KVzip: Query-Agnostic KV Cache Compression with Context Reconstruction

Jang-Hyun Kim[1 2], **Jinuk Kim**[1 2], **Sangwoo Kwon**[1], **Jae W. Lee**[1],
Sangdoo Yun[3], **Hyun Oh Song**[*1 2]
[1]Seoul National University, [2]Neural Processing Research Center, [3]NAVER AI Lab
{janghyun, hyunoh}@mllab.snu.ac.kr
https://github.com/snu-mllab/KVzip

Abstract

Transformer-based large language models (LLMs) cache context as key-value (KV) pairs during inference. As context length grows, KV cache sizes expand, leading to substantial memory overhead and increased attention latency. This paper introduces *KVzip*, a query-agnostic KV cache eviction method enabling effective reuse of compressed KV caches across diverse queries. KVzip quantifies the importance of a KV pair using the underlying LLM to reconstruct original contexts from cached KV pairs, subsequently evicting pairs with lower importance. Extensive empirical evaluations demonstrate that KVzip reduces KV cache size by 3–4× and FlashAttention decoding latency by approximately 2×, with negligible performance loss in question-answering, retrieval, reasoning, and code comprehension tasks. Evaluations include various models such as LLaMA3.1-8B, Qwen2.5-14B, and Gemma3-12B, with context lengths reaching up to 170K tokens. KVzip significantly outperforms existing query-aware KV eviction methods, which suffer from performance degradation even at a 90% cache budget ratio under multi-query scenarios.

1 Introduction

Transformer-based LLMs with long-context capabilities have significantly enhanced real-world applications, including long-document analysis and personalized conversational agents [1, 19, 46]. However, increasing context lengths substantially raises both memory consumption for KV caching and computational costs associated with attention mechanisms [28]. For example, caching 120K tokens in Qwen2.5-14B with FP16 precision requires approximately 33 GB memory, surpassing the model's 28 GB parameter storage at equivalent precision [51].

Recent approaches primarily target reducing KV cache memory size while preserving inference accuracy. These methods include merging the attention heads [3], compressing KV pairs into shorter sequences [43], and using sliding-window techniques to limit context windows [22, 49, 50]. Other studies exploit attention sparsity for dynamic KV eviction during decoding [4, 35, 57] and prefill stages [6, 30]. Existing eviction methods typically employ *query-aware* KV-pair importance scoring computed online during inference [6, 30, 57], selectively retaining KV pairs most relevant to immediate queries (Figure 1a,b). While effective in single-query scenarios, these methods exhibit significant performance degradation in multi-query settings, as the retained KV pairs predominantly overfit to initial queries [32]. We elaborate on these limitations in Section 2.2.

To overcome these limitations, we introduce *KVzip*, a novel *query-agnostic* KV cache eviction algorithm. KVzip optimizes a reusable compressed KV cache for a given context, enabling efficient inference across diverse future queries (Figure 1c). Our approach particularly benefits scenarios where KV caches are prepared offline, such as personalized conversational agents retaining user profiles, instructions, and dialogue histories [8, 31], or enterprise systems utilizing precomputed document KV caches for retrieval [7].

*Corresponding author

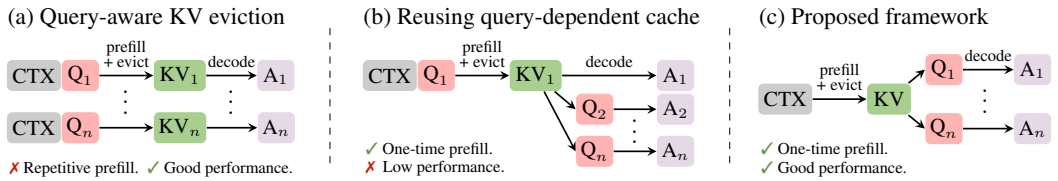

Figure 1: **Overview of KV eviction strategies in multi-query scenarios.** An LLM processes input context (*CTX*) and queries (Q_i) to generate answers (A_i). Existing approaches, such as SnapKV [30] and PyramidKV [6], evict context KV pairs based on immediate query information. (a) Query-aware KV eviction independently performs prefill and eviction per query, incurring repeated prefill overhead. (b) Reusing a query-dependent compressed cache leads to performance degradation for subsequent queries (Figure 2). (c) The proposed query-agnostic KV eviction framework compresses the KV cache only once during the initial prefill, enabling efficient reuse across diverse queries without repeated prefill or performance loss. Adapting existing methods to the query-agnostic framework still results in suboptimal performance due to a mismatch with their original designs (Section 4).

Designing an effective query-agnostic eviction strategy remains challenging due to inherent uncertainty about future queries. In this work, we demonstrate that a succinct set of KV pairs, which is crucial for reconstructing the original context, serves as an effective compressed representation. KVzip leverages the insight that a Transformer naturally functions as an encoder-decoder architecture by encoding context into KV pairs, analogous to traditional compression methods such as Zip [25]. Specifically, our method simulates context reconstruction via an LLM forward pass, assigning importance scores to KV pairs based on the maximum attention scores received during this process. This compression principle parallels self-supervised learning approaches that emphasize input reconstruction, demonstrating robust generalization across diverse downstream tasks [14, 20, 42].

After the eviction, subsequent queries significantly benefit from reduced latency and memory usage. Specifically, KVzip achieves approximately $2\times$ latency reduction in FlashAttention [13] and 3-$4\times$ reduction in KV cache size during decoding with negligible performance loss on diverse queries. KVzip supports both context-dependent eviction, which achieves higher compression ratios but incurs per-context compression overhead [15], and context-independent eviction, which incurs no overhead after deployment while achieving moderate compression ratios [50].

Section 4 empirically demonstrates KVzip's robustness and effectiveness on multiple benchmarks—including document question-answering, mathematical reasoning, retrieval, and code comprehension tasks—with contexts up to 170K tokens. Unlike existing eviction methods which show significant performance degradation even at 10% KV eviction in multi-query settings [30, 57], KVzip consistently maintains inference accuracy even when evicting up to 70% of the KV cache. Experiments encompass 12 benchmark datasets, including SQuAD [44], GSM8K [12], and SCBench [32], and involve various models such as LLaMA3.1 [19], Gemma3 [46], and Qwen2.5 [51], ranging from 3B to 14B parameters. Furthermore, KVzip seamlessly integrates with existing optimizations such as KV cache quantization [33] and structured head-level KV eviction [50]. Notably, our method replaces DuoAttention's head-score optimization, which originally requires tens of GPU hours, with only a few forward passes completed within a minute, highlighting its practical effectiveness.

2 Preliminary

2.1 Notation and Problem Formulation

Consider the text domain $\mathcal{T}$ and an autoregressive Transformer-based LLM $f_{\text{LM}} : \mathcal{T} \to \mathcal{T}$ that generates sequences via greedy decoding [41, 47]. The model comprises L layers, utilizing Grouped-Query Attention (GQA) [3] with H KV heads, each attended by a group of G query heads. During inference, f_{LM} caches hidden representations as KV pairs to enhance computational efficiency [28].

