate the post-RoPE keys of all H_{kv} KV heads at position t into $k_t \in \mathbb{R}^D$, $D = H_{\text{kv}} d$, and the values into $v_t \in \mathbb{R}^D$. On a calibration corpus, truncated SVD of the stacked key and value activations yields bases $B_K \in \mathbb{R}^{r_K \times D}$ and $B_V \in \mathbb{R}^{r_V \times D}$, formed by the leading right-singular vectors and computed once offline. Each token is stored as $z_{K,t} = B_K k_t$ and $z_{V,t} = B_V v_t$; *e.g.*, $r_K = r_V = D/4$ makes the persistent KV payload $4\times$ smaller than in Quest. Partitioning $B_K = [B_{K,1}, \ldots, B_{K,H_{\text{kv}}}]$ with $B_{K,h} \in \mathbb{R}^{r_K \times d}$, and B_V likewise, the latents implicitly represent KV head h as $\hat{k}_{t,h} = B_{K,h}^\top z_{K,t}$ and $\hat{v}_{t,h} = B_{V,h}^\top z_{V,t}$.

Selection metadata. DSA selects blocks using per-head summaries of their keys, such as the min/max vectors of Quest in Equation 1. Duet computes these summaries from the original post-RoPE keys when a block fills, rather than from reconstructed latents. The summaries are kept as separate metadata that outlives the native copy, so the selector never reads latents: it scores blocks per KV head, shared within each GQA group, on exactly the summaries of uncompressed DSA.

Latent attention. Let query head j map to KV head h, with post-RoPE query q_j. Extending Equation 2 to the value path gives, for any attention weights $\alpha_{j,t}$,

$$q_j^\top \hat{k}_{t,h} = \tilde{q}_j^\top z_{K,t}, \qquad \textstyle\sum_t \alpha_{j,t}\, \hat{v}_{t,h} = B_{V,h}^\top \sum_t \alpha_{j,t}\, z_{V,t}, \qquad \tilde{q}_j = B_{K,h}\, q_j. \tag{3}$$

Each query is thus projected once per step, scoring and value aggregation operate directly on latents, and $B_{V,h}^\top$ is applied once to the aggregated result; no key or value is reconstructed per token. This mirrors absorbed inference in multi-head latent attention (MLA) (Liu et al., 2024a; Meng et al., 2025), except that the latent is derived post hoc from a GQA model and, as keys are compressed after RoPE, $B_{K,h}$ acts on the rotated query at runtime instead of being folded into the query projection.

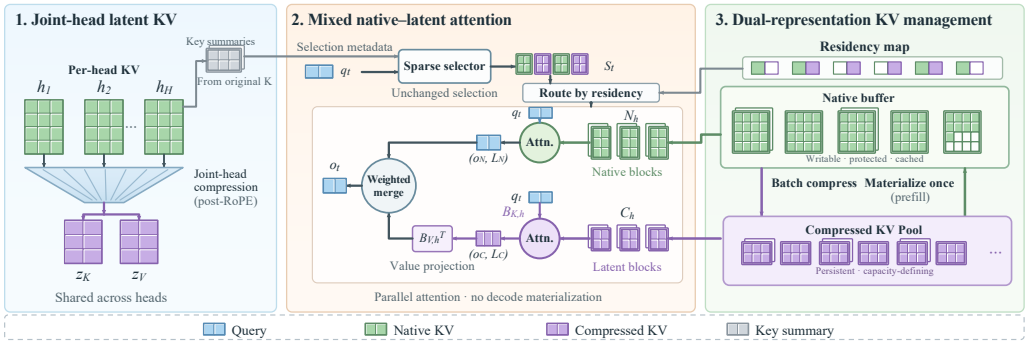

Figure 4: Dual-representation KV management in Duet. Compressed KV provides persistent storage, and a bounded native pool holds writable, protected, and pending blocks. Decode consumes mixed history directly, whereas extend prefill materializes compressed prefixes once.

3.2 Mixed Native–Latent Sparse Attention

During decode, the selected blocks of a request may be native, compressed-only, or both, and Duet attends to them where they reside rather than converting them into a homogeneous KV workspace. The operator has two parts: a routing–attention–merge pipeline that serves each representation with its own kernel and recombines the results exactly, and a residency-aware regrouping that keeps the mixture from diluting either kernel's parallel work.

Routing, parallel attention, and merging. Duet first resolves each selected block through the residency map (Section 3.3) and splits the selected tokens of KV head h into $\mathcal{N}_h$ from native slots, and $\mathcal{C}_h$ as latents. A dual-resident block is routed to $\mathcal{N}_h$, so every selected token is consumed exactly once. The two sets are then attended in parallel on separate CUDA streams: the existing sparse-attention kernel processes $\mathcal{N}_h$ unchanged, while a latent kernel evaluates Equation 3 over $\mathcal{C}_h$ without materializing native KV, sharing each latent block and basis slice across the query heads of a GQA group. For each query head, they return partial outputs and log-sum-exps (o_N, L_N) and (o_C, L_C), where $o_C \in \mathbb{R}^{r_V}$ is still latent, and a fused epilogue applies the value basis and the merge:

$$o = \frac{e^{L_N - M}\, o_N + e^{L_C - M}\, B_{V,h}^{\top} o_C}{e^{L_N - M} + e^{L_C - M}}, \qquad M = \max(L_N, L_C). \tag{4}$$

Since $\mathcal{N}_h \cap \mathcal{C}_h = \emptyset$, Equation 4 equals one softmax attention over $\mathcal{N}_h \cup \mathcal{C}_h$ in which compressed tokens take their reconstructions $(\hat{k}, \hat{v})$, adding no error beyond compression itself.

Residency-aware regrouping. Routing alone leaves the parallel work poorly shaped. Both kernels are launched over the same split-K chunks (Hong et al., 2024), but the selector emits blocks in its own order, interleaving native and compressed ones, so nearly every chunk holds both representations and each kernel visits almost every chunk only to skip much of it. This interleaving is an artifact of selection order rather than a property of attention, which is invariant to the order of keys. Duet therefore stably regroups the complete selected blocks by residency in a per-step index buffer, keeping the partially filled tail block, which is always native, at the end. Mixing is then confined to the few chunks straddling the representation boundary. Each chunk is tagged native-only, latent-only, or mixed, and a kernel exits immediately on chunks without its own work. The regrouping changes only execution order: it touches neither the selector output nor the block tables.

3.3 Dual-Representation KV Management

Duet is built on vLLM (Kwon et al., 2023) and manages KV as one logical cache with two physical representations. The compressed slot is the canonical block identity: each logical block reserves one at creation, and its ID is the only identity seen by block tables, prefix-cache metadata, reference counts, and the scheduler, so capacity is counted in compressed slots. Native KV acts as a cache over these blocks: a global residency map attaches an optional native slot to each ID, placing every block, including prefix-shared ones, in one of three states (native-only, dual-resident, or compressed-only). The rest of the runtime follows from this abstraction: how blocks move between states, which states each serving phase requires, and why these moves are invisible to graph replay.

Table 2: LongBench results across three model families. Higher is better.

