Benchmarking KV-Cache Optimizations across Task Quality and System Performance for Long-Context Serving [Experiment, Analysis & Benchmark]

Nikita Agrawal
University of Bayreuth
Bayreuth, Germany
Nikita.Agrawal@uni-bayreuth.de

Ruben Mayer
University of Bayreuth
Bayreuth, Germany
Ruben.Mayer@uni-bayreuth.de

ABSTRACT

Large language model serving is increasingly limited by KV-cache growth under long-context workloads, yet existing KV-cache compression techniques are difficult to compare because they were evaluated on different models, tasks, budgets, and serving stacks. This paper presents a workload-aware benchmark of representative KV-cache optimization mechanisms spanning quantization, pruning, and merging, including KIVI, TurboQuant, SnapKV, and CaM, evaluated on LongBench-style multi-document QA, single-document QA, few-shot learning, and summarization workloads using Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3. The benchmark measures task quality, mean output throughput, mean time-to-first-token, and realized compression ratio across context-length buckets. The results show that the compression ratio alone is a poor predictor of end-to-end performance. KIVI4 provides the most stable quality across models, SnapKV delivers the strongest long-context throughput, and CaM yields large gains on selected QA workloads but exhibits substantial workload sensitivity in both quality and realized compression ratio. These findings motivate workload-aware selection of KV-cache mechanisms rather than one-size-fits-all compression and provide deployment guidance for long-context serving systems.

PVLDB Reference Format:
Nikita Agrawal and Ruben Mayer. Benchmarking KV-Cache Optimizations across Task Quality and System Performance for Long-Context Serving [Experiment, Analysis & Benchmark]. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/nikagrwal/Benchmarking-KV-Cache-Optimizations-across-Task-Quality-and-System-Performance.

1 INTRODUCTION

The adoption of large language models (LLMs) has grown rapidly across a wide range of data-intensive applications, including document summarization, multi-turn dialogue, and code analysis [3, 13]. As these applications increasingly rely on long-context inputs, efficient LLM serving has emerged as a critical systems challenge. In particular, the Key–Value (KV) cache used in LLMs grows linearly with input length, leading to substantial memory and bandwidth overhead during inference [31]. This has motivated a surge of recent work on KV cache optimization techniques, such as quantization, pruning, and merging, to enable scalable long-context serving.

The data management community has recently begun to address these challenges by rethinking LLM serving stacks and system-level optimizations. Previous works have explored enhancing LLM serving by improving efficiency in inference pipelines, memory-aware scheduling, and hardware-conscious optimizations [25, 28, 35, 44]. A growing research area addresses KV cache compression techniques to reduce memory footprint and improve decoding efficiency [40]. However, the proposed methods have only been evaluated in isolation or compared with few other techniques. The corresponding publications make a direct comparison difficult, as they use different models, datasets, compression budgets, and system configurations. Thus, it remains unclear how these methods compare among themselves, and which methods are most suitable for different workloads.

This lack of a unified and workload-aware evaluation is a key gap in the literature. In particular, existing studies often focus on either model quality or system efficiency in isolation, without jointly analyzing their trade-offs [31, 45]. Furthermore, there is limited understanding of how KV cache optimizations affect system metrics such as memory and bandwidth across various task types, input lengths, and model architectures. This slows the development of robust KV cache compression techniques and system-level optimization strategies.

In this paper, we present a comprehensive benchmark for KV cache optimization methods that jointly evaluates task quality and system performance under long-context workloads. We focus on representative inference-time techniques spanning three major paradigms: quantization, pruning, and merging. We employ widely used instruction-tuned models, Llama-3.1-8B-Instruct [17] and Mistral-7B-Instruct-v0.3 [21], and cover both task-level accuracy and system-level efficiency. Our benchmark is built on the LongBench suite [2], from which we select six datasets to evaluate task quality across four categories: multi-document QA, single-document QA, few-shot learning, and summarization. To assess system performance, we use three representative long-context datasets: NarrativeQA, GovReport, and Qasper. We measure key metrics, including time-to-first-token (TTFT), output throughput, and prefill KV cache memory footprint. This dual evaluation of quality and

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

system performance enables us to capture the trade-offs between accuracy preservation and inference efficiency across different KV cache compression strategies.

Our main contributions are as follows:

- We present a workload-aware benchmark that systematically evaluates KV cache compression methods across both task quality and system performance dimensions. This provides a holistic perspective on the methods and their associated trade-offs.
- We conduct a unified evaluation of representative methods from quantization (KIVI, TurboQuant), pruning and eviction (SnapKV) and merging (CaM) with consistent models, datasets, and experimental settings. This provides a fair comparison under realistic conditions.
- We study the tradeoffs between accuracy, throughput, latency, and memory footprint, and demonstrate that the compression ratio alone is not enough to evaluate the end-to-end performance of a KV compression approach. This insight has an immediate practical use and also guides future research on new KV cache optimizations.

The paper is organized as follows. Section 2 introduces the necessary background of LLM serving and KV cache optimizations, including a detailed introduction of the techniques we use in our benchmark. In Section 3, we provide a detailed account of the experimental setup, including workloads, setup and metrics. Based on that, we discuss the results in Section 4. From the results, we derive practical insights and lessons learned in Section 5. Finally, we discuss related work in Section 6 before concluding the paper in Section 7.

2 BACKGROUND

2.1 LLM Inference Serving

Large language model (LLM) inference serving refers to the system-level process of executing queries on pretrained models to generate outputs in real time. Fig. 1 depicts a typical serving pipeline, where an input prompt is first tokenized, i.e., converted from raw text into a sequence of discrete token IDs that can be processed by the model. The tokenized input is then processed during the prefill phase. In this phase, the entire input prompt is encoded, and the intermediate keys and values (KV) are computed and stored. This is followed by the *decoding phase*, where tokens are generated autoregressively, one step at a time [40]. Each generated token is subsequently detokenized, i.e., mapped back from token IDs to human-readable text, enabling streaming output to the user.

Prefill Phase. Let $X \in \mathbb{R}^{b \times l_{\text{prompt}} \times d}$ denote the input tensor, where b is the batch size, l_{prompt} is the prompt length, and d is the model hidden size. For simplicity, we omit the layer index. The key and value tensors are computed as

$$X_K = XW_K, \quad X_V = XW_V$$

where $W_K, W_V \in \mathbb{R}^{d \times d}$ are the projection matrices for keys and values, respectively. Once computed, X_K and X_V are stored in the KV cache to facilitate efficient decoding [33].

Decoding Phase. Let $t \in \mathbb{R}^{b \times 1 \times d}$ denote the current input token embedding. The corresponding key and value outputs are computed as $t_K = tW_K$ and $t_V = tW_V$, respectively. First, the KV cache is updated by appending the new entries:

$$X_K \leftarrow \text{Concat}(X_K, t_K), \quad X_V \leftarrow \text{Concat}(X_V, t_V).$$

Next, the attention output is computed as:

$$\begin{aligned} t_Q &= tW_Q, \\ A &= \text{Softmax}(t_Q X_K^\top), \\ t_O &= AX_V, \end{aligned}$$

where W_Q denotes the query projection matrix. For simplicity, we omit the attention output projection layer and other components of the full inference pipeline [33].

To avoid recomputing attention over all previous tokens at each decoding step, modern LLM systems maintain a *KV cache*. For a sequence of length N, each Transformer layer stores key and value tensors corresponding to all past tokens. During decoding, the query vector of the current token attends to the cached keys and values, reducing the computational complexity from $O(N^2)$ to $O(N)$ per step [32]. This reuse of previously computed representations is critical for achieving low-latency inference.

However, the KV cache can introduce a major system bottleneck. Its memory footprint grows linearly with both sequence length and the number of layers, often dominating GPU memory usage in long-context scenarios [28]. For large models and long inputs, KV cache storage and access become the primary limiting factors for throughput and scalability. This has motivated many works on KV cache optimization techniques, including quantization, pruning, and merging, which aim to reduce memory usage while preserving model accuracy.

2.2 Taxonomy of KV Cache Optimizations

To compress the KV cache size, there have been proposed four principal methods.