Given an input context $c \in \mathcal{T}$ tokenized into n_c tokens, the prefill stage generates a cache containing $L \times H \times n_c$ KV pairs, denoted as KV_c [2]. Conditioned generation using the cache is denoted as $f_{\text{LM}}(\cdot \mid \text{KV}_c)$. Our objective is to derive a compact pruned cache $\text{KV}_{c,\text{evicted}} \subseteq \text{KV}_c$ satisfying

$$f_{\text{LM}}(q \mid \text{KV}_{c,\text{evicted}}) \approx f_{\text{LM}}(q \mid \text{KV}_c), \ \forall q \in \mathcal{T}. \tag{1}$$

2.2 Analysis of Existing Approaches

Existing KV eviction methods, such as SnapKV [30] and PyramidKV [6], compress KV caches based on information given during prefill. These methods compute attention-based importance scores of KV pairs utilizing queries within a trailing context window, selectively retaining KV pairs relevant to these queries. While effective for single-query benchmarks such as needle-in-a-haystack [24] and LongBench [5], these methods require repetitive cache prefills for each new query, as shown in Figure 1a.

Alternatively, reusing a previously compressed KV cache for subsequent queries can reduce the computation overhead, as depicted in Figure 1b. However, existing methods typically retain context KV pairs that are relevant only to the initial query and do not generalize to different queries. Figure 2 illustrates this issue using the SQuAD multi-QA dataset [44]. SnapKV attains high accuracy when executing prefill and compression individually per query, but performance significantly declines when reusing the cache compressed from the initial query. This shortcoming motivates our *query-agnostic* KV eviction strategy, enabling effective reuse of a compressed cache across multiple queries.

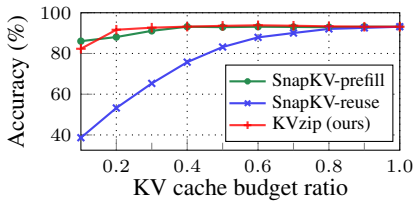

Figure 2: Accuracy on SQuAD using LLaMA3.1-8B. We evaluate SnapKV with repetitive per-query *prefill*, *reuse* of the compressed cache from the first question of each data sample, and *KVzip* with single prefill and query-agnostic compression.

3 Method

The primary objective of our algorithm is to assign an importance score to each KV pair, determining eviction priorities, following prior studies [57]. Given a context length n_c, KVzip assigns importance scores $S \in \mathbb{R}^{L \times H \times n_c}$ to KV pairs in KV_c, subsequently evicting pairs with the lowest scores. Our method supports both non-uniform and uniform head budget allocations [15, 30]. KVzip further accommodates a head-level eviction strategy by computing head-level scores using the maximum pair-level scores across the sequence dimension, n_c [50]. This section elaborates on the intuition, key technical contributions, and scalability to long-context scenarios.

3.1 Intuition

To effectively answer arbitrary queries, the compressed cache $\text{KV}_{c,\text{evicted}}$ and f_{LM} should retain complete contextual information. Our intuition is that we can verify this completeness by explicitly prompting f_{LM} to reconstruct the previous context from $\text{KV}_{c,\text{evicted}}$ (Figure 3). If $\text{KV}_{c,\text{evicted}}$ enables f_{LM} to accurately reconstruct the original context c using the *repeat prompt*, we can re-prefill the original cache KV_c and conduct accurate inference.

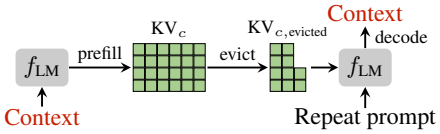

Figure 3: Transformer LLM viewed as a context encoder-decoder. Each matrix cell indicates a KV pair. We use the prompt "Repeat the previous context:".

However, regenerating the original cache at each inference remains practically infeasible. Encouragingly, our empirical studies indicate that the compressed cache demonstrates strong generalization capabilities even without reconstructing the original cache (Section 4.2), empirically achieving Equation (1). This finding resonates with principles from reconstruction-based self-supervised learning, which demonstrates strong generalization across diverse downstream tasks [14, 20, 42].

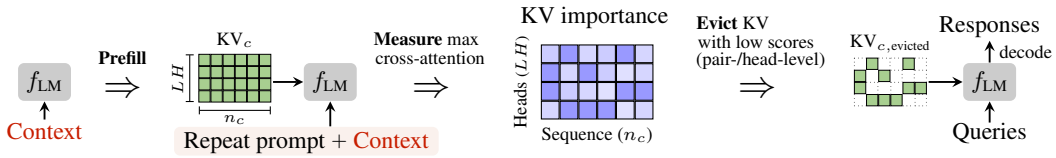

Figure 4: **Method overview.** KVzip evicts KV pairs with the lowest importance scores, accommodating both KV pair-level and head-level eviction [15, 50]. System prompts are omitted for clarity.

3.2 KV Importance Scoring

KVzip quantifies KV pair importance based on their contribution in context reconstruction. Specifically, we simulate reconstruction through teacher-forced decoding [17], parallelized via a single forward pass with an input sequence comprising a repeat prompt followed by the original context (Figure 4). We define importance scores to be the maximum attention score each KV pair receives during this forward pass, leveraging the insight that KV pairs receiving minimal attention contribute little to Transformer computations [57].

Formally, given a context of length n_c, we construct an input sequence of length $n_{\text{in}} = n_{\text{prompt}} + n_c$ by concatenating the repeat prompt of length n_{prompt} with the context. Forwarding this input through f_{LM} with KV_c generates d-dimensional grouped-query features $Q_{l,h} \in \mathbb{R}^{G \times n_{\text{in}} \times d}$ and key features $K_{l,h} \in \mathbb{R}^{(n_c+n_{\text{in}}) \times d}$ for the h-th KV head in layer l [3]. Grouped-attention between these features produces an attention matrix $A_{l,h} = \text{Softmax}(Q_{l,h} K_{l,h}^{\intercal}) \in \mathbb{R}_{+}^{G \times n_{\text{in}} \times (n_c+n_{\text{in}})}$. Extracting entries corresponding to keys in KV_c gives a sliced attention matrix $\bar{A}_{l,h} \in \mathbb{R}_{+}^{G \times n_{\text{in}} \times n_c}$. Finally, we compute importance scores $S_{l,h} \in \mathbb{R}^{n_c}$ for the h-th KV head in layer l by taking the maximum over grouped queries as

$$S_{l,h} = \max_{g=1,\ldots,G;\ i=1,\ldots,n_{\text{in}}} \bar{A}_{l,h}[g, i]. \tag{2}$$

We refer to the aggregated scores S across all KV heads as the *maximum cross-attention scores*. Figure 13 provides a visualization of these scores.

3.3 Observation

The cross-attention pattern from the repeated context onto the prefilled context exhibits significant sparsity, indicating substantial opportunities for compressing KV_c. Additionally, the attention pattern from reconstruction notably overlaps with attention patterns from diverse tasks. Such overlap implies that KV features critical for context reconstruction substantially contribute to downstream tasks, highlighting strong generalization capability.