(a) Summarization, few-shot, and synthetic tasks

Model	Method	Summarization				Few-shot learning				Synthetic	
		Gov.	QMSum	MNews	VCSum	TREC	TQA	SAMSum	LSHT	Count	Retr.
Qwen3-4B	Dense	**32.40**	**24.75**	25.17	14.99	56.75	**89.37**	40.78	41.00	**1.50**	96.50
	Quest	32.37	24.44	25.35	14.76	56.25	89.31	**40.92**	**41.25**	**1.50**	**97.50**
	Duet	30.18	24.28	**25.62**	**15.15**	**58.00**	**89.37**	40.79	41.00	1.25	96.50
Llama-3.1-8B	Dense	**34.40**	**25.51**	26.84	**17.56**	15.50	91.50	13.07	**0.00**	10.00	**99.50**
	Quest	34.31	**25.51**	**26.85**	17.40	16.00	**91.58**	12.92	**0.00**	**11.00**	**99.50**
	Duet	27.78	24.52	24.22	15.47	**20.00**	91.42	**13.37**	**0.00**	9.18	**99.50**
Mistral-Nemo-12B	Dense	30.90	24.20	**26.27**	**17.30**	**12.00**	89.39	42.22	27.50	0.50	97.00
	Quest	**31.28**	**24.22**	26.11	17.19	11.50	89.19	**42.26**	27.50	0.50	97.00
	Duet	27.58	21.78	23.63	15.01	11.00	**89.54**	40.83	**28.00**	**1.00**	**99.00**

(b) Question answering tasks

Model	Method	Single-document QA			Multi-document QA				Avg.
		Qasp.	MFQA	NQA	HQA	2Wiki	MuSiQue	DuR.	
Qwen3-4B	Dense	43.46	54.44	27.70	55.18	43.45	32.28	30.13	41.76
	Quest	**43.60**	**54.91**	**27.74**	55.11	43.58	31.84	**30.16**	**41.80**
	Duet	42.51	46.64	24.70	**55.25**	**43.70**	**32.52**	29.74	41.01
Llama-3.1-8B	Dense	44.69	**56.08**	27.97	57.61	**49.42**	**33.27**	34.09	37.47
	Quest	**45.07**	55.72	28.03	**57.82**	49.08	32.09	**34.35**	**37.48**
	Duet	42.25	55.32	**29.34**	57.80	47.12	31.87	30.43	36.45
Mistral-Nemo-12B	Dense	**42.70**	**57.97**	**26.77**	50.26	45.91	27.05	32.34	38.25
	Quest	42.23	57.46	26.65	**50.28**	46.47	**28.10**	**32.41**	**38.26**
	Duet	42.23	57.79	24.52	50.06	**46.55**	27.18	28.53	37.31

Abbreviations: Gov.=GovReport, MNews=MultiNews, TQA=TriviaQA, Count=PassageCount, and Retr.=PassageRetrieval-en, Qasp.=Qasper, MFQA=MultiFieldQA, NQA=NarrativeQA, HQA=HotpotQA, DuR.=DuReader.

Block lifecycle. New tokens are written natively, and a block's summary is finalized when it fills. A block becomes pending once it leaves the local window; when W blocks are pending, Duet compresses them for all layers in one batched conversion between model steps, making them dual-resident. Attention sinks and local windows are protected and never compressed. Native copies are reclaimed lazily, only when the pool needs space and the block is unpinned, leaving it compressed-only; the reverse transition, materialization, serves extend prefill. The native pool therefore reserves $L_{\max} + S_{\max}(n_{\text{sink}} + n_{\text{win}}) + W$ blocks: one maximum-length materialized prefix, $n_{\text{sink}} + n_{\text{win}}$ protected blocks for each of $S_{\max}$ running sequences, and at most W pending blocks.

Phase-specialized execution. Decode reads selected blocks in their current form through the mixed operator and never forces a state change. Extend prefill, which appends a new chunk to a cached prefix, requires native KV: it materializes the compressed-only blocks of that prefix once into pinned native slots, runs the native sparse prefill kernel over the whole chunk, and then unpins them. Admitting at most one extend prefill per step keeps materialization within the $L_{\max}$ reservation.

4 Experiments

4.1 Experimental Setup

Models and hardware. We evaluate Duet on three instruction-tuned GQA models from 4B to 12B parameters: Qwen3-4B-Instruct-2507 (Yang et al., 2025), Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and Mistral-Nemo-Instruct-2407 (Mistral AI Team, 2024), referred to by their base names below. Efficiency experiments run on an NVIDIA RTX A6000 (48 GB) for Qwen3-4B and an NVIDIA RTX PRO 6000D (84 GB) for the two larger models.

Baselines and configurations. We implement Duet on vLLM (Kwon et al., 2023) and compare it with two baselines in the same codebase: vLLM with dense attention, and Quest (Tang et al., 2024) as the uncompressed DSA reference. Duet uses Quest as its selector with identical settings, so the gap between the two isolates the effect of compression. Quest selects up to 2048 tokens per query in blocks of 16 tokens, and both sparse methods always attend two sink and two local-window blocks ($n_{\text{sink}} = n_{\text{win}} = 2$). Unless stated otherwise, Duet uses $r_K = r_V = D/4$, a $4\times$ reduction of the

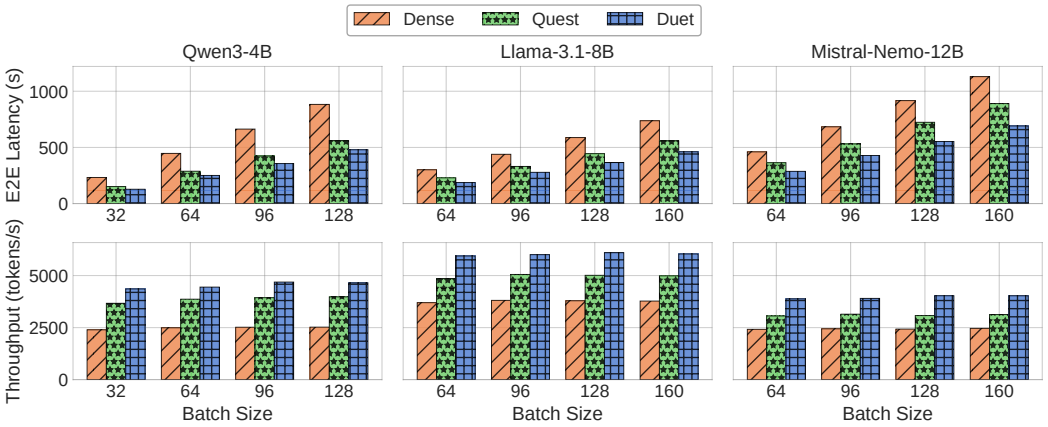

Figure 5: Offline end-to-end latency (top) and token throughput (bottom) under increasing batch sizes.

persistent KV payload since latents stay in BF16 without quantization, and a compression watermark of $W = 512$ blocks. Detailed configurations can be found in Appendix D.

Benchmarks and workloads. We measure accuracy on LongBench (Bai et al., 2024) and RULER (Hsieh et al., 2024). Efficiency is measured in three settings: offline batch inference, which submits a batch of requests at once; single-turn online serving; and multi-turn online serving, where conversations issue each turn after the previous response, so later turns can reuse earlier context through prefix caching. We report time to first token (TTFT), inter-token latency (ITL), and token throughput. See Appendix D for more details.

4.2 Main Results

Accuracy. Quest matches dense attention on LongBench (Table 2), and Duet stays within about one point of Quest on the average for all three models. Retrieval-heavy tasks such as PassageRetrieval are nearly unaffected, consistent with the unchanged selector: Duet locates the same blocks as Quest, and compression only approximates their contents. The drops concentrate on summarization tasks such as GovReport, which aggregate information from many selected blocks and are therefore more exposed to compression error. Results on RULER are reported in Appendix F.

Offline throughput. Quest outperforms dense attention by reading less KV at each decoding step (Figure 5). Duet improves throughput by a further 1.15–$1.3\times$ over Quest, and 1.6–$1.9\times$ over dense attention, across all models and batch sizes, since its smaller KV footprint admits more concurrent sequences. End-to-end latency drops accordingly.

Online serving. In single-turn serving (Figure 6), Duet sustains up to $1.27\times$ the token throughput of Quest and a much lower TTFT, as requests wait less for KV capacity. Its P95 ITL is about 7% higher, because each decoding step serves a larger batch and attention additionally processes latents. Multi-turn serving amplifies the gap (Figure 7): Duet reaches up to $1.9\times$ the throughput of Quest and keeps TTFT low at all request rates, whereas the throughput of Quest degrades as load grows. Table 5 attributes this gap to prefix reuse. With $4\times$ the KV capacity, Duet keeps earlier turns cached in compressed form, so a new turn extends a cached prefix instead of recomputing it, raising the prefix-cache hit rate from 6.9% to 63.8%. Dense attention and Quest instead evict these prefixes under memory pressure, recompute them, and queue about $10\times$ more requests. As a result, Duet runs a $1.7\times$ larger average batch while occupying less than half of its KV capacity, and its P95 ITL is also lower, since fewer recomputation prefills interleave with decoding. Results on other models can be found in Appendix E.