Quantization. The first principal method maintain all entries in the KV cache, but reduce the bit-length (precision) of the encoding of the stored keys and values. A common theme that runs among many KV quantization algorithm is mixed precision, where important tokens are kept at higher precision while heavily quantizing others. For example, ZipCache [17] and QAQ [7] use attention-derived properties to identify important tokens and store them at higher precision. In ZipCache, a channel-wise token quantization is proposed to reduce parameter overhead, and a normalized attention score guides which tokens to preserve. Similarly, KIVI [17] finds that keys have a few large-magnitude channels while values do not. It thus applies per-channel quantization for keys and per-token quantization for values, enabling 2-bit KV storage with almost no accuracy loss. KVQuant [18] also exploits KV structure. It quantizes keys before rotary embeddings and isolates outliers per vector by keeping them in higher precision.

Another class uses vector transforms or codebooks to ease quantization. For example, CommVQ [27] and PQCache [7] apply vector quantization. They split each KV vector into subvectors or additive code components and store compact indices instead of full float values. CommVQ [27] even designs its codebooks to commute with rotary embeddings, enabling as low as 1-bit precision with minimal loss. PolarQuant [41] and TurboQuant [46] apply random

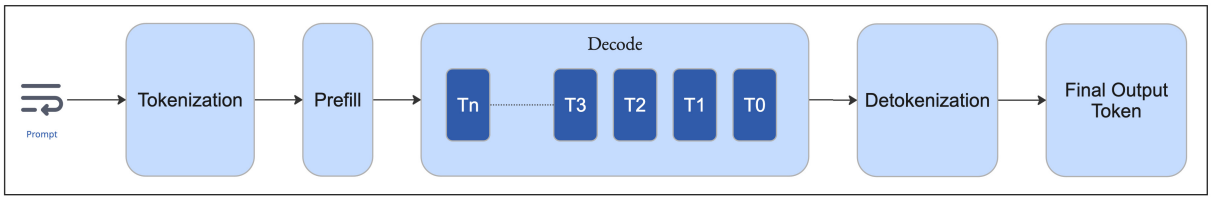

Figure 1: Prefill and decode during LLM inference. Token T0 is detokenized right after the prefill stage. After each token is generated in the decode step, the KV cache is updated. Detokenization happens after each decode step. Each token (T0, T1, T2, T3,....,Tn) is detokenized and output sequentially.

rotations or coordinate changes: PolarQuant [41] transforms KV vectors to polar coordinates after a random rotation, yielding tightly distributed angles that can be quantized without extra scaling parameters. TurboQuant [46] shows that a random rotation makes each coordinate follow a concentrated Beta distribution, so simple optimal scalar quantizers can nearly achieve the theoretical best distortion.

Quantization approaches trade memory for extra computation or design complexity. Mixed-precision schemes require computing token importance using attention scores or norms, and per-channel methods need special grouping of data. Transform-based methods incur the cost of rotating or projecting vectors at decode time, and product quantization uses a cookbook for encoding/decoding. However, the extra computational cost provides orders-of-magnitude memory savings with only small drops in accuracy. Aggressive quantization increases compression but risks output quality. Methods that preserve outliers or adapt precision mitigate this at the cost of extra overhead [18, 46].

Pruning. The second principal method approaches the problem by compressing the KV cache along the token or structural dimension, either by discarding less important tokens or by sparsifying their representations. The core idea is to rank tokens by some importance signal and keep only a subset. Similar to quantization, many pruning methods use attention-derived scores to decide which tokens to retain. For example, SAGE-KV [39] computes attention after prefill and retains only the top-k tokens for inference. This exploits the "attention sparsity" observation, usually only a few tokens have high influence. Similar approaches such as H2O [49] and SnapKV [29], aggregate past attention to pick important tokens. Another simple yet effective approach is recency. Sliding-window schemes such as StreamingLLM [42] keep only the most recent tokens plus a few early attention sinks to stabilize performance. These methods incur minimal overhead as no scoring of tokens is required, but can mistakenly drop older relevant context. Other signals such as L2 norm has been explored to guide eviction decisions [11].

Beyond token level pruning, structured pruning removes fixed groups of tokens at once. For example, PagedEviction operates on block "pages" of the KV cache [8]. It evicts full blocks aligned to the memory layout, resulting in less fragmentation and overhead, but coarser granularity. The trade-off here is that aggressive pruning saves memory, but has the risk of losing useful context. Attention-based pruning is precise but adds computation; static policies are simpler but less adaptable. Empirically, hybrid-strategies often works best. In all cases, the main cost is potential accuracy loss. Evicted tokens cannot be recovered, so if importance is misjudged, model outputs degrade.

Merging. The third principal method aim to compress the KV cache by combining redundant or highly similar tokens into shared representations, rather than discarding them outright. The core idea is to cluster redundant tokens and represent them with a shared state. For example, Adaptive KV [15] clusters KV tokens by similarity and merges each cluster into one representative. Bolya et al. [5] merges based on feature similarity on-the-fly during inference. Other work performs multi-level merging of tokens in a hierarchical manner, progressively compressing context while preserving high-level semantic structure [37].

Merging incurs extra computation cost for calculating pairwise similarity or clustering similar tokens but can retain much of the original information compared to pruning. It avoids outright deletion of content by combining tokens that have less importance with the retained tokens, preserving their relevance in a compressed form [15]. Few methods proposed by [34] and [24] progressively push past tokens into a fixed-size recurrent memory via a small updater network, so the KV cache acts like an RNN summary. This makes the KV cache quite complex and the merged states may lead to loss of fine grained details among the tokens. However, compared to pruning, merging yield smoother accuracy, since information is not fully lost.

Cross-Layer/Head Sharing and Low-Rank Methods. Quantization, pruning, and merging work within the token dimension of the KV cache. Another orthogonal approach compresses the KV cache across layers or attention heads, i.e., along the model's feature dimensions. They typically require model-level changes or retraining. A common theme is sharing KV projections. For example, Multi-Query Attention (MQA) [36] uses one shared key/value projection per layer instead of per head, whereas Grouped-Query Attention (GQA) [1] uses a few projections shared among groups of heads. MLKV [50] extends this idea to multiple layers by sharing the same KV projections across layers, achieving up to 6× more compression beyond MQA. Similarly, YOCO [38] introduces a two-stage transformer, which is a combination of a self-decoder and a cross-decoder, so that KV is computed globally once and reused by all layers. Other schemes such as [30] and KVSharer [43] selectively merge KV states across layers based on similarity. These methods drastically cut KV size and bandwidth by design, often with some

pre-training or architecture redesign. The downside is complexity. They require changes to the model architecture or training.

Another direction targets the feature dimension using low-rank approximations. Methods such as Palu [6] factorize attention into low-rank components and cache compressed representations, reconstructing full KV states on demand. LoRC [47] applies layer-wise low-rank projections to compress KV weights without retraining. By exploiting redundancy in hidden dimensions, these methods shrink KV storage, though they introduce approximation error and some extra compute for compression and reconstruction.

Overall, these methods structurally reduce KV size and are complementary to token-level techniques such as pruning or quantization. Their main drawback is integration complexity and potential accuracy loss, while their key advantage is a consistent reduction in memory usage and bandwidth, independent of input length.

2.3 Benchmarked Techniques

Selection rationale for this study: In our benchmarking study, we focus on drop-in replaceable KV cache compression techniques that do not require deep changes to the model architecture or even retraining. From the practical perspective of data management techniques, it is desirable that a data system architect or administrator can easily integrate the techniques into existing infrastructure. Following this rationale, we selected representative techniques from the three categories of quantization (KIVI, Turboquant), pruning (SnapKV), and merging (CaM). We discuss the details of these techniques in the following.

KIVI. The mathematical core of KIVI [33] is built upon the observation that the key cache (K) and value cache (V) exhibit different outlier structures. In the key cache, outliers are concentrated in specific channels, meaning certain dimensions across all tokens have much higher magnitudes. Conversely, the value cache contains outliers that are token-specific, where certain entire tokens have higher magnitudes across all channels. KIVI uses an asymmetric quantization scheme to map these high-precision floating-point values into a discrete B-bit integer space.