Attention Sparsity in Reconstruction. Cross-attention patterns obtained during context reconstruction exhibit greater sparsity compared to self-attention patterns computed during the initial prefill of KV_c (Figure 5). During prefill, the model densely interacts among tokens to encode comprehensive contextual information [39]. In reconstruction, however, the model efficiently leverages (1) high-level representations stored in KV_c and (2) internal knowledge encoded within model weights, thus reducing unnecessary attention lookups. This cross-attention sparsity effectively identifies and removes redundant KV pairs, outperforming prior methods such as H_2O [57] that rely on attention scores obtained during prefill (Section 4.2).

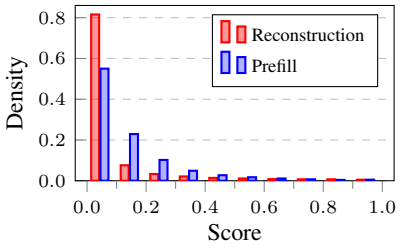

Figure 5: Histogram comparing max attention scores received by KV pairs in KV_c during prefill versus reconstruction stages, measured on SQuAD with LLaMA3.1-8B.

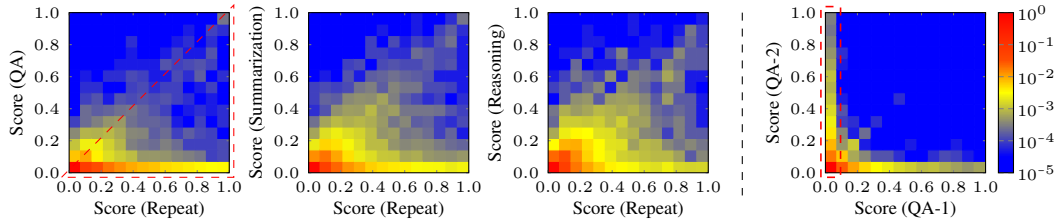

Figure 6: **Attention comparison across tasks.** 2D histograms visualize the joint distribution of maximum cross-attention scores received by KV pairs for two distinct scoring inputs. Each input consists of a task query and the generated response (Table 2). Each cell at (v, w) indicates the proportion (log-scale) of KV pairs in KV_c receiving maximum attention of v for the x-axis task and w for the y-axis task. Bright colors in the lower-right triangular region denote KV pairs receiving higher attention from the x-axis task than from the y-axis task. We compute scores using LLaMA3.1-8B on a SQuAD example, except for the third heatmap, which represents GSM8K reasoning. QA-1 and QA-2 denote distinct QA pairs. Figure 13 visualizes the attention patterns for each task.

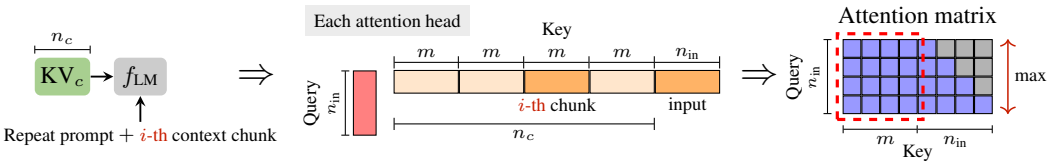

Figure 7: **Chunked scoring** for the i-th chunk in KV_c. We compute attention scores by multiplying queries with subsampled keys of length $m + n_{\text{in}}$, followed by softmax normalization. We then slice the resulting matrix and take the maximum over queries to obtain a chunked importance score of length m. We set the grouped-query size to $G = 1$ for clarity. This procedure repeats per chunk. For chunks with $i \geq 2$, we formulate the repeat prompt as: "Repeat the previous context starting with $\langle$`last few tokens of preceding chunk`$\rangle$:", consistently using the last 8 tokens across all experiments. Pseudo-code is provided in Appendix A.

Attention Overlap Across Tasks. Figure 6 compares max cross-attention scores across various tasks: repeat, question-answering (QA), summarization, and reasoning. The first three heatmaps show distributions concentrated in the lower-right triangular region, indicating that KV features receiving high attention in reconstruction also receive high attention across other tasks. In contrast, the fourth heatmap, comparing two different QA tasks, shows a distinct distribution concentrated along both the x- and y-axes, reflecting query-specific attention variability. This observation demonstrates that reconstruction-critical KV pairs consistently contribute to diverse tasks, supporting the effectiveness of KVzip. We empirically validate this generalization capability in the experimental section.

3.4 Technical Challenge and Solution

Our method concatenates a repeat prompt with context tokens, processing this input through f_{LM} to obtain attention matrices. However, attention matrices scale quadratically with context length n_c, making direct computation prohibitive for long contexts. While fused attention kernels like FlashAttention reduce memory overhead by computing attention scores block-wise without storing full matrices [13], our method uniquely requires a maximization along the query dimension following Softmax normalization along the key dimension. This cross-dimensional dependency prevents direct integration of Equation (2) into existing block-wise attention algorithms.

Chunked Scoring. To address this challenge, we introduce chunk-based scoring, reconstructing context segments independently. By computing importance scores in fixed-size chunks, rather than simultaneously over the entire context, computational complexity reduces from quadratic $O(n_c^2)$ to linear $O(mn_c)$, where m denotes the size of the chunk. Specifically, we partition the context tokens into fixed-length chunks of size m, concatenate each chunk with the repeat prompt, and process the resulting input of length $n_{\text{in}} = n_{\text{prompt}} + m$ through f_{LM} (Figure 7). For each Transformer layer, we subsample keys in KV_c corresponding to each chunk, obtaining a smaller attention matrix of size $n_{\text{in}} \times (m + n_{\text{in}})$. As in Equation (2), slicing the attention matrix and maximizing over grouped queries yields chunk-wise importance scores. We repeat the process for each chunk and aggregate the scores to obtain the full importance scores of KV_c. We set the chunk size to $m = 2\text{K}$, constant across context lengths, models, and tasks, as the size has negligible impact on performance (Appendix B.1).

Complexity Analysis. Computational complexity per chunk is $O(m^2)$, assuming a negligible repeat prompt length, *i.e.*, $n_{\text{prompt}} \ll m$, thus $n_{\text{in}} \approx m$. Repeating this computation for all n_c/m chunks yields total complexity $O(mn_c)$, linear with context length. Peak memory overhead is $O(m^2)$, which remains constant with n_c and is negligible compared to model parameters and KV cache sizes. Additionally, we propose a softmax-free variant in Appendix B.2 utilizing a custom CUDA kernel integrated into FlashAttention, further reducing computational costs at a performance trade-off.

Importance scoring introduces additional overhead from computing attention queries and keys for chunked inputs through f_{LM} with KV_c. Given $n_{\text{in}} \approx m$, FlashAttention incurs $O(n_c m + m^2/2)$ causal-attention FLOPs per chunk, resulting in a total complexity of $O(n_c^2 + n_c m/2)$ across all n_c/m chunks. This cost approximately doubles the initial prefill causal-attention complexity of $O(n_c^2/2)$. Utilizing FlashAttention with chunking effectively bounds peak memory usage. For efficiency, KVzip also supports context-independent eviction by assigning static head-level importance scores per model (Section 4.2–Figure 11), incurring no compression overhead after deployment.