4.3 Extended Studies

Ablation of Mixed attention operator. We compare the mixed operator with materialize-then-attend, which gathers the selected native blocks and reconstructs the selected compressed blocks into a contiguous native workspace before running the sparse-attention kernel. Figure 8 breaks down

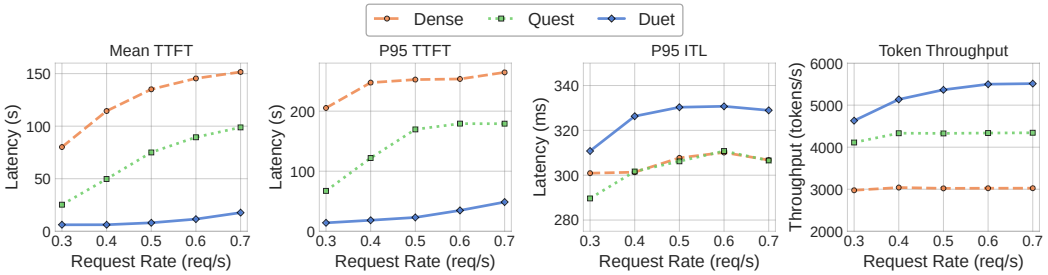

Figure 6: Single-turn online serving of Llama-3.1-8B on an RTX PRO 6000D.

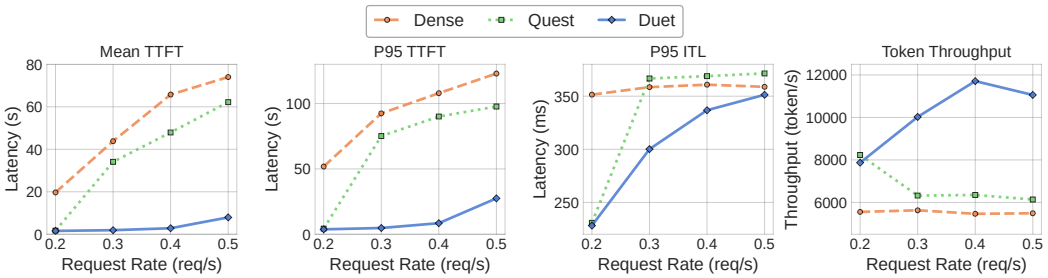

Figure 7: Multi-turn online serving of Qwen3-4B on an RTX A6000.

single-layer decode latency using Qwen3-4B across native KV ratios, *i.e.*, the fraction of selected KV that is native. Materialization is dominated by decompression when most selected KV is compressed and by native copying when most is native; direct mixed attention avoids both at every ratio. Table 3 separates the two parts of the operator: routing each representation to its own kernel is 1.47–$1.60\times$ faster than materialize-then-attend, and residency-aware regrouping raises the speedup to 1.65–$1.69\times$. Regrouping helps most at batch size 1, where split-K chunks are the main source of parallelism and chunks mixing both representations waste a larger share of it.

Compatibility with quantization. Low-rank compression reduces the dimension of each KV state, whereas quantization reduces the bits per element, so the two compose. Figure 9 additionally quantizes the key and value latents of Duet to 4 and 8 bits (K4V8) on the multi-turn workload of Qwen3-4B. The smaller footprint further raises throughput at higher request rates and lowers TTFT at the highest rate, where KV capacity again becomes the bottleneck. Additional results on accuracy under quantization and different compression ratios are provided in Appendix G.

Table 5: Serving statistics of Llama-3.1-8B on the multi-turn workload at 0.4 req/s.

Metric	Dense	Quest	Duet
Avg. Batch Size	20.2	20.4	34.7
Max Batch Size	23	24	70
Avg. Waiting Reqs.	34.7	33.3	3.5
KV Usage (%)	87.4	87.9	41.2
Prefix Hit (%)	6.5	6.9	63.8

Calibration cost. Duet requires no training or fine-tuning. Its only preparation is a one-time offline calibration, which takes at most about 100 seconds per model on a single GPU (Table 4).

5 Related Work

KV quantization and compression. KV compression shrinks the cache along three axes: the bits per element, the elements per token, and the tokens retained. Quantization lowers element precision (Liu et al., 2024d; Hooper et al., 2024; Kang et al., 2024; Zandieh et al., 2026; Patel, 2026), and QServe (Lin et al., 2025) co-designs 4-bit KV with serving kernels. Low-rank projection, tensor decomposition, and transform coding reduce the elements per token (Saxena et al., 2024; Chang et al., 2025; Krishnan & Schulz, 2026; Staniszewski & Lancucki, 2026), while other methods exploit cross-layer redundancy (Chang et al., 2026; Liu et al., 2024b) or prune key channels (Xu et al., 2025); TransMLA (Meng et al., 2025) and MHA2MLA (Ji et al., 2025) go further, converting pretrained models to latent attention through fine-tuning. Eviction methods instead discard less important tokens (Xiao et al., 2024b; Zhang et al., 2023; Li et al., 2024). Beyond algorithms, compressed

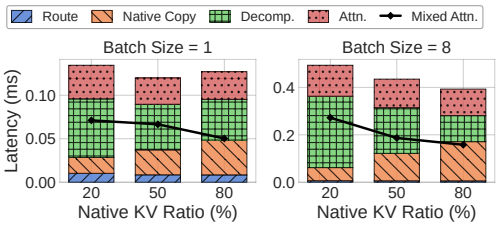

Figure 8: Decode attention latency of materialize-then-attend (bars) and direct mixed attention (line).

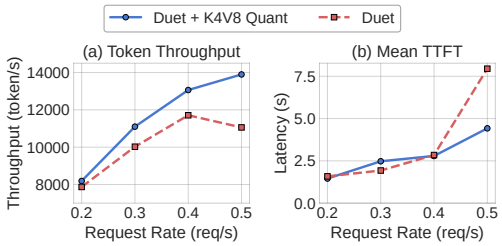

Figure 9: Multi-turn serving of Qwen3-4B with and without K4V8 quantization of the latents.

Table 3: Ablation of the mixed attention operator. Single-layer decode latency in ms.

Method	BS=1	BS=8
Materialize-then-Attn	0.052 (1.00×)	0.393 (1.00×)
+ Direct Mixed	0.035 (1.47×)	0.246 (1.60×)
+ Residency-Aware	**0.031 (1.69×)**	**0.239 (1.65×)**

Table 4: One-time offline calibration cost with 500K calibration tokens.

Model	GPU	Time (s)
Qwen3-4B	A6000	101.9
Llama-3.1-8B	PRO 6000D	42.0
Mistral-Nemo-12B	PRO 6000D	75.4

KV has been accelerated with dedicated kernels (Cheng et al., 2025; Jiang et al., 2025) and used to reduce KV storage, transfer, and memory in serving systems (Liu et al., 2024c; Xiang et al., 2025; Mi et al., 2026; Guo & Joshi, 2026; Liu et al., 2026; Zhang et al., 2025b). Unlike these works, Duet requires no fine-tuning and makes compressed KV a first-class execution-time representation under an external sparse selector. It is also orthogonal to quantization, which reduces bits per element rather than elements per token, so its BF16 latents can be further quantized.

KV offloading. KV offloading relaxes GPU capacity limits by keeping part of the cache in host memory and transferring it on demand. General-purpose systems move KV at layer or head granularity (Sheng et al., 2023; Luo et al., 2025), whereas sparse attention makes offloading far more practical, since only selected entries need to be transferred. Sparse offloading systems differ mainly in how they locate these entries: by speculating on upcoming layers (Lee et al., 2024; Hao et al., 2025), by keeping block summaries or hot entries on the GPU (Xiao et al., 2024a; Chen et al., 2024; Lee et al., 2025; Xie et al., 2026), or by vector search and sampling over host-resident KV (Zhang et al., 2025a; Liu et al., 2025; Chen et al., 2026; 2025). ShadowKV (Sun et al., 2025) further pairs offloading with compression, keeping low-rank keys on the GPU while fetching values. These systems relocate KV across memory tiers, whereas Duet shrinks it in place. The two are therefore complementary: offloading Duet's latents would cut host memory and per-block transfer by the compression ratio, and the mixed operator could consume them directly.

6 Conclusion and Limitations

We presented Duet, a serving-native KV compression framework for dynamic sparse-attention LLMs. Sparse attention reduces how much KV each query reads but not how much KV must stay resident, and existing KV compression methods still leave open how compressed KV should be created, executed, and managed during online serving. Duet closes this gap by making compressed KV a first-class execution-time representation while preserving the semantics of the underlying sparse selector. It combines a joint-head latent representation, a mixed native–latent operator, and a dual-representation runtime. Extensive studies show that Duet delivers substantial throughput improvements without sacrificing task accuracy, translating KV memory savings into practical serving gains.