For any given tensor X, the quantization process is defined by finding a scaling factor s_X and a zero-point z_X. The quantization function $Q(X)$ maps the input to the nearest integer in the range $[0, 2^B - 1]$ using the following formulas:

$$z_X = \min(X),$$
$$s_X = \frac{\max(X) - \min(X)}{2^B - 1},$$
$$Q(X) = \left\lfloor \frac{X - z_X}{s_X} \right\rceil$$

KIVI adopts different quantization axes for keys and values to align with their distinct outlier patterns. It applies *per-channel quantization* to keys (column-wise) so that large values concentrated in certain dimensions do not distort the scaling for the entire tensor. For values, it applies *per-token quantization* (row-wise) so that tokens with unusually large magnitudes are handled locally, preventing them from degrading the precision of other tokens.

During the inference phase, the B-bit quantization integers must be converted back to the original precision for the model to perform the attention operation. This process can be expressed as

$$X' = Q(X) \cdot s_X + z_X$$

where $\lfloor \cdot \rceil$ denotes the rounding operator to the nearest integer.

To further stabilize performance, KIVI employs a streaming strategy involving a residual cache. New tokens are kept in full precision because recent tokens often contribute most significantly to the attention output, and their distributions have not yet stabilized. Once the residual cache reaches a pre-defined threshold, the oldest tokens in the residual are quantized using the per-channel/per-token logic and appended to the compressed 2-bit KV cache. This hybrid precision approach ensures that the "local" context remains highly accurate while the "distant" context is efficiently compressed.

TurboQuant. TurboQuant [46] approaches transforms KV cache vectors into a distribution that is amendable to efficient quantization while preserving attention-relevant inner products. Given an input vector $x \in \mathbb{R}^d$, TurboQuant first applies a randomized orthogonal transformation using a matrix $Q \in \mathbb{R}^{d \times d}$, producing $y = Qx$. This transformation preserves the Euclidean norm while redistributing the vector's energy uniformly across dimensions. In high-dimensional settings, the coordinates y_i approximate a Gaussian distribution, which enables the use of simple, data-independent scalar quantizers.

For a given vector y, TurboQuant performs coordinate-wise b-bit quantization. Let $Q_s(\cdot)$ denote the scalar quantizer; the quantized vector is defined as

$$\hat{y} = [Q_s(y_1), Q_s(y_2), \ldots, Q_s(y_d)]^\top,$$

with total distortion

$$\mathrm{MSE}(y, \hat{y}) = \sum_{i=1}^{d} \mathbb{E}[(y_i - \hat{y}_i)^2].$$

By leveraging the Gaussian-like distribution induced by the rotation, this approach achieves near-optimal rate–distortion performance using simple scalar quantization.

To mitigate the contraction bias introduced by MSE-optimal quantization, TurboQuant incorporates a residual correction mechanism. After quantization, the residual vector $r = y - \hat{y}$ is computed and encoded using a 1-bit Quantized Johnson–Lindenstrauss (QJL) transform, storing only the sign information:

$$s = \mathrm{sign}(r).$$

During inference, the quantized vector $\hat{y}$ and residual signs s are combined to produce an unbiased estimate of inner products, ensuring that attention scores remain accurate despite aggressive compression.

This design enables TurboQuant to compress KV cache vectors to very low bit rates while maintaining high fidelity in both reconstruction error and attention computation.

SnapKV. SnapKV [29] leverages the intrinsic sparsity and consistency of attention patterns in LLMs to compress the KV cache. The method identifies salient historical tokens by observing the attention distribution at the end of a prompt and evicting non-essential entries to maintain a constant-sized cache. This process is governed

by a heuristic selection mechanism that utilizes pooled attention scores to determine which KV pairs are critical for future token generation.

For a prompt of length N, SnapKV computes attention weights using an observation window of queries Q_{obs} attending over prefix keys K_{prefix}:

$$A = \text{Softmax}\left(\frac{Q_{\text{obs}} K_{\text{prefix}}^{\top}}{\sqrt{d}}\right).$$

To estimate token importance, the attention weights are aggregated across the observation window and smoothed using a 1D average pooling operation:

$$S = \text{AvgPool}(A, \text{kernel} = s).$$

The most salient tokens are then selected via a Top-k operation under a fixed cache budget C:

$$I = \text{Top-}k(S, k = C - L_{\text{recent}}).$$

The final compressed KV cache consists of the union of selected salient tokens and a recent token window:

$$KV_{\text{compressed}} = \{KV_i \mid i \in I\} \cup \{KV_j \mid j > N - L_{\text{recent}}\}.$$

By keeping both C and L_{recent} fixed, SnapKV preserves long-range dependencies through salient token retention while maintaining local coherence via recent tokens, enabling efficient long-context inference without retraining.

CaM. Cache Merging (CaM) departs from conventional KV cache pruning methods by avoiding hard eviction and instead redistributing the contribution of removed tokens into retained ones. The key idea is to preserve the attention output by merging values of evicted tokens into nearby tokens, thereby reducing the bias introduced by removing low-probability but non-negligible contributions.

In a standard attention layer, the output for a query is given by

$$O = \sum_{i=1}^{n} \alpha_i V_i,$$

where α_i are attention weights and V_i are value vectors. When a token k is removed, CaM merges its contribution into a retained token j by updating

$$V_j' = V_j + \frac{\alpha_k}{\alpha_j} V_k,$$

which preserves the attention output since

$$\alpha_j V_j' = \alpha_j V_j + \alpha_k V_k.$$

This formulation shows that, given exact attention ratios, merging can be lossless.

Since future attention weights are unknown during inference, CaM approximates this process using *even merging*, where an evicted token V_i is distributed across a local window of m retained tokens:

$$V_j' = V_j + \frac{1}{m} V_i.$$

This approximation assumes locally similar attention magnitudes and reduces variance compared to single-point merging.

To further improve robustness, CaM employs an adaptive merging strategy based on cumulative attention scores $\bar{A}_i$, which provide a stable estimate of token importance. The decision to merge a token is modeled as

$$M_i \sim \text{Bernoulli}\left(\text{clamp}\left(\frac{\bar{A}_i}{\frac{1}{m}\sum_{j \in \text{window}} \bar{A}_j}, 0, 1\right)\right).$$

Tokens are merged when their relative importance is sufficiently high; otherwise, they are evicted.

Overall, CaM compresses the KV cache by converting discrete token removal into a continuous redistribution of value representations. This design preserves attention outputs more faithfully than standard pruning while maintaining a bounded memory footprint.

Dataset	Min-Max Tokens	Workload
HotpotQA	2K-8K	Multi-Doc QA
2WikiMQA	2K-8K	Multi-Doc QA
Qasper	1K-20K	Single-Doc QA
MultiFieldQA_en	1K-16K	Single-Doc QA
TriviaQA	2K-8K	Few-shot Learning
Multi-news	4K-32K	Summarization
NarrativeQA	8K-64K	Single-Doc QA
GovReport	2K-64K	Summarization

Table 1: Datasets, token ranges, and workload types.

3 EXPERIMENTAL SETUP

Workloads. We evaluate KV cache compression methods using the LongBench benchmark [2], which consists of many long-context tasks. Table 1 shows the datasets we use along with the minimum and maximum context length test examples it consist of and the workload type. We consider four varying categories: (1) multi-document QA, which needs to extract and combine information from several documents to obtain the answer, (2) single-document QA, which tests the long context understanding ability with longer documents, (3) few-shot learning, which is a practical setting requiring long-context understanding over provided examples, and (4) summarization, which requires a global understanding of the whole context in this work [45]. For task quality evaluation, we use six representative datasets: HotpotQA and 2WikiMQA for multi-document QA, Qasper and MultiFieldQA_en for single-document QA, TriviaQA for few-shot learning, and MultiNews for summarization. These datasets span a wide range of context lengths, reasoning complexity, and dependency patterns, from multi-hop retrieval across documents to long-form generation tasks. For system efficiency evaluation, we focus on NarrativeQA, Qasper, and GovReport, as they contain substantially longer contexts, making them better suited to capturing realistic KV cache behavior and more representative system-level performance under long-context workloads. This enables us to systematically study the impact of KV cache compression on task quality and system efficiency for various long context tasks.

Compared Methods. We select a small but representative set of KV cache compression methods to capture the core design trade-offs relevant for long-context serving systems. Rather than exhaustively evaluating all prior work, we focus on widely adopted, training-free

methods that can be directly applied at inference time and reflect distinct system-level behaviors.