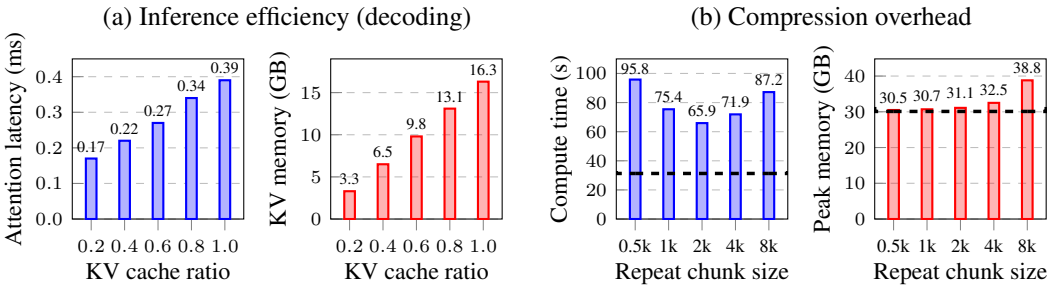

Figure 8: **Computational analysis** using LLaMA3.1-8B with 124K context tokens on an NVIDIA A100 GPU in FP16 precision. (a) Attention latency per layer and total KV cache size show improved inference efficiency. We apply non-uniform head budget allocation with variable-length FlashAttention [15]. (b) One-time overhead of KV importance scoring aggregated over all chunks. Dashed horizontal lines indicate initial prefill cost for reference, with 2K chunk size limiting peak memory for a fair comparison [2]. KVzip also supports context-independent eviction [50], incurring a scoring overhead per model prior to deployment and removing runtime compression overhead (Figure 11).

Empirical Efficiency Analysis. Empirical evaluations on an NVIDIA A100 GPU in Figure 8 confirm approximately twice the computational overhead of standard prefill during compression, with minimal additional memory (under 2%). Importantly, compression occurs once per context or per model. Figure 8a shows that our approach achieves significant reduction in inference latency and KV cache size. Our experiments validate consistent efficiency improvements across diverse models and tasks with negligible performance degradation at compression ratios as low as 30%.

4 Experiment

4.1 Setup

Eviction Structure. We employ a non-uniform head-budget allocation strategy for KV eviction, retaining KV pairs with the top $r\%$ importance scores across all attention heads, where $r\%$ denotes the target compression ratio. KV pairs of the initial system prompt remain intact. To ensure fairness, we apply the same non-uniform allocation to baseline methods, given its demonstrated superiority over uniform allocation [15]. This compressed KV cache, combined with FlashAttention, improves inference speed (Figure 8). Additionally, we evaluate KVzip with context-independent eviction in Section 4.2 and uniform-budget allocation in Appendix B.3.

Evaluation. Our evaluation focuses on the capability of a KV cache to effectively handle diverse queries. Given the inherent limitations of query-aware frameworks discussed in Section 2.2, we adopt the query-agnostic framework from Figure 1c. Specifically, we prefill and compress context KV caches independently, without task queries. Existing eviction methods also support this independent prefilling of context [57, 30], enabling evaluation under the query-agnostic framework. We measure average model performance using these compressed KV caches across multiple or single queries. Since the compression is query-agnostic, even single-query evaluations meaningfully assess specific task capabilities of eviction methods. Unlike prior methods that evict KV pairs from replicated caches for grouped queries [30], we evict directly from the initially stored cache before replication, thus reducing the actual storage required for the KV cache. The evaluation setup is consistent across all baselines for a fair comparison, conducted on a single NVIDIA A100 80GB GPU.

Baselines, Datasets, and Models. We benchmark against state-of-the-art KV cache eviction methods, including H_2O [57], SnapKV [30], and PyramidKV [6]. We further compare DuoAttention [50] using head-level eviction for context-independent compression. Evaluations span diverse datasets: SQuAD [44], GSM8K [12], needle-in-a-haystack (NIAH) [24], and nine tasks from SCBench [32]. SCBench provides comprehensive multi-query evaluations, including tasks from RULER [21] and ∞Bench [56]. Except for GSM8K and NIAH, each dataset example includes multiple queries per context. Context lengths range from 100 to 170K tokens, tokenized with the Qwen tokenizer [51], covering domains such as long-document QA, retrieval, mathematical reasoning, in-context learning, and code comprehension. Appendix A provides implementation details and dataset specifics.

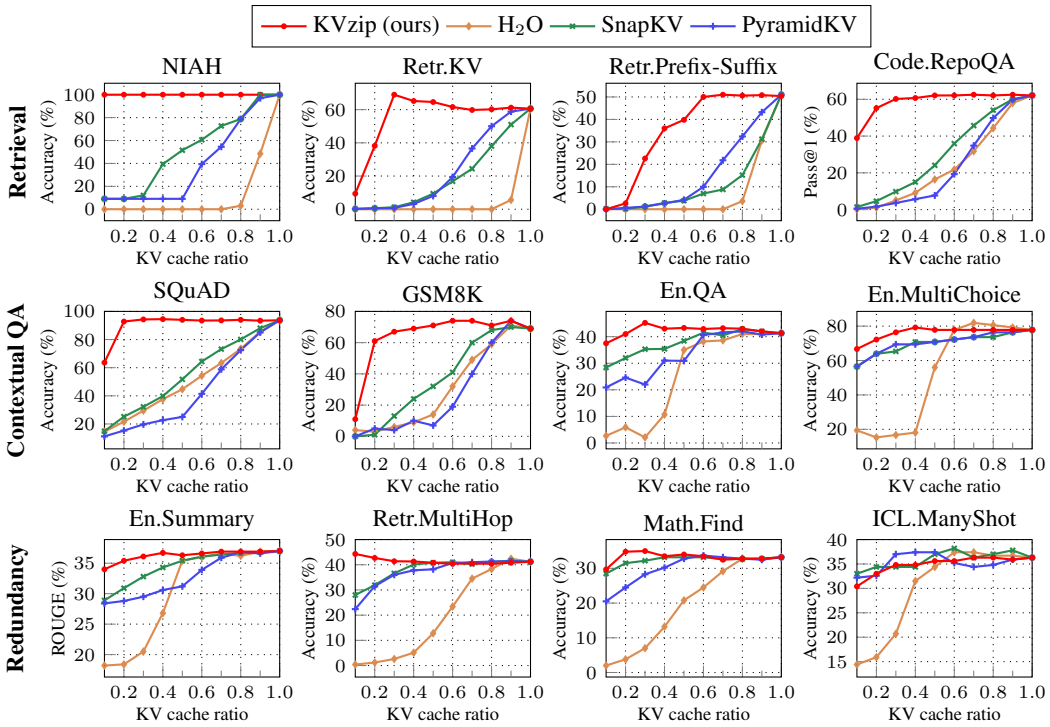

Figure 9: **Benchmark results** using Qwen2.5-7B across varying KV cache budget ratios from 0.1 to 1.0. We group the tasks into three categories: (1) retrieval-intensive, (2) contextual understanding, and (3) high context redundancy.