Duet still leaves several directions open for future work. Our experiments show that its low-rank compression composes well with KV quantization, but how to jointly select the compression ratio and bit width, possibly per layer or per workload, to reach the best accuracy–efficiency trade-off remains an open problem. In addition, the native pool is currently sized statically from the serving configuration; adapting the split between native and compressed capacity to the runtime load, for example by enlarging the native pool when few requests are active and shrinking it under memory pressure, could further improve the trade-off between accuracy and throughput.

AI USE STATEMENT

In this work, we did not use generative AI tools for tasks with required disclosure. Additionally, we used generative AI tools to search for and identify related work, to polish the writing for readability, and to assist with figure creation, including adjusting figure layout and styling. We have reviewed all AI-assisted work. All literature found with AI assistance was read by the authors, and every reference was checked against its original source. We verified all technical descriptions and numerical results, and all figures are plotted from our own measured data. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. It studies system-level efficiency for LLM inference and involves no human subjects, crowdsourcing, or personal data; all experiments use publicly available models and benchmarks under their respective licenses. By lowering the memory cost of long-context serving, Duet may make LLM deployment more accessible, which also inherits the broader risks of LLMs, including hallucination, bias, and potential misuse. Moreover, KV compression is a lossy approximation: although Duet preserves the sparse-attention selector and keeps average accuracy close to the uncompressed baselines, it may still alter model outputs on individual inputs. Duet does not make model outputs more reliable, and we encourage practitioners to validate compressed models on their target tasks before deployment, particularly in high-stakes applications.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide a detailed description of both the method and the experimental setup. The complete design of Duet is described in section 3, covering the offline calibration and post-RoPE joint-head compression (subsection 3.1), the mixed native–latent sparse attention operator (subsection 3.2), and the dual-representation KV runtime (subsection 3.3). For the experiments, we specify the models, hardware, baselines, and hyperparameters of Duet in Table 4, and provide the calibration corpus, the shared serving configuration, and the settings of all three workloads in subsection D.1. Every LongBench and RULER subtask is described together with its source in subsection D.2. Beyond the main results, we report the full RULER results, additional serving metrics, and the effect of the compression ratio and quantization in Appendix F, Appendix E, and Appendix G, respectively. Finally, the cost analyses that motivate our design choices are derived step by step, with all assumptions stated, in Appendix B and Appendix C.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 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, 2024.

Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. RocketKV: Accelerating long-context LLM inference via two-stage KV cache compression. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 3358–3392. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/behnam25a.html.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed Abdelfattah, and Kai-Chiang Wu. Palu: Kv-cache compression with low-rank projection. In *International Conference on Learning Representations*, volume 2025, pp. 50222–50249, 2025.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Hung-Yueh Chiang, Yash Akhauri, Xilai Dai, Huiqiang Jiang, Yucheng Li, Luis Ceze, Kai-Chiang Wu, and Mohamed S. Abdelfattah. xKV: Cross-layer KV-cache compression via aligned singular vector extraction. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=XDb69lykA8.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. Arkvale: Efficient generative llm inference with recallable key-value eviction. *Advances in Neural Information Processing Systems*, 37:113134–113155, 2024.

Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jing Liu, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Bailu Ding, Xiao Yan, Jiawei Jiang, Chen Chen, Mingxing Zhang, Cheng Li, Yuqing Yang, Fan Yang, and Mao Yang. Retroinfer: A vector storage engine for scalable long-context llm inference. *Proc. VLDB Endow.*, 19(5):1016–1031, May 2026. ISSN 2150-8097. doi: 10.14778/3796195.3796212. URL https://doi.org/10.14778/3796195.3796212.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, et al. Magicpig: Lsh sampling for efficient llm generation. In *International Conference on Learning Representations*, volume 2025, pp. 44169–44190, 2025.

Feng Cheng, Cong Guo, Chiyue Wei, Junyao Zhang, Changchun Zhou, Edward Hanson, Jiaqi Zhang, Xiaoxiao Liu, Hai Li, and Yiran Chen. Ecco: Improving memory bandwidth and capacity for llms via entropy-aware cache compression. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 793–807, 2025.

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 Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365/.

Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1074–1084, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1102. URL https://aclanthology.org/P19-1102/.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In Lu Wang, Jackie Chi Kit Cheung, Giuseppe Carenini, and Fei Liu (eds.), *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 70–79, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5409. URL https://aclanthology.org/D19-5409/.

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.

Yipin Guo and Siddharth Joshi. Splitzip: Ultra fast lossless kv compression for disaggregated llm serving. *arXiv preprint arXiv:2605.01708*, 2026.

Jitai Hao, Yuke Zhu, Tian Wang, Jun Yu, Xin Xin, Bo Zheng, Zhaochun Ren, and Sheng Guo. Omnikv: Dynamic context selection for efficient long-context llms. In *International Conference on Learning Representations*, volume 2025, pp. 87443–87464, 2025.

Wei He, Kai Liu, Jing Liu, Yajuan Lyu, Shiqi Zhao, Xinyan Xiao, Yuan Liu, Yizhong Wang, Hua Wu, Qiaoqiao She, Xuan Liu, Tian Wu, and Haifeng Wang. DuReader: a Chinese machine reading comprehension dataset from real-world applications. In Eunsol Choi, Minjoon Seo, Danqi Chen, Robin Jia, and Jonathan Berant (eds.), *Proceedings of the Workshop on Machine Reading for Question Answering*, pp. 37–46, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2605. URL https://aclanthology.org/W18-2605/.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong (eds.), *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. Flashdecoding++: Faster large language model inference with asynchronization, flat gemm optimization, and heuristics. *Proceedings of Machine Learning and Systems*, 6:148–161, 2024.

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. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

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

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1419–1436, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.112. URL https://aclanthology.org/2021.naacl-main.112/.

Tao Ji, Bin Guo, Yuanbin Wu, Qipeng Guo, Lixing Shen, Zhan Chen, Xipeng Qiu, Qi Zhang, and Tao Gui. Towards economical inference: Enabling deepseek's multi-head latent attention in any transformer-based llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33313–33328, 2025.

Bo Jiang, Taolue Yang, Youyuan Liu, Chengming Zhang, Xubin He, and Sian Jin. Kvcomp: A high-performance, llm-aware, lossy compression framework for kv cache. *arXiv preprint arXiv:2509.00579*, 2025.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

Greg Kamradt. Needle in a haystack. https://github.com/gkamradt/needle-in-a-haystack, 2023. Accessed: 2026-09-25.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018. doi: 10.1162/tacl_a_00023. URL https://aclanthology.org/Q18-1023/.

Rahul Krishnan and Volker Schulz. A jolt for the kv cache: Near-lossless kv cache compression via joint tucker and jl-residual allocation for llms. *arXiv preprint arXiv:2607.12550*, 2026.

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 *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Heejun Lee, Geon Park, Jaduk Suh, and Sung Ju Hwang. Infinitehip: Extending language model context up to 3 million tokens on a single gpu. *arXiv preprint arXiv:2502.08910*, 2025.

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.

Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pp. 1–7, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1072228.1072378. URL https://doi.org/10.3115/1072228.1072378.

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. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024b.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, et al. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *Advances in Neural Information Processing Systems*, 38: 54358–54385, 2025.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024c.

Zedong Liu, Xinyang Ma, Dejun Luo, Hairui Zhao, Bing Lu, Wenjing Huang, Yida Gu, Xingchen Liu, Zheng Wei, Jinyang Liu, et al. Kvserve: Service-aware kv cache compression for communication-efficient disaggregated llm serving. In *Proceedings of the ACM SIGCOMM 2026 Conference*, pp. 1–15, 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. 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. 32332–32344. PMLR, 21–27 Jul 2024d. URL https://proceedings.mlr.press/v235/liu24bz.html.

Cheng Luo, Zefan Cai, Hanshi Sun, Jinqi Xiao, Bo Yuan, Wen Xiao, Junjie Hu, Jiawei Zhao, Beidi Chen, and Anima Anandkumar. Headinfer: Memory-efficient llm inference by head-wise offloading. *arXiv preprint arXiv:2502.12574*, 2025.