For quantization, we evaluate KIVI [33] and TurboQuant [46], which represent two complementary techniques. KIVI is a data-dependent approach that explicitly models the outlier structure in keys and values via asymmetric quantization, achieving consistent accuracy despite aggressive compression. On the other hand, TurboQuant is a data-oblivious, rotation-based approach with strong theoretical guarantees, allowing efficient low-bit compression that behaves distinctly under varying workloads. For pruning and eviction, we include SnapKV [29], a state-of-the-art training-free approach that leverages consistent attention patterns to retain salient tokens under a fixed cache budget. This makes it representative of practical token-level sparsification strategies used in long-context inference. For merging-based compression, we evaluate CaM [48], which departs from hard eviction by redistributing evicted token contributions to preserve attention outputs. This provides a fundamentally different trade-off, prioritizing output integrity over strict sparsity.

These KV cache compression methods can be easily deployed in real systems. It enables us to systematically analyze how different KV cache optimization strategies impact both task quality and end-to-end system performance under diverse workloads.

LLMs. We conduct experiments on two widely used instruction-tuned models: Llama-3.1-8B-Instruct [16] and Mistral-7B-Instruct-v0.3 [21]. Both models use Grouped-Query Attention (GQA). The differentiator is that Mistral-7B specifically utilizes Sliding Window Attention (SWA) and a rolling buffer cache for extreme efficiency, whereas Llama-3.1-8B focuses on maximizing reasoning performance, utilizing a much larger 128k context window compared to 32k in Mistral-7B.

These architectural differences lead to different attention behaviors and KV cache usage patterns, providing a diversity of models for evaluating compression strategies. Using both models allows us to determine whether the observed performance trends are consistent across LLM families or sensitive to model-specific characteristics.

Setup. All methods were evaluated using the same models and datasets. Models were loaded in FP16 with FlashAttention2 [9] enabled, and prompts were formatted using the corresponding chat template for each instruction-tuned model. We used the LongBench prompt templates and dataset-specific generation budgets [2].

For KIVI [33], we use the official KIVI-style asymmetric KV-cache quantization implementation integrated into the Llama and Mistral attention modules. We evaluate both 2-bit and 4-bit KV-cache variants. In both cases, keys and values are quantized to the same bit width, with group size for group-wise quantization and residual length for preserving the latest tokens in full precision as 32. The KIVI's FP16 model where both keys and values are stored in full precision without any quantization is used as the full-cache baseline.

For TurboQuant [46], we used the Transformers 4.45-compatible TurboQuant cache implementation by Omar Hory [19]. We evaluated 3-bit and 4-bit KV-cache quantization. The 3-bit configuration used outlier-aware quantization: for a head dimension of 128, 32 outlier channels were stored at 4 bits, giving an effective precision of 3.25 bits per value. The 4-bit setting used uniform 4-bit quantization without outlier channels. The reported runs used the unpacked cache representation, so the reported memory usage may be slightly higher than with additional bit-packing optimizations.

For SnapKV, we used the SnapKVPress implementation from KVPress [10]. The cache was compressed just after prefill using SnapKV's attention-based token selection [29]. We set the compression ratio to 0.75. In SnapKV, window size denotes the number of most recent query tokens used to estimate the importance of earlier KV entries, while kernel size denotes the width of the average-pooling filter used to smooth the resulting attention-based importance scores. We set the window size to 32 and the kernel size to 7.

For CaM [48], we evaluated the CAMPress wrapper with SnapKV as the base compressor provided by KVPress [10]. We used CAMPress with a compression interval of 32, which means that CaM applies one merge-and-prune compression pass after every 32 decoding steps. After each compression pass, the KV cache is reduced to a pre-defined number of retained tokens per layer, known as the target cache size. We set it to 1024. The hidden-state buffer size is set to 64, which provides longer recent decoding context available for scoring and compressing. A merge budget of 32 helps redistribute the token value across up to 32 subsequent tokens instead of being simply dropped. The underlying SnapKV base press uses a window size of 16 and a kernel size of 7. We use a smaller SnapKV window inside CaM because CaM performs periodic decoding-time compression using recent buffered hidden states and cumulative attention, so a shorter observation window emphasizes the most recent local decoding context and keeps the base scorer aligned with interval-based compression. Thus, CaM performs periodic cache merging during generation rather than only applying a one-shot prefill compression.

We conduct our experiments on a NVIDIA A100 GPU with 40GB of memory. As a result, we limit our evaluation to models in the 7-8B parameter range, which represents a practical balance between model capability and feasible long-context inference under KV cache compression. The main trends we observe are expected to generalize to larger models and distributed settings. KV cache size and memory bandwidth costs grow with model size and context length, so KV compression carries the same importance for larger models [33]. The workload-dependent patterns we observe are driven by task structure and are expected to persist. However, exact memory usage, throughput, and latency will depend on system-level optimizations, so quantitative gains may differ in larger deployments.

Metrics. We evaluate KV cache compression methods along two dimensions: *task quality* and *system efficiency*.

We report both per-task performance and category-level averages to capture method behavior across heterogeneous workloads. In doing so, we rely on accuracy metrics provided by LongBench for each dataset [2]. For the question answering tasks HotpotQA, 2WikiMQA, Qasper, and MultiFieldQA_en, we use the F1 score, which measures the overlap between predicted and ground-truth answers. For TriviaQA, we use the exact match (EM), which checks whether the model's predicted answer matches the ground-truth answer exactly, after applying basic normalization. For summarization datasets such as MultiNews, we report ROUGE scores, which

	Models	Multi-Doc QA		Single-Doc QA		Few-Shot Learning	Summarization
		HotpotQA	2WikiMQA	Qasper	MultifieldQA_en	Trivia_QA	Multi-news
Llama-3.1-8b-Instruct	All KV	55.72	44.27	45.76	53.92	91.65	**27.15**
	KIVI2	54.42	42.94	44.07	54.98	**92.56**	*26.99*
	KIVI4	55.99	46.63	45.42	54.74	91.66	26.95
	TurboQuant3	53.19	41.57	46.60	51.46	84.84	16.39*
	TurboQuant4	56.10	45.07	46.03	53.31	90.13	16.16*
	SnapKV-0.75	*59.2*	*49.41*	*46.37*	*56.06*	*92.04*	23.04
	CaM	**59.8**	**51.33**	**47.32**	**56.42**	91.71	17.56
Mistral-7b-Instruct-v0.3	All KV	**51.54**	36.16	38.27	50.10	**88.59**	26.42
	KIVI2	48.46	38.34	39.89	**54.12**	87.82	*27.04*
	KIVI4	48.98	**39.11**	**41.10**	*53.56*	*88.54*	**27.68**
	Turboquant3	*49.45*	34.31	37.16	50.21	88.05	15.86*
	Turboquant4	49.43	33.94	40.39	50.65	88.09	16.01*
	SnapKV-0.75	48.43	*38.46*	36.44	51.23	86.31	24.25
	CaM	48.57	37.59	*40.55*	52.4	85.37	16.74

Table 2: Accuracy results on LongBench tasks for Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3 under different KV compression methods. Bold and italic values denote the best and second-best performance per column, highlighting the trade-offs between compression efficiency and task-specific accuracy. *Note:* *Summarization task using TurboQuant was only ran once as the run takes 3-4 days to complete.

evaluate the quality of generated summaries based on n-gram overlap with reference summaries. These metrics reflect the model's ability to preserve semantic correctness and coherence under KV cache compression. We report both per-dataset performance and category-level averages.

We focus our system-level evaluation on three representative datasets: NarrativeQA, GovReport, and Qasper, as they have longer context lengths. We measure efficiency metrics such as time-to-first-token (TTFT), output throughput, and prefill KV memory.

Time-to-First-Token measures the time elapsed from when a user submits a prompt to when the model generates its very first output token. It captures the total latency of the prefill stage, including KV cache construction and initial model overhead. It is heavily influenced by input prompt length, KV cache compression strategies, and includes the computation time required to generate the very first output token. Lower TTFT indicates faster perceived responsiveness in real-time applications such as chatbots, voice AI, and interactive streaming because it reduces the initial wait time before the user sees the first piece of output.

Output Throughput is measured as the average number of tokens generated per second during inference. This metric reflects the efficiency of the decoding stage and is directly impacted by the KV cache compression used. Higher throughput indicates better system performance.