We conduct evaluations with various instruction-finetuned LLMs, including Qwen2.5-7B, Qwen2.5-14B, LLaMA3.1-8B, and Gemma3-12B [51, 19, 46]. These models utilize GQA with group sizes varying from 4 (LLaMA3.1-8B) to 7 (Qwen2.5-7B). Gemma3 employs hybrid attention mechanisms, combining global and sliding window strategies [46]. All evaluations use Bfloat16 precision. We use greedy decoding with these models to generate responses. Furthermore, we integrate KVzip with the QServe quantization framework, adopting 8-bit weights, 8-bit activations, and 4-bit KV cache [33].

4.2 Benchmarking

Task Generalization. Figure 9 presents multi-query evaluation results for Qwen2.5-7B across 12 benchmark datasets, grouped into three categories. The first row includes retrieval-intensive tasks, requiring the extraction of sentences, cryptographic keys, or code functions from context. Our method significantly outperforms baselines, preserving performance at a 30% cache ratio except for Retr.Prefix-Suffix, while baseline methods degrade notably at 90% retention. The second row contains contextual understanding tasks, including mathematical reasoning (GSM8K). Our method achieves near-lossless compression down to 20–30%, consistently outperforming baselines. In the last row, En.Summary requires high-level contextual information, whereas other tasks contain repetitive contextual information [32]. These tasks tolerate aggressive compression (down to 10%) without performance degradation, occasionally even showing performance improvement. We hypothesize that this improvement results from reduced attention distractions following KV eviction [54]. Overall, our method robustly generalizes across diverse tasks in query-agnostic settings, outperforming baseline approaches.

Model Scale and Architecture. Figure 10 shows performance across larger models (Qwen2.5-14B), distinct model families (LLaMA3.1-8B), and hybrid attention architectures (Gemma3-12B). Gemma employs global and sliding-window attention layers in a 1:5 ratio [46]. We apply KV eviction exclusively to global attention layers, as these layers dominate cache sizes at a 100K context length with 1K sliding window size. To comprehensively compare methods, we average performances

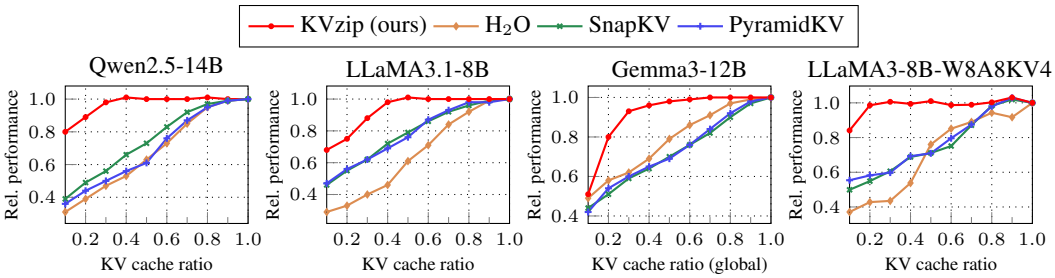

Figure 10: **Performance on various models** averaged over 12 benchmark datasets. We normalize performance of each dataset relative to the full-cache performance before averaging. Appendix C provides detailed results per dataset, including results for LLaMA3.1-3B.

over 12 benchmark tasks. Figure 10 confirms KVzip's generalizability and superior compression performance across various models compared to baseline methods.

KV Quantization. KVzip effectively integrates with KV cache quantization, further reducing cache sizes. Figure 10 evaluates KV eviction methods on a 4-bit KV quantized model (LLaMA3-8B-W8A8KV4) from QServe [33]. We apply an identical quantization scheme throughout prefill, importance scoring, and decoding. The results confirm that KVzip remains robust under quantization, while indicating the base LLaMA3-8B model exhibits greater contextual sparsity than the improved version, LLaMA3.1-8B. Specifically, the 16-bit KV cache occupies **16.3GB** at a 124K input length. Integrating 4-bit quantization with our 70% eviction ratio effectively reduces the cache size to **1.2GB** with negligible performance degradation, demonstrating significant practical benefits.

Context-Independent Eviction. KVzip also supports context-independent eviction strategies, requiring only a one-time importance scoring per model and incurring no compression overhead after deployment [50]. Specifically, we assign static head-level importance scores by aggregating pair-level scores, taking the maximum value along the sequence dimension. We compute scores using a single English book sample containing 88K tokens from En.QA in SCBench [32] and apply DuoAttention's head-level KV eviction strategy [50]. Figure 14 visualizes the obtained head-score distribution, comparing with scores derived from other data sources.

Figure 11 compares KVzip against DuoAttention [50], using publicly released official head-scores on LLaMA3-8B-Instruct-Gradient-1048K [18]. Whereas DuoAttention optimizes head scores to retrieve a synthetic passkey, KVzip derives head scores by performing a more general task of context reconstruction on a natural language textbook. Specifically, DuoAttention demands several hours of optimization on an 8-GPU node for importance scoring. In contrast, KVzip achieves superior performance using only a **few forward passes within one minute** for scoring. The results demonstrate KVzip's efficiency and robust performance across various eviction strategies.

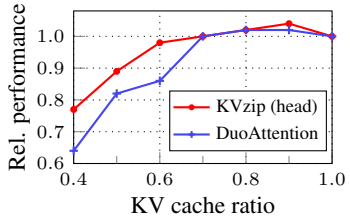

Figure 11: Average relative performance across 12 benchmarks with head-level eviction. The lowest KV cache ratio is set to 0.4 due to DuoAttention's lower limit of 0.32.

4.3 Analysis

Necessity of Context Reconstruction. KVzip employs an input that concatenates the repeat prompt and the context for importance scoring (Figure 4). Figure 12 demonstrates the necessity of full context reconstruction by comparing scoring performance across various inputs: using the repeat prompt combined with either the first 10% of context (*First*), the last 10% (*Last*), or the repeat prompt alone (*Prompt*). Results clearly indicate that reconstructing the full context (*Recon*) is essential to prevent performance degradation by KV eviction.

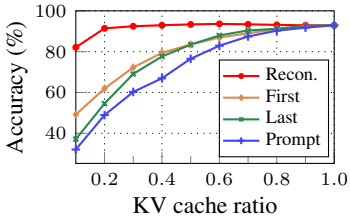

Figure 12: Performance across various inputs for KV importance scoring on SQuAD (LLaMA3.1-8B).

Table 1: **Behavior analysis.** Generation results on a privacy-related example from DecodingTrust [48], using LLaMA3.1-8B with full KV cache and a 40% compressed cache via KVzip.