Fanxu Meng, Pingzhi Tang, Xiaojuan Tang, Zengwei Yao, Xing Sun, and Muhan Zhang. Transmla: Multi-head latent attention is all you need. *arXiv preprint arXiv:2502.07864*, 2025.

Liang Mi, Weijun Wang, Jinghan Chen, Ting Cao, Haipeng Dai, and Yunxin Liu. Efficient remote kv cache reuse with gpu-native video codec. In *Proceedings of the ACM SIGCOMM 2026 Conference*, pp. 30–45, 2026.

Mistral AI Team. Mistral nemo. https://mistral.ai/news/mistral-nemo, July 2024. Accessed: 2026-09-25.

Junlin Mu, Hantao Huang, Jihang Zhang, Minghui Yu, Tao Wang, and Yidong Li. Sals: Sparse attention in latent space for kv cache compression. *Advances in Neural Information Processing Systems*, 38:384–403, 2025.

Dipkumar Patel. Turboangle: Near-lossless kv cache compression via uniform angle quantization. *arXiv preprint arXiv:2603.27467*, 2026.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124/.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for kv cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15332–15344, 2024.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Konrad Staniszewski and Adrian Lancucki. Kv cache transform coding for compact storage in llm inference. In *International Conference on Learning Representations*, volume 2026, pp. 37768–37815, 2026.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57355–57373. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/sun25b.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 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.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Han Wu, Mingjie Zhan, Haochen Tan, Zhaohui Hou, Ding Liang, and Linqi Song. VCSUM: A versatile Chinese meeting summarization dataset. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 6065–6079, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.377. URL https://aclanthology.org/2023.findings-acl.377/.

Xingyu Xiang, Raj Joshi, Yuhan Liu, Jiayi Yao, Chenxingyu Zhao, Junchen Jiang, Yang Zhou, Eddie Kohler, and Minlan Yu. Shadowserve: Interference-free kv cache fetching for distributed prefix caching. *arXiv preprint arXiv:2509.16857*, 2025.

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. *Advances in neural information processing systems*, 37:119638–119661, 2024a.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024b.

Zhiqiang Xie, Zhangheng Huang, Tingwei Huang, Ziyi Xu, Ruiyang Ma, and Christos Kozyrakis. Hisparse: Scaling sparse-attention decoding with hierarchical kv cache management. *arXiv preprint arXiv:2608.07009*, 2026.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. Think: Thinner key cache by query-driven pruning. In *International Conference on Learning Representations*, volume 2025, pp. 56691–56709, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, 2025.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, volume 2026, pp. 56418–56439, 2026.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30, 2025a.

Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John CS Lui, and Haibo Chen. Diffkv: Differentiated memory management for large language models with parallel kv compaction. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 431–445, 2025b.

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. *Advances in neural information processing systems*, 36:34661–34710, 2023.

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 Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5905–5921, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.472. URL https://aclanthology.org/2021.naacl-main.472/.

A Per-Layer Compression Error and Selector Perturbation

Joint-head compression lowers reconstruction error, but computing block summaries from its reconstructed keys perturbs the sparse selector. We quantify both effects on Qwen3-4B, which has 36 layers and $H_{\text{kv}} = 8$ KV heads of dimension $d = 128$, so $D = 1024$. The default joint-head rank is 256 ($D/4$), corresponding to a compression ratio of $1/4$.

Per-layer reconstruction error. For each layer, we fit SVD projections to the post-RoPE keys and values and compare two schemes at the same compression ratio ρ, *i.e.*, the same number of latent elements per token. Per-head compression fits a separate projection of rank ρd to each KV head, whereas joint-head compression fits one projection of rank ρD to the concatenation of all heads. We report the relative Frobenius error $\|X - \hat{X}\|_F / \|X\|_F$, where X stacks the keys or values of a layer and $\hat{X}$ is its reconstruction. Figure 10 shows this error for every layer, and Table 1 averages it over layers. Joint-head compression is more accurate at every layer and compression ratio, for both keys and values, with a particularly large margin for values. Its advantage therefore reflects redundancy shared across heads throughout the model rather than a few favorable layers.

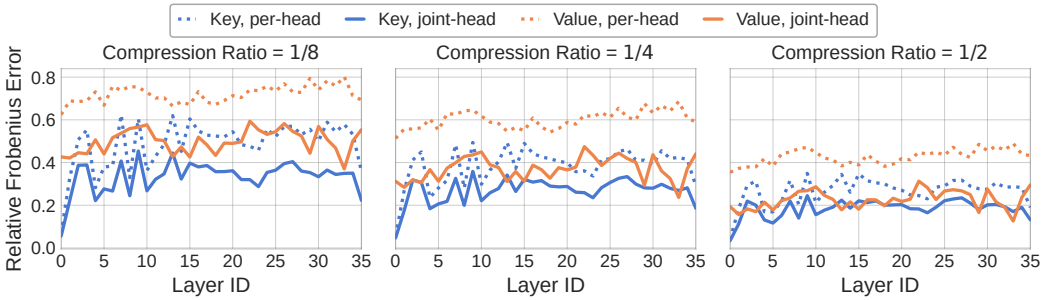

Figure 10: Per-layer relative Frobenius error of per-head and joint-head SVD compression on Qwen3-4B at compression ratios $1/8$, $1/4$ (rank 256), and $1/2$. Table 1 reports the average over layers.

Selector perturbation. We next measure how the selector changes if it reads block summaries computed from reconstructed keys. For each layer and KV head, we run Quest twice with the same budget: once with min/max summaries of the original keys, selecting the block set $\mathcal{S}$, and once with summaries of keys reconstructed from joint-head post-RoPE latents of rank $r_K = 256$, selecting $\hat{\mathcal{S}}$. For each query, the *selection disagreement* $1 - |\mathcal{S} \cap \hat{\mathcal{S}}|/|\mathcal{S}|$ is the fraction of originally selected blocks that are replaced, and the *attention-mass loss* $\big(A(\mathcal{S}) - A(\hat{\mathcal{S}})\big)/A(\mathcal{S})$ is the fraction of the attention captured by the original selection that is lost. Here $A(\cdot)$ sums the full-attention probabilities over the tokens of the given blocks, computed with the original keys and averaged over the query heads of the GQA group. Figure 11 reports the maximum of both quantities over all evaluated queries for every layer and KV head.

The perturbation is widespread rather than confined to a few heads: at every layer and KV head, some queries have more than 10% of their selected blocks replaced. Selection errors also differ in kind from payload errors. DSA reads only the selected blocks, so a block dropped by the selector loses its attention entirely, whereas reconstruction error only perturbs the blocks that are attended. Slightly negative mass losses occur because the Quest score upper-bounds the attention logit rather than estimating it exactly, so the original selection is not always mass-optimal. These results motivate computing summaries from the original keys, which leaves the selection unchanged by construction.

B Cost of Pre- and Post-RoPE Key Compression

Attention over compressed keys. Let q be a post-RoPE query of a query head mapped to KV head h, $z_{K,t} \in \mathbb{R}^{r_K}$ the joint-head key latent of token t, $B_{K,h} \in \mathbb{R}^{r_K \times d}$ the key compression basis of KV head h, and $R_t \in \mathbb{R}^{d \times d}$ the RoPE rotation at position t. Pre-RoPE compression stores the latent of the unrotated key, whose rotated reconstruction is $R_t B_{K,h}^\top z_{K,t}$, whereas post-RoPE compression