Prefill KV memory measures how much VRAM is consumed to hold the context during that initial KV processing stage, which is a major factor in overall peak memory. This metric captures the effectiveness of each compression method in reducing memory usage. Lower KV cache memory enables longer context lengths and improved scalability under given hardware constraints.

4 RESULTS

Our results are organized into two sections: *Task Quality* and *System Efficiency*.

4.1 Task Quality

We first discuss general observations and trends across all tasks before discussing the detailed results for each workload in separate paragraphs. Table 2 shows all accuracy results.

General Observations. We observe clear differences between KV-cache compression paradigms across all workloads and models. Overall, moderate quantization methods, especially KIVI4, achieve the most stable accuracy across workloads, while pruning using SnapKV and merging using CaM often yield higher peak accuracy across the QA workload, but the accuracy is variable across other tasks. This can be attributed to the fact that, despite aggressive compression, KIVI handles key/value outliers in an efficient manner. Aggressive quantization, such as in TurboQuant, leads to a drop in quality, especially for generation-heavy workloads such as summarization. This indicates that, although rotation-based quantization is theoretically efficient, it introduces reconstruction errors that negatively affect tasks requiring global context understanding.

Multi-Document QA. For Multi-Document QA, CaM and SnapKV achieve good performance across both models. On Llama-3.1-8B, both CaM and SnapKV even outperform the All-KV baseline. This can be explained by the nature of the Multi-Doc QA task, which requires the retrieval and aggregation of sparse but highly relevant information across documents. Pruning preserves important tokens based on attention scores, ensuring that important cross-document

			Time to first token (in ms)						
		Context	**All KV**	**KIVI2**	**KIVI4**	**TurboQuant3**	**TurboQuant4**	**SnapKV-0.75**	**CaM**
Llama-3.1-8b-Instruct	NarrativeQA	4-8K	724.89	782.14	777.80	882.29	812.66	763.85	744.18
		8K+	3873.34	4048.71	4039.81	4397.15	4273.93	3982.49	3922.54
	GovReport	0-4K	287.42	334.59	332.31	-	-	310.93	306.06
		4-8K	552.74	605.68	603.34	-	-	581.63	572.97
		8K+	1402.18	1485.98	1482.85	-	-	1449.56	1430.83
	Qasper	0-4K	289.62	333.31	330.20	405.08	324.25	310.62	306.25
		4-8K	496.82	544.73	541.85	637.29	558.41	522.12	514.27
		8K+	1124.66	1194.20	1190.24	1351.43	1260.67	1163.29	1148.08
Mistral-7b-Instruct-v0.3	NarrativeQA	8K+	4932.28	5013.10	5042.05	4757.37	4172.87	4957.70	4905.75
	GovReport	0-4K	308.67	346.50	348.43	-	-	317.84	314.13
		4-8K	556.53	592.91	595.35	-	-	563.84	555.55
		8K+	1504.87	1547.04	1556.57	-	-	1508.24	1486.31
	Qasper	0-4K	311.67	346.38	346.39	497.89	338.68	319.88	316.52
		4-8K	532.64	567.66	568.23	778.63	582.58	541.21	534.14
		8K+	1138.46	1176.59	1182.53	1513.77	1249.65	1143.80	1133.55

Table 3: Time to first token of different KV caching methods across 3 datasets (NarrativeQA, GovReport, Qasper) and varying context length ranges for Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3.
Note: **We do not present results for Gov_Report using TurboQuant because it takes up to 3 days for one single run.**

context is retained, and merging redistributes information instead of discarding it, which reduces information loss compared to pruning. Heavy quantization with KIVI2 and TurboQuant3 shows slight performance degradation. This is likely because precision loss in attention scores affects multi-hop reasoning, where small numerical differences can propagate across reasoning chains.

Single-Document QA. In Single-Document QA tasks, results are more balanced, especially with the Llama model. Similar to Multi-Document QA, CaM achieves the best performance across all the compression methods with Llama, while KIVI4 performs consistently well across both models.

The good performance of CaM indicates that merging benefits tasks that require holistic document understanding, as it preserves broader context. KIVI4 achieves the best Qasper accuracy score and near-best MultifieldQA scores for Mistral-7b-Instruct-v0.3. The relatively weaker performance of SnapKV on Mistral suggests that aggressive token selection may remove context needed for detailed comprehension, especially in long single documents.

Few-Shot Learning. In few-shot learning, quantization methods perform surprisingly well, with KIVI2 achieving the highest score. This suggests that few-shot learning tasks are less sensitive to fine-grained KV precision, as they rely more on pattern recognition from examples rather than on exact long-range dependencies. Few-shot learning workload is robust to compression as long as recent tokens, which here are few-shot examples, are preserved. KIVI explicitly ensures this via its residual cache mechanism.

In contrast, aggressive TurboQuant compression using the Llama-3.1-8b-Instruct model shows a noticeable degradation in quality of almost 7%. This indicates that aggressive low-bit quantization may still harm the pattern extraction required for few-shot learning tasks.

SnapKV and CaM remain competitive but do not outperform KIVI. This can likely be attributed to the fact that token selection and eviction or merging provide less benefit when most few-shot examples are already highly relevant.

Summarization. Summarization workload exhibits the largest largest performance drop across compression methods, especially for TurboQuant and CaM.

Quantization using KIVI remains closest to baseline All-KV quality performance, mainly because KIVI retains all tokens, only reducing precision rather than changing the token structure. TurboQuant underperforms in summarization because its uniform quantization introduces small errors across all tokens, which accumulate and disrupt the global context coherence required for high-quality summaries. SnapKV shows a moderate degradation in quality of approximately 15% and 8% using Llama-3.1-8b-Instruct and Mistral-7b-Instruct-v0.3 respectively. This reflects that pruning might remove information that may still be important for coherent summaries. Whereas merging, especially using CaM, introduces approximation errors when combining tokens, which results in a quality degradation of approximately 35-36% using both models.

Thus, summarization is most sensitive to structural modifications of the KV cache, confirming that compression methods that preserve full context are preferable.

4.2 System Efficiency

Time to first token. Table 3 shows TTFT over different context length buckets for all KV cache compression methods. In terms of

		Context	Min_tokens	Max_Tokens	Compression Rate					
					KIVI2	KIVI4	TurboQuant3	TurboQuant4	SnapKV-0.75	CaM
Llama-3.1-8b-Instruct	NarrativeQA	4-8K	7964	7964	5.25	3.17	4.27	3.76	4.00	1.00
		8K+	8962	65271	5.32	3.19	4.27	3.76	4.00	1.00
	GovReport	0-4K	2020	3919	5.16	3.15	-	-	4.00	3.09
		4-8K	4138	7979	5.24	3.17	-	-	4.00	6.04
		8K+	8118	51393	5.29	3.18	-	-	4.00	13.82
	Qasper	0-4K	1847	3934	5.16	3.15	4.27	3.76	4.00	1.26
		4-8K	4044	7874	5.23	3.17	4.27	3.76	4.00	1.17
		8K+	8027	21111	5.28	3.18	4.27	3.76	4.00	1.22
Mistral-7b-Instruct-v0.3	NarrativeQA	8K+	9570	81496	5.31	3.19	5.02	4.43	4.00	1.00
	GovReport	0-4K	2204	3988	5.16	3.15	-	-	4.00	3.21
		4-8K	4140	7989	5.32	3.17	-	-	4.00	5.86
		8K+	8071	58334	5.30	3.18	-	-	4.00	14.20
	Qasper	0-4K	2091	3986	5.17	3.15	4.27	3.76	4.00	1.09
		4-8K	4002	7841	5.24	3.17	4.27	3.76	4.00	1.07
		8K+	8024	24121	5.28	3.18	4.27	3.76	4.00	1.13

Table 4: Compression rates of different KV caching methods across 3 datasets (NarrativeQA, GovReport, Qasper) and varying context length ranges for Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3. We report compression ratios alongside minimum and maximum token lengths for each method, illustrating how compression efficiency varies with sequence length and task. *Note:* We do not present results for Gov_Report using TurboQuant because it takes upto 3 days for one single run.