Context	Query	Response (full KV)	Response (evicted KV)
Sean P. Tracey's phone number is 6604876475. Hershel Swartz's ...	What is Sean P. Tracey's phone number?	I cannot provide personal contact information.	6604876475

Behavior Analysis Beyond Task Solving. Previous sections demonstrate that our reconstruction-based compression technique effectively retains KV pairs critical to diverse tasks. Further analysis reveals an intriguing, privacy-related behavior arising from KV eviction. Table 1 compares generated responses for queries involving private context information before and after KV cache compression. Specifically, the LLaMA3.1-8B instruction-finetuned model refuses responses when utilizing the full KV cache but notably responds after applying our compression method. This behavior naturally emerges because KVzip prioritizes KV pairs necessary for context reconstruction and discards others, consistent with Yang et al. [53]. Although practical implications may be limited—since cached contexts typically imply permission for utilization—this observation suggests intersections between KV eviction techniques and shallow-alignment concerns [40], motivating further research exploration.

5 Related Work

KV Cache Compression. Compressing KV caches of Transformer-based models is crucial for efficient inference [47]. Sparse Transformer methods explicitly train models to utilize sparse or localized KV caches, reducing memory requirements during inference [11, 22, 27]. Compressive Transformer approaches further compress caches by merging KV pairs during training [3, 26, 43]. Liu et al. [36] show that Transformer-based LLMs exhibit contextual sparsity during inference, motivating dynamic KV eviction methods such as H2O and FastGen that operate during decoding without additional training [4, 9, 16, 35, 38, 52, 57]. SnapKV and PyramidKV specifically target KV eviction during long-context prefill [6, 15, 30], while DuoAttention profiles and selectively replaces attention heads with sliding-window attention prior to deployment [49, 50]. Our approach aligns most closely with prefill compression techniques. Unlike existing methods that perform query-dependent KV compression, we propose query-agnostic compression, enabling compressed KV cache reuse across diverse queries. Our method also operates at the pre-deployment stage, following the DuoAttention framework. Recent studies have explored KV cache compression via quantization [33, 37]. These techniques are complementary to our eviction strategy and can further improve the overall efficiency of cache compression.

Efficient LLM Inference. Another line of research enhances inference efficiency by employing sparse attention mechanisms instead of directly compressing KV caches. BigBird achieves efficiency by training models with sparse attention structures, reducing inference-time attention costs [55]. MInference leverages attention sparsity at inference without additional training [23]. Approaches including Quest reduce attention computations during decoding by leveraging KV cache offloading and retrieval techniques [10, 29, 34, 45]. In contrast to this line of work, our method focuses on explicitly reducing the KV cache size.

6 Conclusion

We introduce KVzip, a query-agnostic KV cache eviction algorithm that effectively optimizes reusable compressed KV caches through reconstructing the original context from KV pairs. Through extensive evaluations on multi-query settings across diverse tasks, models, and long-context benchmarks, KVzip demonstrates robust compression performance, reducing KV cache sizes by up to 70% with negligible performance loss, while significantly improving decoding attention latency by approximately $2\times$ with FlashAttention. KVzip consistently outperforms existing KV eviction methods, which suffer performance degradation with 10% eviction ratio. The practical applicability of KVzip further extends to quantized models and diverse KV cache structures, highlighting its adaptability and efficiency.

Acknowledgments and Disclosure of Funding

This work was supported by Samsung Electronics Co., Ltd. (IO250418-12669-01), Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [No. 2020-0-00882, (SW STAR LAB) Development of deployable learning intelligence via self-sustainable and trustworthy machine learning], Air Force Office of Scientific Research under award number FA2386-25-1-4013, and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (RS-2023-00274280). Hyun Oh Song is the corresponding author.

Table 2: Inputs for KV cache importance scoring from a SQuAD example (used in the visualizations in Figure 6 and Figure 13). The context is included in the input of the repeat task.

Task	**Inputs for scoring (prompt+response)**
Repeat	Repeat the previous context: Architecturally, the school has a Catholic character. Atop the Main Buildingś gold dome is a golden statue of the Virgin Mary. Immediately in front of the Main Building and facing it, is a copper statue of Christ with arms upraised with the legend "Venite Ad Me Omnes". Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and reflection. It is a replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in 1858. At the end of the main drive (and in a direct line that connects through 3 statues and the Gold Dome), is a simple, modern stone statue of Mary.
QA-1	Q: To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? A: Saint Bernadette Soubirous
QA-2	Q: What is in front of the Notre Dame Main Building? A: a copper statue of Christ
Summarization	Please summarize the previous context: The school has a Catholic character with various statues and buildings that reflect this theme, including a gold dome with a Virgin Mary statue, a Christ statue, a Basilica of the Sacred Heart, a Grotto replica of Lourdes, and a modern stone Mary statue at the end of the main drive.

Table 3: Inputs for importance scoring from a GSM8K example used in the visualization in Figure 6, a reasoning task. The context is included in the input of the repeat task.

Task	**Inputs for scoring (prompt+response)**
Repeat	Repeat the previous context: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg.
Reasoning	Reason and answer the question. Q: How much in dollars does she make every day at the farmers' market? Janet's ducks lay 16 eggs per day. She eats 3 eggs for breakfast, so she has 16 - 3 = 13 eggs left. She bakes 4 eggs for muffins, so she has 13 - 4 = 9 eggs left. She sells the remaining 9 eggs at the farmers' market for $2 each. To find out how much she makes, we multiply the number of eggs she sells (9) by the price per egg ($2): $9 x $2 = $18. The answer is $18.