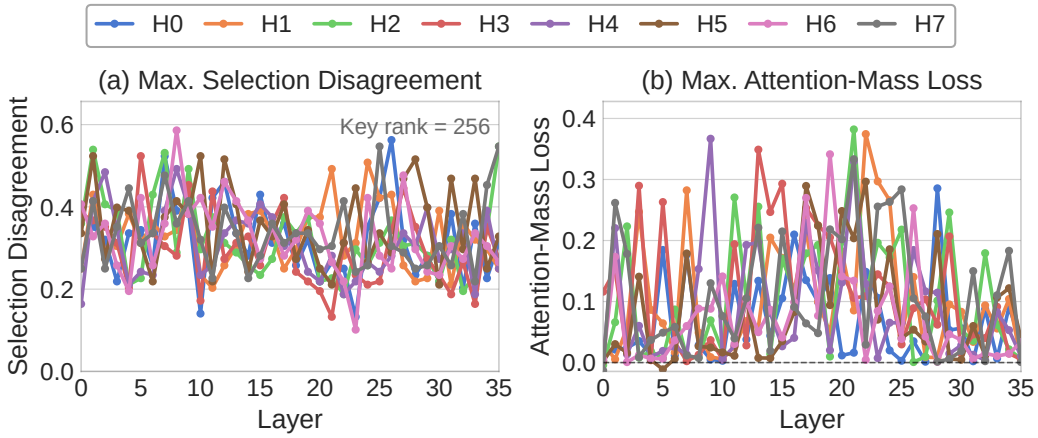

Figure 11: Maximum perturbation of Quest selection on Qwen3-4B when block summaries are computed from keys reconstructed at rank 256, for each layer and KV head (H0–H7). (a) Fraction of selected blocks that change. (b) Relative loss of the attention mass captured by the selected blocks.

stores the latent of the rotated key. The attention scores are

$$\text{pre-RoPE:} \quad q^\top R_t B_{K,h}^\top z_{K,t} = \left(B_{K,h} R_t^\top q\right)^\top z_{K,t}, \tag{5}$$

$$\text{post-RoPE:} \quad q^\top B_{K,h}^\top z_{K,t} = \left(B_{K,h}\, q\right)^\top z_{K,t}. \tag{6}$$

In the post-RoPE form, the projected query $B_{K,h}q$ does not depend on t: it is computed once per query, and each selected key is scored directly on its latent with r_K multiply-adds. In the pre-RoPE form, R_t lies between the query and the basis, so the projected query changes with the key position. Each selected key must then be reconstructed and rotated, which costs $r_K d$ multiply-adds per KV head, or scored with a query re-projected for its position, which costs the same per query head. We adopt reconstruction, which is cheaper because the query heads of a GQA group share it. Values are not rotated, so both forms aggregate them in latent space and apply $B_{V,h}^\top$ once to the result, as in Equation 3.

Cost model. We count the cost of one decode query in a single attention layer of Qwen3-4B, with $H_q = 32$ query heads, $H_{\text{kv}} = 8$ KV heads of dimension $d = 128$, $r_K = r_V = 256$, and $T = 2048$ selected tokens per KV head. Latents are stored in BF16, and our pre-RoPE implementation materializes the reconstructed keys in an FP32 workspace. A multiply-add counts as two FLOPs; softmax, log-sum-exp, and split-K merging are omitted, as they are small relative to the matrix products. The two paths require

$$F_{\text{post}} = 2H_q d r_K + 2H_q T r_K + 2H_q T r_V + 2H_q r_V d \approx 71.3 \text{ MFLOPs}, \tag{7}$$

$$F_{\text{pre}} = 2H_{\text{kv}} T r_K d + 3H_{\text{kv}} T d + 2H_q T d + 2H_q T r_V + 2H_q r_V d \approx 1.13 \text{ GFLOPs}, \tag{8}$$

where $3H_{\text{kv}}Td$ counts split-half RoPE, which spends four multiplications and two additions on each pair of dimensions. Key reconstruction accounts for 95% of F_{pre}. Scoring on latents costs twice as much as scoring native keys, since $r_K = 2d$, but remains small next to reconstruction.

For memory traffic, each KV head reads the key and value latents of its own selected tokens, 8 MiB each, so the post-RoPE path moves 16 MiB. The pre-RoPE path reads the same latents, but also writes the 8 MiB FP32 key workspace, reads and rewrites it for RoPE, and reads it again for attention, for 48 MiB in total. The count excludes reused or small tensors, namely the bases, RoPE tables, queries, outputs, and merge buffers. Table 6 breaks down both paths. FLOPs and traffic both grow linearly with batch size, since each request reads its own selected tokens, so the arithmetic intensity does not depend on batch size.

Measured efficiency. On an RTX A6000 at batch size 1, the post-RoPE and pre-RoPE kernels take 0.0239 ms and 0.0708 ms, the 3× gap in Figure 1(b). Their effective bandwidths, 702 and 711 GB/s, are both close to the 768 GB/s peak of the GPU, and their arithmetic intensities, 4.25 and

Table 6: FLOPs and dominant memory traffic of one decode query in a single attention layer of Qwen3-4B with 2048 selected tokens per KV head.

Path	Operation	FLOPs	Traffic
Post-RoPE	Query projection, $2H_q d r_K$	2.10 M	–
	Latent scoring, $2H_q T r_K$	33.55 M	8 MiB
	Latent aggregation, $2H_q T r_V$	33.55 M	8 MiB
	Value projection, $2H_q r_V d$	2.10 M	–
	Total	71.30 M	16 MiB
Pre-RoPE	Key reconstruction, $2H_{\mathrm{kv}} T r_K d$	1073.74 M	16 MiB
	RoPE, $3H_{\mathrm{kv}} T d$	6.29 M	16 MiB
	Native scoring, $2H_q T d$	16.78 M	8 MiB
	Latent aggregation, $2H_q T r_V$	33.55 M	8 MiB
	Value projection, $2H_q r_V d$	2.10 M	–
	Total	1132.46 M	48 MiB

22.5 FLOP/B, stay below its compute-to-bandwidth ratio. Both kernels are thus memory-bound, and the latency gap follows the $3\times$ traffic gap rather than the $16\times$ FLOP gap. The two gaps differ in nature. The workspace traffic depends on the implementation and could be reduced by fusing reconstruction, rotation, and attention into one kernel, whereas the reconstruction FLOPs are intrinsic to pre-RoPE compression and grow with the number of selected tokens. Post-RoPE compression avoids both.

C Direct Latent Execution versus Materialize-Once for Extend Prefill

Cost model. Consider an extend-prefill chunk of T query tokens whose cached prefix contains N compressed-only tokens, of which each query selects S. Selection is shared by the $g = H_q/H_{\mathrm{kv}}$ query heads of a GQA group. We count the compressed-history FLOPs per layer, the quantity plotted in Figure 3(a), with a multiply-add as two FLOPs; attention to native blocks and to the chunk itself is identical in both paths and is omitted.

Direct latent execution, labeled *Fused* in Figure 3(a), evaluates Equation 3 for every query: each query head projects its query, scores and aggregates S latents, and applies the value projection once, for a total of

$$F_{\mathrm{dir}} = 2T H_q \,(S + d)(r_K + r_V). \tag{9}$$

Materialize-once reconstructs the keys and values of all N compressed tokens for all KV heads and then runs native sparse attention, which spends $4d$ FLOPs per query head on each selected token:

$$F_{\mathrm{mat}} = 2N H_{\mathrm{kv}}\, d\,(r_K + r_V) + 4T H_q S d. \tag{10}$$

Break-even chunk size. The ratio plotted in Figure 3(a) is

$$\frac{F_{\mathrm{dir}}}{F_{\mathrm{mat}}} = \frac{g\,(T/N)\,(S + d)(r_K + r_V)}{d\,(r_K + r_V) + 2g\,(T/N)\,Sd}. \tag{11}$$

It depends on the chunk and prefix lengths only through T/N, so the curves for different prefixes are shifted copies of one another. The ratio equals one at

$$T^* = \frac{N\,d\,(r_K + r_V)}{g\big[(S + d)(r_K + r_V) - 2Sd\big]}. \tag{12}$$

Chunks shorter than T^* require fewer FLOPs on the direct path, whereas longer chunks favor materialize-once, whose advantage approaches $(1 + d/S)(r_K + r_V)/(2d)$ as the chunk grows. Figure 3(a) uses $g = 4$ and $d = 128$ as in Qwen3-4B, $r_K = r_V = 256$ for the compression ratio of $1/4$, and a 2048-token budget. Then $T^* \approx N/36$, *i.e.*, about 230, 460, and 910 tokens for compressed prefixes of 8K, 16K, and 32K tokens, where the curves cross the equal-FLOPs line, and the advantage approaches $2.1\times$.