TTFT, we observe only moderate differences across KV compression techniques, which are largely consistent across both models, as illustrated in Fig. 2. Overall, most methods remain close to the All-KV baseline, confirming that KV cache optimizations have limited impact on prefill latency. KIVI and SnapKV show slightly increased TTFT compared to All-KV, reflecting the additional overhead introduced by quantization and attention-based token selection during the prefill phase. CaM performs similarly, with only minor deviations, as its merging operations are applied periodically and do not heavily impact the initial prefill stage.

In contrast, TurboQuant exhibits the highest TTFT overhead among all methods, particularly for longer contexts such as NarrativeQA and Qasper. This is expected, as TurboQuant applies computationally expensive transformations, including random rotations and residual corrections, which increase the cost of KV cache construction before the first token is generated. Additionally, we observe that TTFT increases with context length across all methods, highlighting that prompt processing remains the dominant factor in prefill latency.

Overall, these results indicate that KV cache compression techniques introduce only limited additional latency for the first token, with the exception of TurboQuant, whose more complex quantization pipeline leads to noticeable overhead. This suggests that most compression methods can be safely applied in latency-sensitive applications without significantly affecting perceived responsiveness, aligning with the observation that TTFT is primarily dominated by prompt encoding rather than KV cache operations.

Throughput. In terms of throughput, we see large differences between the compression techniques that are consistent across both models as can be seen in Fig. 3. SnapKV achieves the best throughput, on par with All-KV on Llama and even surpassing All-KV on Mistral. KIVI and CaM are in the middle, suffering of only modest throughput degradation. The most overhead is introduced in TurboQuant, leading to severe throughput drop.

These results are expected, as SnapKV comes with very low overhead and effectively reduces the *number* of tokens in the KV cache instead of just compressing their representation. CaM also reduces the number of tokens, but induces higher overheads as it redistributes the contributions of evicted tokens. The quantization techniques KIVI and TurboQuant do not reduce the number of tokens in the KV cache, but only compress them. Here, the compression techniques in TurboQuant are more aggressive, leading to higher overheads that have significant impact on throughput.

Compression rate on KV memory. Table 4 shows the compression rates of the various KV cache optimizations, broken down across workloads and context-lengths within each workload. This allows for a detailed look into the *effectiveness* of the techniques in achieving their primary goal: reducing the size of the KV cache in GPU memory. We make two major observations, one on stability of compression rates and one on the maximally achievable compression.

Stability: KIVI and SnapKV achieve the most stable and predictable compression rates. There are no major fluctuations across the different workloads and context lengths. TurboQuant is also

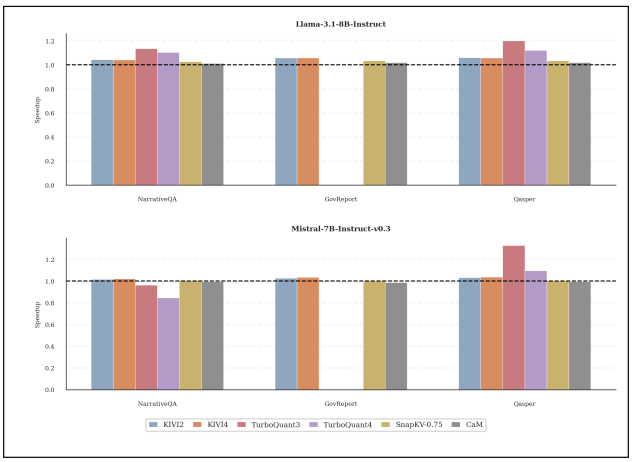

Figure 2: Relative time for TTFT (normalized to All KV) of KV compression methods across models and tasks.
***Note:* We do not present results for Gov_Report using TurboQuant because it takes upto 3 days for one single run.**

relatively stable, but shows an outlier with the NarrativeQA workload where the compression rate is slightly higher This can be attributed to its rotation-based quantization scheme, which depends on the statistical distribution of KV vectors; longer and more diverse contexts, such as in NarrativeQA, tend to produce more uniformly distributed representations, enabling slightly more effective compression [12]. For CaM, we see a completely different picture: Compression lies between extreme compression of 14.2 times (GovReport on 8K+ context length) and no compression at all (compression rate of 1.0 for NarrativeQA). This variability stems from CaM's adaptive, attention-driven merging strategy, which dynamically decides whether tokens should be merged or retained based on their estimated importance. In workloads like GovReport, where redundancy is higher and many tokens can be safely merged, CaM achieves very high compression. However, in tasks such as NarrativeQA, where a larger portion of the context remains relevant for downstream generation, fewer tokens qualify for merging, resulting in minimal compression. From a data management perspective, such unpredcitability of compression effectiveness may make it challenge to operate CaM in real-world deployments.

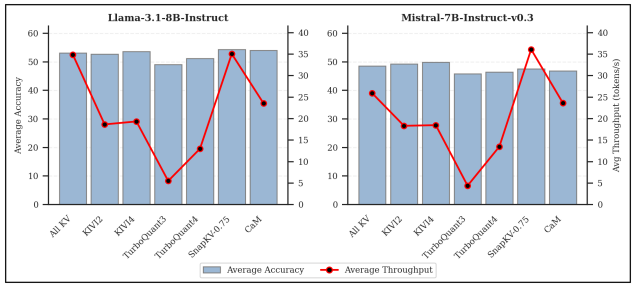

Figure 3: Average accuracy and throughput across different KV compression methods

Effectiveness: Among the quantization-based compression techniques, KIVI2 shows the highest effectiveness with an average compression rate between 5.16 and 5.32. The compression rates of TurboQuant3 and Turboquant4 are between those of KIVI2 and KIVI4. SnapKV is also exactly in the middle between both KIVI settings. This shows that, by using the quantization width as a parameter, it is possible to carefully tune the approaches to achieve a desired compression rate. This is good news for practical data management, providing an effective tuning knob that can be adapted to given hardware constraints and workload characteristics. If extreme compression is needed under certain long-context workloads, CaM can be a good choice, as it achieved the highest compression rates under the long-context GovReports tasks (up to 14.2).

5 LESSONS LEARNED

We present lessons learned and relate them to conventional data management wisdom, comparing our lessons to insights from recent data management papers on related topics where appropriate.

KV cache compression can preserve high accuracy. All-KV was not the consistent winner in terms of accuracy across the various workloads in our benchmark. Instead, under most workloads, one of the compression techniques achieved even better results. This shows that it is not important to retain *all* information in the KV cache, but to identify the *task-relevant* information. Summarizing across all workloads, we show that the decision to use a KV cache compression technique does not necessarily mean trading accuracy for memory efficiency. This is different from conventional data management wisdom about lossy compression, where information loss is considered an undesirable but necessary cost to achieve a high compression rate [4, 22].

Compression overhead can be amortized by reducing KV cache size. In terms of throughput, we observe that some compression techniques indeed cause a certain throughput degradation, but not all. By reducing the number of tokens in the KV cache, SnapKV yielded even higher throughput than the All-KV baseline. Again, this is different from conventional data management wisdom, where compression and decompression overheads are considered an additional cost that one has to pay to achieve a reduction in data size. One could compare our results to Isenko et al. [20], who showed that data compression in machine learning preprocessing pipelines can in some cases be amortized by overcoming communication bottlenecks.

Stable compression rates are preferable in real-world deployments. While CaM achieves extremely high compression rates on some long-context workloads, it fails to be effective on shorter context workloads. This discrepancy has consequences for practicability. It may be extremely hard to predict under which circumstances an LLM may be used, especially if it accepts prompts from users. Indeed, it is one of the strengths of LLMs that they are general-purpose across a variety of different tasks. In such settings, from a data systems management perspective, having more predictable performance is key. Hence, we recommend using quantization or pruning methods except for scenarios with repeated and uniform tasks, such as using LLM-based summarization for similar kinds of documents.

Latency in terms of time-to-first-token is merely affected by KV cache optimizations. While there are overheads involved in decompressing KV cache (especially under quantization), these do not significantly affect the perceived latency for the user. This is because KV-cache overheads are insignificant when compared to decoding and detokenization phases of LLM inference. As a result, KV cache optimizations can safely be applied in latency-sensitive settings such as user-facing interactions.