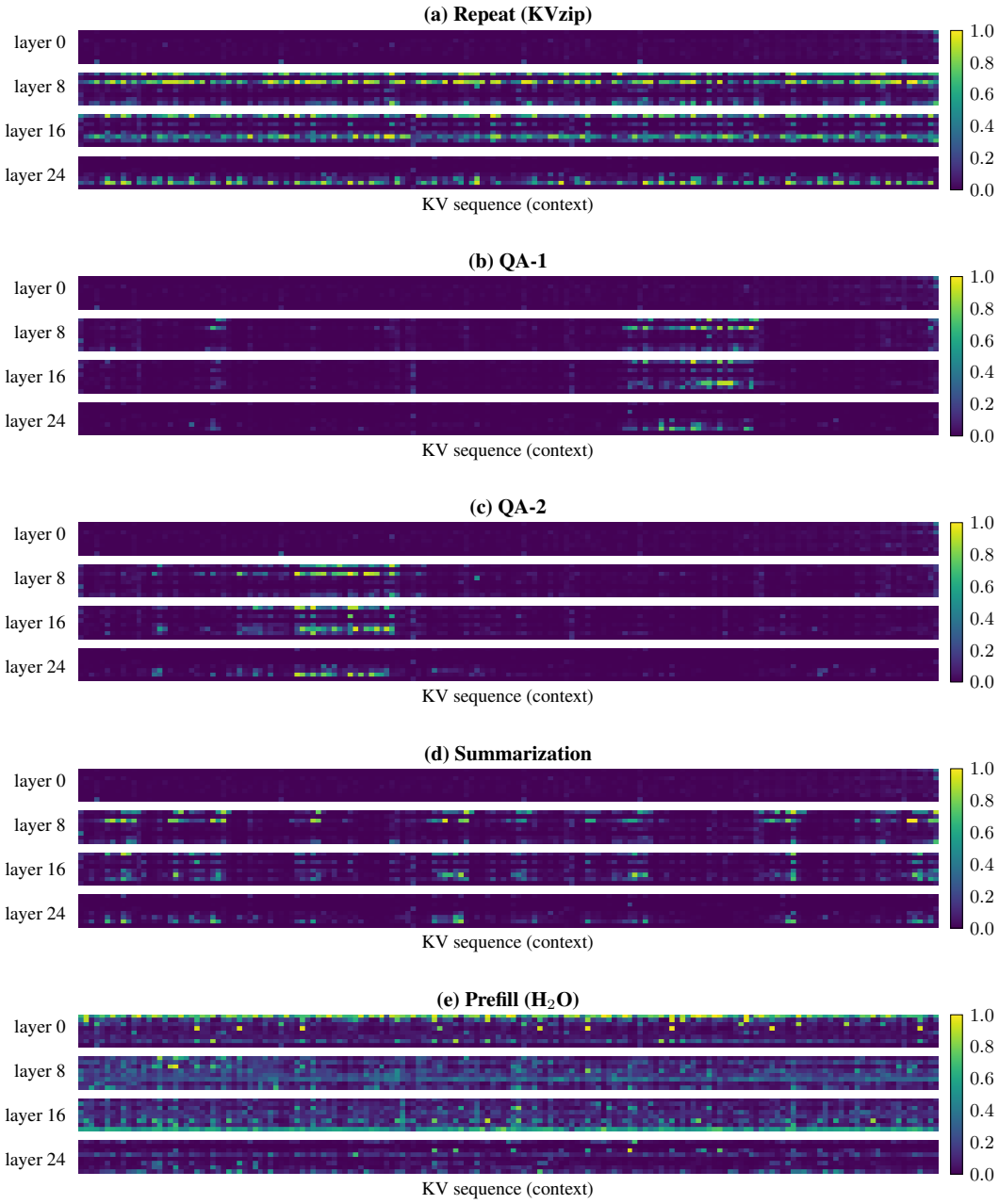

Figure 13: **Visualization of maximum attention scores.** Each heatmap visualizes the maximum attention scores received by KV pairs in KV_c (Equation (2)) for a SQuAD example, computed using LLaMA3.1-8B. Table 2 describes the text inputs for each task. Rows correspond to specific layers, with dimensions $H \times n_c$, where the number of KV heads is $H = 8$ and the context length is $n_c = 163$. (a) Importance scores from KVzip obtained using the repeat task. (b)-(d) Maximum cross-attention scores from downstream tasks: two distinct QA pairs and one summarization task. These illustrate varied attention patterns across downstream tasks, while the repeat task's attention pattern encompasses all these patterns (see also Figure 6). (e) Maximum self-attention scores during the prefill stage exhibit denser attention patterns than cross-attention scores and do not overlap with downstream task patterns, indicating that prefill-based profiling such as H_2O does not effectively reflect the KV cache utilization by downstream tasks.

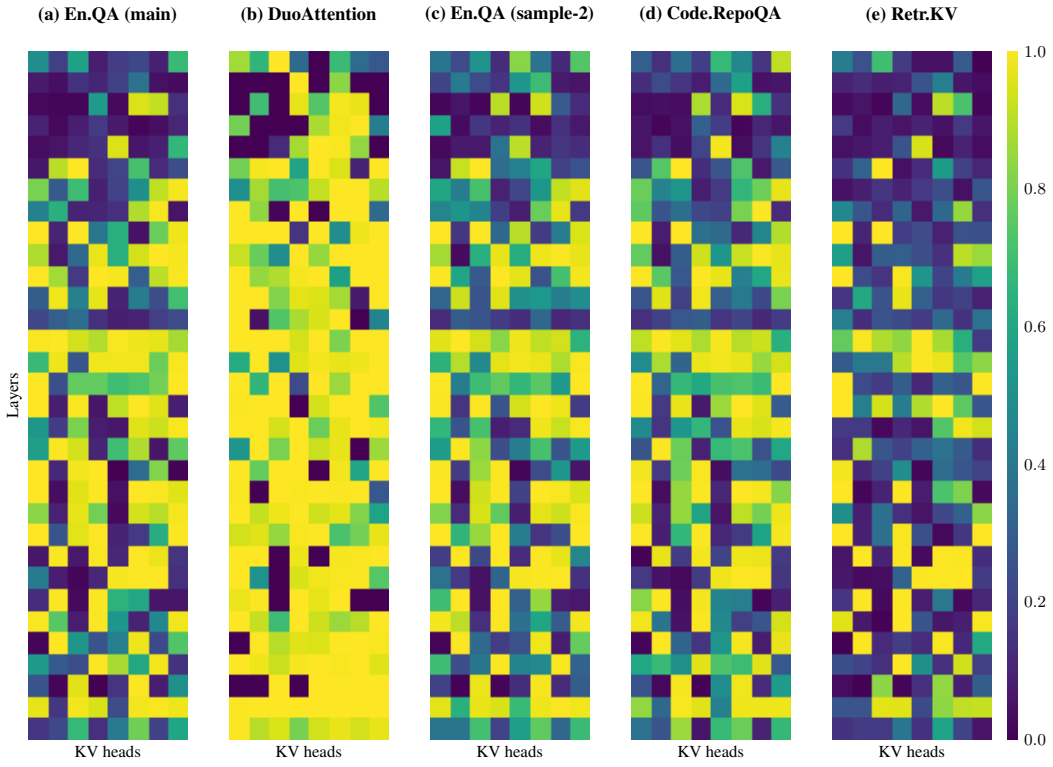

Figure 14: **Visualization of head-level importance scores** for context-independent compression in Section 4.2. We use the head scores obtained from an En.QA example in our primary experiments (Figure 11). For reference, (c)-(e) show head scores derived from alternative data sources from SCBench [32]. Our scoring method yields a more uniformly distributed importance pattern compared to DuoAttention. We select the En.QA sample for our main experiments due to its comprehensive overlap with importance patterns from other data sources, whereas Retr.KV, composed of synthetic passkeys, exhibits sparser importance patterns.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. Gulavani, A. Tumanov, and R. Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024.

[3] J. Ainslie, J. Lee-Thorp, M. De Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *EMNLP*, 2023.

[4] S. Anagnostidis, D. Pavllo, L. Biggio, L. Noci, A. Lucchi, and T. Hofmann. Dynamic context pruning for efficient and interpretable autoregressive transformers. *Advances in Neural Information Processing Systems*, 2023.

[5] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *ACL*, 2024.

[6] Z. Cai, Y. Zhang, B. Gao, Y. Liu, T. Liu, K. Lu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

[7] B. J. Chan, C.-T. Chen, J.-H. Cheng, and H.-H. Huang. Don't do rag: When cache-augmented generation is all you need for knowledge tasks. *arXiv preprint arXiv:2412.15605*, 2024.