Interpretation. The two paths trade a per-access cost against a one-time cost. The direct path never reconstructs KV, but it pays the latent cost on every pair of query and selected token: with $r_K = r_V = 2d$, a latent access costs twice a native one, and each query adds its own projections. This overhead grows with TS. Materialize-once instead pays a reconstruction cost proportional to N once and amortizes it over all T queries of the chunk. Decode, with a single query per request, lies far below T^* for any practical prefix and therefore consumes latents directly, whereas extend prefill amortizes materialization across its chunk.

D Evaluation Details

D.1 Workloads and Configurations

Calibration and serving configuration. The compression projections of Duet are calibrated once per model, offline, on 500K tokens drawn from sequences of up to 32K tokens, and are kept fixed across all experiments. All systems share one serving configuration: a maximum model length of 32K tokens, at most 160 concurrent sequences, a GPU memory utilization of 0.7, *i.e.*, the fraction of GPU memory that vLLM may use for weights, activations, and the KV cache, and CUDA Graphs and prefix caching enabled. The maximum model length and the number of concurrent sequences set $L_{\max}$ and $S_{\max}$ of the native pool in Duet (Section 3.3). With 16-token blocks, $n_{\text{sink}} = n_{\text{win}} = 2$, and $W = 512$, the pool therefore reserves $2048 + 160 \times 4 + 512 = 3200$ blocks, or 51,200 tokens of native KV.

Workloads. The efficiency experiments use three workloads. Offline batch inference submits all requests of a batch at once (Figure 5), each with up to 16,384 input tokens and up to 1,024 output tokens. Single-turn online serving uses requests of the same lengths, which arrive as a Poisson process at the request rates in Figure 6. Multi-turn online serving issues three-turn conversations (Figure 7): the first turn carries up to 14,000 input tokens, each later turn appends up to 1,024 new input tokens, and every turn generates up to 256 output tokens. A conversation sends its next turn 8 seconds after the previous response completes, and conversations arrive with inter-arrival times drawn from a Gamma distribution with burstiness 1, which reduces to exponential inter-arrival times. Each later turn is therefore an extend prefill of about 1K new tokens over a cached prefix of 14K–16K tokens, which exercises prefix caching and the materialize-once path of Duet.

D.2 Benchmark Details

LongBench. LongBench (Bai et al., 2024) is a bilingual benchmark for long-context understanding. We evaluate the 17 tasks in Table 7, which span five categories. Single-document QA answers questions about one long document: stories and scripts in NarrativeQA, NLP papers with questions from practitioners in Qasper, and articles from diverse fields in MultiFieldQA. Multi-document QA requires multi-hop reasoning over several Wikipedia passages in HotpotQA, 2WikiMultihopQA, and MuSiQue, and answering Chinese questions from retrieved web documents in DuReader. Summarization condenses government reports in GovReport, meeting transcripts for a given query in QMSum, multiple news articles in MultiNews, and Chinese meeting transcripts in VCSum. Few-shot learning provides in-context examples for question classification into 50 classes in TREC, Chinese news classification into 24 classes in LSHT, reading comprehension in TriviaQA, and dialogue summarization in SAMSum. The synthetic tasks count the distinct paragraphs of a repetitive article in PassageCount and identify which of several paragraphs a given summary describes in PassageRetrieval. The average in Table 2 is the macro average over the 17 tasks.

RULER. RULER (Hsieh et al., 2024) is a synthetic benchmark that tests retrieval, multi-hop tracing, and question answering over long inputs. We use the nine tasks in Table 8. The needle-in-a-haystack (NIAH) tasks, which extend the original NIAH test (Kamradt, 2023), hide key–value pairs (needles) in a long filler text (haystack) and ask for the value of a given key; they differ in the haystack, the value type, and the number of distractor or queried needles. Variable tracking hides a chain of variable assignments and asks for all variables that hold a given value. The QA tasks place the paragraphs that answer a question among distractor paragraphs from the same dataset. NIAH and variable tracking are scored by the fraction of reference answers that appear in the output, and QA is scored as correct if the output contains any reference answer.

Table 7: LongBench tasks used in our evaluation.

Category	Task	Metric
Single-doc QA	NarrativeQA (Kočiský et al., 2018)	F1
	Qasper (Dasigi et al., 2021)	F1
	MultiFieldQA (Bai et al., 2024)	F1
Multi-doc QA	HotpotQA (Yang et al., 2018)	F1
	2WikiMultihopQA (Ho et al., 2020)	F1
	MuSiQue (Trivedi et al., 2022)	F1
	DuReader (He et al., 2018)	ROUGE-L
Summarization	GovReport (Huang et al., 2021)	ROUGE-L
	QMSum (Zhong et al., 2021)	ROUGE-L
	MultiNews (Fabbri et al., 2019)	ROUGE-L
	VCSum (Wu et al., 2023)	ROUGE-L
Few-shot learning	TREC (Li & Roth, 2002)	Accuracy
	TriviaQA (Joshi et al., 2017)	F1
	SAMSum (Gliwa et al., 2019)	ROUGE-L
	LSHT (Bai et al., 2024)	Accuracy
Synthetic	PassageCount (Bai et al., 2024)	Accuracy
	PassageRetrieval (Bai et al., 2024)	Accuracy

Table 8: RULER tasks used in our evaluation.

Task	Description
S1	Single needle with a numeric value in a haystack of repeated noise sentences.
S2	Single needle with a numeric value in a haystack of essays.
S3	Single needle with a UUID value in a haystack of essays.
MK1	One queried needle among three distractor needles with different keys in a haystack of essays.
MK2	One queried needle in a haystack composed entirely of distractor needles.
MQ	Four needles with different keys in a haystack of essays; the values of all four must be retrieved.
VT	A four-hop chain of variable assignments in noise sentences; all variables holding the queried value must be returned.
SQuAD	Single-hop QA on SQuAD (Rajpurkar et al., 2018) with distractor paragraphs.
HotpotQA	Multi-hop QA on HotpotQA (Yang et al., 2018) with distractor paragraphs.

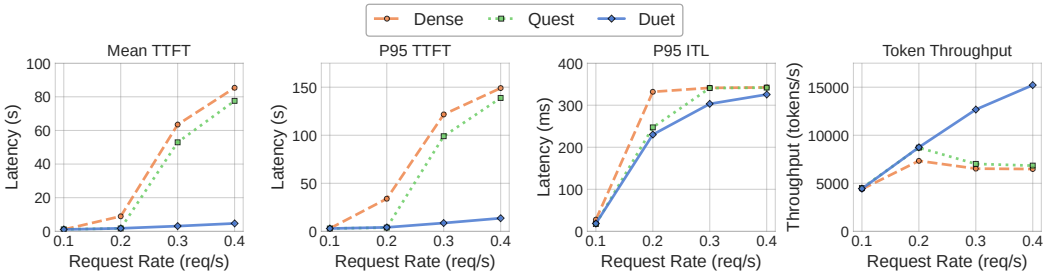

Figure 12: Multi-turn online serving of Llama-3.1-8B on an RTX PRO 6000D.

E Serving Metrics on More Models

Figure 12 and Figure 13 repeat the online serving experiments of Table 4 on the other two models under the same workloads and configurations, and show the same trends. In multi-turn serving of Llama-3.1-8B, all three systems perform similarly at low load, but beyond 0.2 req/s the throughput of dense attention and Quest saturates and then declines, whereas Duet keeps scaling and reaches about 2.3× the throughput of Quest at 0.4 req/s. Its mean TTFT stays below 5 seconds at all request rates, compared with more than 75 seconds for Quest at the highest rate. In single-turn serving of Mistral-Nemo-12B, all systems are capacity-bound over the tested rates; Duet sustains about 1.3× and 1.8× the token throughput of Quest and dense attention and lowers mean TTFT by 2.6–3.2× relative to Quest, at the cost of a slightly higher ITL.

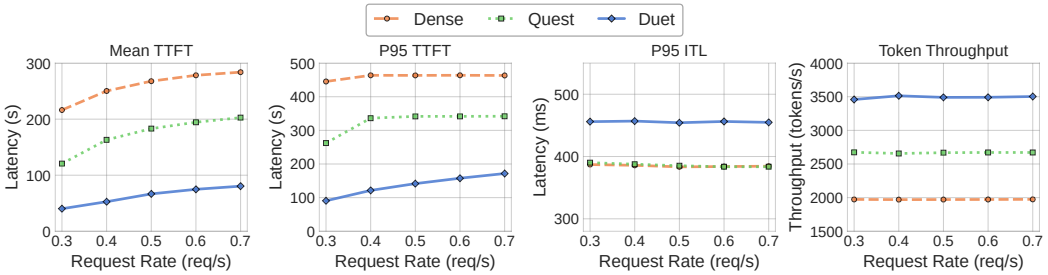

Figure 13: Single-turn online serving of Mistral-Nemo-12B on an RTX PRO 6000D.