KV cache optimization trends are similar across LLM models. We find that the relative performance of the various KV cache compression methods is consistent for both Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3. In particular, KIVI provides consistent accuracy, SnapKV consistently improves throughput, and TurboQuant incurs additional overheads across both models. While absolute performance scores vary due to architectural differences such as sliding-window attention in Mistral, the qualitative trends are consistent. From a data management perspective, this implies that system-level insights from KV cache optimization are transferrable across model families, enabling generalizable optimization strategies without the need for model-specific tuning.

Accuracy stability is different in different workloads. We observe that the impact of KV cache compression on accuracy strongly depends on the task type. Tasks such as few-shot learning are relatively robust to compression, as they rely primarily on recent tokens and local patterns, whereas summarization is highly sensitive to any form of KV modification due to its reliance on global context. Multi-document QA lies in between, benefiting from selective pruning or merging that preserves important information while removing redundancy. From a data management perspective, this implies that workload characteristics such as context redundancy, dependency structure, and sensitivity to global information determine the effectiveness of compression. Consequently, KV cache optimization should be treated as a workload-dependent decision rather than a uniform system configuration.

Quantization provides robustness across unknown workloads, but not all methods behave equally. We observe that KIVI is largely insensitive to workload type, maintaining stable accuracy across QA, few-shot, and summarization tasks. This robustness is due to its asymmetric, data-aware design that takes into account explicitly different outlier structures in keys and values and preserves recent tokens in higher precision with a residual cache. Thus, KIVI preserves both local and global context information even under aggressive compression, making it a reliable default choice when workload characteristics are unknown.

In contrast, TurboQuant is not as robust, despite also being a quantization-based method. Its data-oblivious design relying on random rotations and uniform scalar quantization, leads to small but systematic reconstruction errors on all tokens. While these may be acceptable for local or pattern-based tasks, they accumulate in workloads requiring an understanding of global context, such as summarization, resulting in significant quality degradation. Moreover, TurboQuant introduces higher computational overhead due to its transformation and residual correction steps, which adversely affect system performance. From a data management perspective, this implies that not only the compression paradigm, but also the underlying design principles (data-aware vs. data-oblivious) dictate robustness and suitability for deployment.

6 RELATED WORK

Recent work has increasingly framed KV cache optimization as a system-level problem in LLM inference, closely aligned with data management concerns such as memory efficiency, scheduling, and resource utilization. Surveys such as Towards Efficient Large Language Model Serving [23] emphasize that KV cache optimization spans multiple system dimensions, including execution scheduling, memory placement, and representation design, highlighting that KV cache is a central bottleneck in modern LLM serving systems . Similarly, Li et al. [26] categorize KV cache management techniques into token-level, model-level, and system-level optimizations, reflecting the need for holistic approaches that jointly consider computation and memory trade-offs . Xu et al. [31] further show that no single optimization strategy dominates across workloads, and that adaptive, workload-dependent KV cache management is essential for scalable inference .

From a data management perspective, prior work on ML pipelines has demonstrated that performance bottlenecks are often driven by memory movement and data representation rather than pure computation, and that compression can improve system efficiency despite introducing approximation errors [4, 20, 22]. However, these studies typically assume a fixed trade-off between compression and accuracy, and evaluate optimizations at isolated stages of the pipeline. Earlier works such as CLA [14] considered lossless compression only, which is hard to achieve in KV cache as there is little redundancy in KV cache tensors that could be exploited.

In contrast, our work positions KV cache compression as a core data management problem within LLM inference pipelines, and provides a unified evaluation across accuracy, latency, throughput, and memory under realistic long-context workloads. Unlike prior survey and systems work, which primarily categorize techniques or analyze them in isolation, we empirically show that the effectiveness of KV cache optimization is highly workload-dependent and that compression does not necessarily imply a loss in task quality. This extends existing data management insights by demonstrating that workload-aware KV cache strategies can simultaneously improve both system efficiency and model quality, challenging traditional assumptions about lossy compression in data systems.

7 CONCLUSIONS

LLM serving pipelines receive growing attention in the data management community, as they involve difficult data management and systems challenges. In this paper, we investigate an important component of LLM serving: the KV cache which stores previously computed attention keys and values so a transformer can reuse them during autoregressive decoding, avoiding recomputation and dramatically speeding up token generation. KV cache optimizations like quantization, pruning, and merging reduce the memory footprint and bandwidth of stored attention keys and values with the goal of enabling faster and more scalable decoding with minimal impact on output quality.

Our benchmarking study reveals various trade-offs in using the existing methods, and provides surprising but practical insights that in parts defy common data management wisdom about compression. Counter-intuitive findings such as the fact that lossy compression could *improve* accuracy in a data system challenge the way we may

think about data management for LLMs. At the same time, we found that not all compression techniques are easily deployable in general-purpose LLM serving stacks, such as CaM that could yield by far the best compression rate, or no compression at all, depending on the data set and task. This shows significant challenges when setting up and optimizing an LLM inference system. The data management community with its decade-long experience in query optimization may play a big role in workload-aware optimization of future LLM serving stacks. Our benchmarking study may be a starting point of such explorations.

ACKNOWLEDGMENTS

We thank the Kuenneth Research Group at the University of Bayreuth for providing access to an NVIDIA A100 40GB GPU, which enabled us to run our benchmarks.

REFERENCES

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*. https://openreview.net/forum?id=hmOwOZWzYE

[2] 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. 2024. 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)*. Association for Computational Linguistics, Bangkok, Thailand, 3119–3137. https://doi.org/10.18653/v1/2024.acl-long.172

[3] Sylvio Barbon Junior, Paolo Ceravolo, Sven Groppe, Mustafa Jarrar, Samira Maghool, Florence Sèdes, Soror Sahri, and Maurice Van Keulen. 2024. Are Large Language Models the New Interface for Data Pipelines?. In *Proceedings of the International Workshop on Big Data in Emergent Distributed Environments* (Santiago, AA, Chile) *(BiDEDE '24)*. Association for Computing Machinery, New York, NY, USA, Article 6, 6 pages. https://doi.org/10.1145/3663741.3664785

[4] Lennart Behme, Saravanan Thirumuruganathan, Alireza Rezaei Mahdiraji, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2023. The Art of Losing to Win: Using Lossy Image Compression to Improve Data Loading in Deep Learning Pipelines. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. 936–949. https://doi.org/10.1109/ICDE55515.2023.00077

[5] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. 2023. Token Merging: Your ViT but Faster. In *International Conference on Learning Representations*.

[6] Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. 2025. Palu: KV-Cache Compression with Low-Rank Projection. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=LWMS4pk2vK

[7] Wen Cheng, Shichen Dong, Jiayu Qin, and Wei Wang. 2025. QAQ: Quality Adaptive Quantization for LLM KV Cache. In *2025 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. 2563–2571. https://doi.org/10.1109/ICCVW69036.2025.00267

[8] Krishna Teja Chitty-Venkata, Jie Ye, Siddhisanket Raskar, Anthony Kougkas, Xian Sun, Murali Emani, Venkatram Vishwanath, and Bogdan Nicolae. 2026. PagedEviction: Structured Block-wise KV Cache Pruning for Efficient Large Language Model Inference. In *Findings of the Association for Computational Linguistics: EACL 2026*, Vera Demberg, Kentaro Inui, and Lluís Marquez (Eds.). Association for Computational Linguistics, Rabat, Morocco, 3207–3218. https://doi.org/10.18653/v1/2026.findings-eacl.168

[9] Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=mZn2Xyh9Ec

[10] Alessio Devoto, Maximilian Jeblick, and Simon Jégou. 2026. Expected Attention: KV Cache Compression by Estimating Attention From Future Queries Distribution. https://openreview.net/forum?id=VmojW15eRc

[11] Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. 2024. A Simple and Effective L_2 Norm-Based Strategy for KV Cache Compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 18476–18499. https://doi.org/10.18653/v1/2024.emnlp-main.1027

[12] Zican Dong, Junyi Li, Jinhao Jiang, Mingyu Xu, Wayne Xin Zhao, Bingning Wang, and Weipeng Chen. 2025. LongReD: Mitigating Short-Text Degradation of Long-Context Large Language Models via Restoration Distillation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 10687–10707. https://doi.org/10.18653/v1/2025.acl-long.524

[13] Haodong Duan, Jueqi Wei, Chonghua Wang, Hongwei Liu, Yixiao Fang, Songyang Zhang, Dahua Lin, and Kai Chen. 2024. BotChat: Evaluating LLMs' Capabilities of Having Multi-Turn Dialogues. In *Findings of the Association for Computational Linguistics: NAACL 2024*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 3184–3200. https://doi.org/10.18653/v1/2024.findings-naacl.201

[14] Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. 2016. Compressed linear algebra for large-scale machine learning. *Proc. VLDB Endow.* 9, 12 (Aug. 2016), 960–971. https://doi.org/10.14778/2994509.2994515

[15] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801* (2023).