Table 9: Accuracy on RULER tasks. Higher is better. Avg. is the macro average over the nine tasks shown.

Model	Method	S1	S2	S3	MK1	MK2	MQ	VT	SQuAD	HotpotQA	Avg.
Qwen3-4B	Dense	100.00	100.00	100.00	100.00	100.00	100.00	100.00	81.00	56.00	93.00
	Quest	100.00	100.00	100.00	100.00	100.00	100.00	100.00	82.00	56.00	93.11
	Duet	100.00	100.00	100.00	100.00	99.00	99.00	99.00	80.00	55.00	92.44
Llama-3.1-8B	Dense	100.00	100.00	100.00	100.00	100.00	100.00	78.20	88.00	62.00	92.02
	Quest	100.00	100.00	100.00	100.00	100.00	100.00	78.40	88.00	62.00	92.04
	Duet	100.00	100.00	100.00	100.00	97.00	97.50	81.60	89.00	63.00	92.01
Mistral-Nemo-12B	Dense	100.00	100.00	100.00	97.00	100.00	98.50	37.40	81.00	57.00	85.66
	Quest	100.00	100.00	100.00	97.00	100.00	98.75	37.40	81.00	57.00	85.68
	Duet	100.00	100.00	97.00	96.00	99.00	92.75	29.20	82.00	57.00	83.66

Abbreviations: S1–S3=NIAH Single 1–3, MK1–MK2=NIAH Multikey 1–2, MQ=NIAH Multiquery, and VT=Variable Tracking.

F RULER Results

Table 9 reports accuracy on RULER. As on LongBench, Quest matches dense attention on all three models, so the remaining differences isolate the effect of compression. Duet stays within 0.1 points of Quest on Llama-3.1-8B and within 0.7 points on Qwen3-4B. Single-needle retrieval is lossless on both models, with Duet retrieving every needle on S1–S3, consistent with the unchanged selector, which still locates the blocks that hold the needle. Multi-key and multi-query retrieval also remain at or above 97%, and on the QA tasks Duet stays within two points of Quest, even slightly surpassing it on Llama-3.1-8B, where it also improves variable tracking by 3.2 points. On Mistral-Nemo-12B, the average gap of about two points comes mainly from variable tracking, on which even dense attention reaches only 37.4%, and from multi-query retrieval, which remains above 92%; its QA accuracy matches or exceeds that of Quest.

G Effect of Compression Ratio and Quantization

Table 10 evaluates Duet on LongBench under two compression ratios, $r_K = r_V = D/4$ (the default) and $D/8$, each with BF16 latents or with key and value latents further quantized to 4 and 8 bits (K4V8). All other settings follow Table 4.

Compression ratio. Halving the ratio from 1/4 to 1/8 lowers the LongBench average by 1.9, 2.6, and 2.1 points on Qwen3-4B, Llama-3.1-8B, and Mistral-Nemo-12B, respectively. The loss is uneven across tasks. Few-shot learning, synthetic tasks, and multi-hop QA (HotpotQA, 2WikiMQA, and MuSiQue) remain within about one point, since they depend mainly on locating the relevant blocks, which the unchanged selector preserves. The drops instead concentrate on single-document QA and summarization, such as MultiFieldQA, NarrativeQA, and GovReport, which require fine-grained content from many selected blocks and are therefore more sensitive to the rank of the latent space.

Quantization. In contrast, quantization introduces little additional error. At a ratio of 1/4, K4V8 changes the average by -0.54, $+0.01$, and -0.33 points on the three models, and at a ratio of 1/8 by -0.20, $+0.18$, and -0.30 points, with most individual tasks staying within one point. Low-rank projection reduces the dimension of each KV state while quantization reduces the bits per element, and the two compose with little interference.

Table 10: LongBench results of Duet variants with different compression ratios and KV quantization. Higher is better.

(a) Summarization, few-shot, and synthetic tasks

Model	Method	Summarization				Few-shot learning				Synthetic	
		Gov.	QMSum	MNews	VCSum	TREC	TQA	SAMSum	LSHT	Count	Retr.
Qwen3-4B	Duet (ratio 1/4)	30.18	24.28	25.62	15.15	58.00	89.37	40.79	41.00	1.25	96.50
	Duet (ratio 1/8)	27.68	21.70	23.40	14.47	57.50	89.87	40.71	41.00	1.25	95.50
	Duet (ratio 1/8 + K4V8)	26.79	21.32	23.88	14.70	58.00	89.37	40.07	41.00	1.25	96.00
	Duet (ratio 1/4 + K4V8)	29.75	23.66	25.49	15.53	58.00	89.37	40.93	41.25	1.25	95.50
Llama-3.1-8B	Duet (ratio 1/4)	27.78	24.52	24.22	15.47	20.00	91.42	13.37	0.00	9.18	99.50
	Duet (ratio 1/8)	21.51	21.65	21.05	13.90	17.00	91.58	13.03	0.00	9.68	99.50
	Duet (ratio 1/8 + K4V8)	20.86	21.17	21.02	14.05	19.50	91.42	12.91	0.00	10.25	99.50
	Duet (ratio 1/4 + K4V8)	27.13	24.48	23.74	15.56	20.50	91.58	12.92	0.00	10.32	99.50
Mistral-Nemo-12B	Duet (ratio 1/4)	27.58	21.78	23.63	15.01	11.00	89.54	40.83	28.00	1.00	99.00
	Duet (ratio 1/8)	21.25	20.55	19.93	12.88	12.00	89.41	40.63	27.00	1.00	98.50
	Duet (ratio 1/8 + K4V8)	21.52	19.48	20.08	12.86	12.00	89.82	40.53	27.50	1.00	98.50
	Duet (ratio 1/4 + K4V8)	27.00	21.74	23.54	14.52	13.00	89.19	41.19	27.50	0.50	98.50

(b) Question answering tasks

Model	Method	Single-document QA			Multi-document QA				Avg.
		Qasp.	MFQA	NQA	HQA	2Wiki	MuSiQue	DuR.	
Qwen3-4B	Duet (ratio 1/4)	42.51	46.64	24.70	55.25	43.70	32.52	29.74	41.01
	Duet (ratio 1/8)	37.48	35.44	20.59	54.58	43.60	32.10	28.58	39.14
	Duet (ratio 1/8 + K4V8)	37.99	36.16	16.75	54.58	43.73	32.53	27.93	38.94
	Duet (ratio 1/4 + K4V8)	42.45	43.68	21.28	54.91	43.45	32.58	28.86	40.47
Llama-3.1-8B	Duet (ratio 1/4)	42.25	55.32	29.34	57.80	47.12	31.87	30.43	36.45
	Duet (ratio 1/8)	37.66	47.20	20.33	58.07	45.83	32.39	25.84	33.90
	Duet (ratio 1/8 + K4V8)	37.97	46.49	21.73	57.96	47.40	32.18	24.87	34.08
	Duet (ratio 1/4 + K4V8)	42.86	55.50	29.39	57.38	46.75	32.50	29.69	36.46
Mistral-Nemo-12B	Duet (ratio 1/4)	42.23	57.79	24.52	50.06	46.55	27.18	28.53	37.31
	Duet (ratio 1/8)	36.64	51.77	17.70	49.64	46.97	27.99	25.16	35.24
	Duet (ratio 1/8 + K4V8)	36.19	48.21	18.39	49.97	46.55	26.34	25.03	34.94
	Duet (ratio 1/4 + K4V8)	40.97	55.59	22.41	49.74	47.25	27.76	28.34	36.98

Abbreviations: Qasp.=Qasper, MFQA=MultiFieldQA, NQA=NarrativeQA, HQA=HotpotQA, DuR.=DuReader, Gov.=GovReport, MNews=MultiNews, TQA=TriviaQA, Count=PassageCount, and Retr.=PassageRetrieval-en. Avg. is the macro average over all 17 tasks.