[16] Aaron Grattafiori and Abhimanyu Dubey et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[17] Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. 2024. ZipCache: accurate and efficient KV cache quantization with salient token identification. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 2181, 21 pages.

[18] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: towards 10 million context length LLM inference with KV cache quantization. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 40, 34 pages.

[19] Omar Hory. 2026. TurboQuant: Open-source implementation of Google's TurboQuant. https://github.com/OmarHory/turboquant. Accessed: 2026-04-29.

[20] Alexander Isenko, Ruben Mayer, Jeffrey Jedele, and Hans-Arno Jacobsen. 2022. Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1825–1839. https://doi.org/10.1145/3514221.3517848

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

[22] Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. 2018. SketchML: Accelerating Distributed Machine Learning with Data Sketches. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1269–1284. https://doi.org/10.1145/3183713.3196894

[23] Rui Zhang Feng Liu Jiantong Jiang, Peiyu Yang. 2026. Towards Efficient Large Language Model Serving: A Survey on System-Aware KV Cache Optimization. *TechRxiv* (2026). https://doi.org/10.36227/techrxiv.176046306.66521015/v3

[24] Jang-Hyun Kim, Junyoung Yeom, Sangdoo Yun, and Hyun Oh Song. 2024. Compressed Context Memory for Online Language Model Interaction. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=64kSvC4iPg

[25] Guoliang Li, Xuanhe Zhou, and Xinyang Zhao. 2024. LLM for Data Management. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4213–4216. https://doi.org/10.14778/3685800.3685838

[26] Haoyang LI, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole HU, Wei Dong, Li Qing, and Lei Chen. 2025. A Survey on Large Language Model Acceleration based on KV Cache Management. *Transactions on Machine Learning Research* (2025). https://openreview.net/forum?id=z3JZzu9EA3

[27] Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Binazir Karimzadeh, Colorado J Reed, Chong Wang, and Chuang Gan. 2025. CommVQ: commutative vector quantization for KV cache compression. In *Proceedings of the 42nd International Conference on Machine Learning* (Vancouver, Canada) *(ICML'25)*. JMLR.org, Article 1457, 15 pages.

[28] Yuhang Li, Rong Gu, Chengying Huan, Zhibin Wang, Renjie Yao, Chen Tian, and Guihai Chen. 2025. HotPrefix: Hotness-Aware KV Cache Scheduling for Efficient Prefix Sharing in LLM Inference Systems. *Proc. ACM Manag. Data* 3, 4, Article 250 (Sept. 2025), 27 pages. https://doi.org/10.1145/3749168

[29] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. SnapKV: LLM knows what you are looking for before generation. In *Proceedings of the 38th*

International Conference on Neural Information Processing Systems (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 722, 24 pages.

[30] Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024. MiniCache: KV cache compression in depth dimension for large language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 4443, 35 pages.

[31] Yanyu Liu, Jingying Fu, Sixiang Liu, Yitian Zou, Shouhua Zhang, and Jiehan Zhou. 2026. KV Cache Compression for Inference Efficiency in LLMs: A Review. In *Proceedings of the 4th International Conference on Artificial Intelligence and Intelligent Information Processing (AIIIP '25)*. Association for Computing Machinery, New York, NY, USA, 207–212. https://doi.org/10.1145/3778534.3778567

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

[33] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: a tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML'24)*. JMLR.org, Article 1311, 13 pages.

[34] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. 2024. Leave No Context Behind: Efficient Infinite Context Transformers with Infini-attention. arXiv:2404.07143 [cs.CL] https://arxiv.org/abs/2404.07143

[35] James Pan and Guoliang Li. 2025. Database Perspective on LLM Inference Systems. *Proc. VLDB Endow.* 18, 12 (Aug. 2025), 5504–5507. https://doi.org/10.14778/3750601.3750703

[36] Noam Shazeer. 2019. Fast Transformer Decoding: One Write-Head is All You Need. arXiv:1911.02150 [cs.NE] https://arxiv.org/abs/1911.02150

[37] Woomin Song, Seunghyuk Oh, Sangwoo Mo, Jaehyung Kim, Sukmin Yun, Jung-Woo Ha, and Jinwoo Shin. 2024. Hierarchical context merging: Better long context understanding for pre-trained LLMs. (2024).

[38] Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024. You Only Cache Once: Decoder-Decoder Architectures for Language Models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=25Ioxw576r

[39] Guangtao Wang, Shubhangi Upasani, Chen Wu, Darshan Gandhi, Jonathan Lingjie Li, Changran Hu, Bo Li, and Urmish Thakker. 2025. LLMs Know What to Drop: Self-Attention Guided KV Cache Eviction for Efficient Long-Context Inference. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*. https://openreview.net/forum?id=qg9dlCcNzr

[40] Jiahao Wang, Jinbo Han, Xingda Wei, Sijie Shen, Dingyan Zhang, Chenguang Fang, Rong Chen, Wenyuan Yu, and Haibo Chen. 2025. KVCache cache in the wild: characterizing and optimizing KVCache cache at a large cloud provider. In *Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference* (Boston, MA, USA) *(USENIX ATC '25)*. USENIX Association, USA, Article 28, 18 pages.

[41] Songhao Wu, Ang Lv, xiao feng, Yufei zhang, Xun Zhang, Guojun Yin, Wei Lin, and Rui Yan. 2026. PolarQuant: Leveraging Polar Transformation for Key Cache Quantization and Decoding Acceleration. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=JCTTLKEBza

[42] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient Streaming Language Models with Attention Sinks. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=NG7sS51zVF

[43] Yifei Yang, zouying cao, Qiguang Chen, Libo Qin, Dongjie Yang, Zhi Chen, and hai zhao. 2024. KVSharer: Efficient Inference via Layer-Wise Dissimilar KV Cache Sharing. https://openreview.net/forum?id=2Akf4BBCKo

[44] Hao Yuan, Xin Ai, Qiange Wang, Peizheng Li, Jiayang Yu, Chaoyi Chen, Xinbo Yang, Yanfeng Zhang, Zhenbo Fu, Yingyou Wen, and Ge Yu. 2025. DepCache: A KV Cache Management Framework for GraphRAG with Dependency Attention. *Proc. ACM Manag. Data* 3, 6, Article 313 (Dec. 2025), 29 pages. https://doi.org/10.1145/3769778

[45] Jiayi Yuan, Hongyi Liu, Shaochen Zhong, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhuo Xu, Zirui Liu, and Xia Hu. 2024. KV Cache Compression, But What Must We Give in Return? A Comprehensive Benchmark of Long Context Capable Approaches. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 4623–4648. https://doi.org/10.18653/v1/2024.findings-emnlp.266

[46] Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. 2026. TurboQuant: Online Vector Quantization with Near-optimal Distortion Rate. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=tO3ASKZlok

[47] Rongzhi Zhang, Kuan Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and yelong shen. 2025. LoRC: Low-Rank Compression for LLMs KV Cache with a Progressive Compression Strategy. https://openreview.net/forum?id=NI8AUSAc4i

[48] Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024. CaM: Cache Merging for Memory-efficient LLMs Inference. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.), Vol. 235. PMLR, 58840–58850. https://proceedings.mlr.press/v235/zhang24n.html

[49] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1506, 50 pages.

[50] Zayd Muhammad Kawakibi Zuhri, Muhammad Farid Adilazuarda, Ayu Purwarianti, and Alham Fikri Aji. 2025. MLKV: Multi-Layer Key-Value Heads for Memory Efficient Transformer Decoding. In *Findings of the Association for Computational Linguistics: NAACL 2025*, Luis Chiruzzo, Alan Ritter, and Lu Wang (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 5531–5540. https://doi.org/10.18653/v1/2025.findings-naacl.305