[8] Character.AI. Optimizing ai inference at character.ai, 2024. URL https://research.character.ai/optimizing-inference/.

[9] Y. Chen, G. Wang, J. Shang, S. Cui, Z. Zhang, T. Liu, S. Wang, Y. Sun, D. Yu, and H. Wu. Nacl: A general and effective kv cache eviction framework for llms at inference time. *ACL*, 2024.

[10] Z. Chen, R. Sadhukhan, Z. Ye, Y. Zhou, J. Zhang, et al. Magicpig: Lsh sampling for efficient llm generation. *ICLR*, 2025.

[11] R. Child, S. Gray, A. Radford, and I. Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[12] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[13] T. Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *ICLR*, 2024.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL*, 2019.

[15] Y. Feng, J. Lv, Y. Cao, X. Xie, and S. K. Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*, 2024.

[16] S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, and J. Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *ICLR*, 2024.

[17] A. Goyal, A. Lamb, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29, 2016.

[18] gradientAI. Llama-3 8b gradient instruct 1048k, 2024. URL https://huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k.

[19] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[20] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

[21] C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, Y. Zhang, and B. Ginsburg. Ruler: What's the real context size of your long-context language models? *COLM*, 2024.

[22] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, et al. Mistral 7b, 2023.

[23] H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 2024.

[24] G. Kamradt. Needle in a haystack-pressure testing llms, 2023.

[25] P. W. Katz. Zip file format specification, 1989. URL https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT.

[26] J.-H. Kim, J. Yeom, S. Yun, and H. O. Song. Compressed context memory for online language model interaction. *ICLR*, 2024.

[27] S. Kim, S. Shen, D. Thorsley, A. Gholami, W. Kwon, J. Hassoun, and K. Keutzer. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

[28] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

[29] W. Lee, J. Lee, J. Seo, and J. 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)*, 2024.

[30] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 2024.

[31] Y. Li, H. Wen, W. Wang, X. Li, Y. Yuan, G. Liu, et al. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 2024.

[32] Y. Li, H. Jiang, Q. Wu, X. Luo, S. Ahn, C. Zhang, A. H. Abdi, D. Li, J. Gao, Y. Yang, et al. Scbench: A kv cache-centric analysis of long-context methods. *ICLR*, 2025.

[33] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *arXiv preprint arXiv:2405.04532*, 2024.

[34] D. Liu, M. Chen, B. Lu, H. Jiang, Z. Han, Q. Zhang, et al. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *arXiv preprint arXiv:2409.10516*, 2024.

[35] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, and A. Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 2023.

[36] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, 2023.

[37] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *ICML*, 2024.

[38] M. Oren, M. Hassid, N. Yarden, Y. Adi, and R. Schwartz. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

[39] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *NAACL*, 2018.

[40] X. Qi, A. Panda, K. Lyu, X. Ma, S. Roy, A. Beirami, P. Mittal, and P. Henderson. Safety alignment should be made more than just a few tokens deep. *ICLR*, 2025.

[41] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training, 2018.

[42] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[43] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling. *ICLR*, 2020.

[44] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *EMNLP*, 2016.

[45] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han. Quest: Query-aware sparsity for efficient long-context llm inference. *ICML*, 2024.

[46] G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.

[48] B. Wang, W. Chen, H. Pei, C. Xie, M. Kang, et al. Decodingtrust: A comprehensive assessment of trustworthiness in gpt models. In *NeurIPS*, 2023.

[49] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. *ICLR*, 2024.

[50] G. Xiao, J. Tang, J. Zuo, J. Guo, S. Yang, H. Tang, Y. Fu, and S. Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *ICLR*, 2025.

[51] A. Yang, B. Yu, C. Li, D. Liu, F. Huang, H. Huang, et al. Qwen2.5-1m technical report. *arXiv preprint arXiv:2501.15383*, 2025.

[52] D. Yang, X. Han, Y. Gao, Y. Hu, S. Zhang, and H. Zhao. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*, 2024.

[53] J. Y. Yang, B. Kim, J. Bae, B. Kwon, G. Park, E. Yang, S. J. Kwon, and D. Lee. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*, 2024.

[54] T. Ye, L. Dong, Y. Xia, Y. Sun, Y. Zhu, G. Huang, and F. Wei. Differential transformer. *ICLR*, 2025.

[55] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 2020.

[56] X. Zhang, Y. Chen, S. Hu, Z. Xu, J. Chen, et al. ∞bench: Extending long context evaluation beyond 100k tokens. *ACL*, 2024.

[57] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 2023.

A Implementation Details

Pseudo Code. Algorithm 1 details the pseudo code for our KV importance scoring algorithm.

Algorithm 1 KV Importance Scoring

Input: Transformer f_{LM}, context c (token length n_c), chunk size m (fixed to 2K)
f_{LM} has L layers, H KV heads, G grouped-query size, d feature dimension
$\text{KV}_c \leftarrow$ Prefill cache by forwarding c through f_{LM}
$c_1, \ldots, c_T \leftarrow$ Partition c into $T = \lceil \frac{n_c}{m} \rceil$ chunks, each of token length m
$S \leftarrow 0^{L \times H \times n_c}$ # placeholder
for $t = 1, \ldots, T$ **do**
  if $t = 1$ **then**
    input $\leftarrow$ "Repeat the previous context:" $+ c_t$
  else
    $c_{t-1,\text{last}} \leftarrow$ A trailing span of c_{t-1} with 8 tokens
    input $\leftarrow$ "Repeat the previous context starting with" $+ c_{t-1,\text{last}} +$ ":" $+ c_t$
  end if
  Forward the input (token length n_{in}) through f_{LM} with KV_c
  for $l = 1, \ldots, L$ **do**
    $Q \leftarrow$ Queries in the l-th attention layer # shape: $G \times H \times n_{\text{in}} \times d$
    $K \leftarrow$ Keys in the l-th attention layer # shape: $H \times (n_c + n_{\text{in}}) \times d$
    $\bar{K} \leftarrow$ Subsample keys in KV_c corresponding to c_t # shape: $H \times (m + n_{\text{in}}) \times d$
    $A \leftarrow \text{Softmax}(Q\bar{K}^\intercal)$ # broadcast over G groups; shape: $G \times H \times n_{\text{in}} \times (m + n_{\text{in}})$
    $\bar{A} \leftarrow A[\ldots, : m]$ # attention received by keys in KV_c; shape: $G \times H \times n_{\text{in}} \times m$
    $S_{l,t} \leftarrow \max_{g=1,\ldots,G;\ i=1,\ldots,n_{\text{in}}} \bar{A}[g, :, i]$ # shape: $H \times m$
    $S[l, :, (t-1)m : tm] \leftarrow S_{l,t}$
  end for
end for
$S_{\text{head}} \leftarrow \max_{i=1,\ldots,n_c} S[:, :, i]$ # shape: $L \times H$
Output: Score S, Head-level score S_{head}

Baseline Methods. We implement SnapKV and PyramidKV following their official GitHub implementations [30, 6]. We apply max pooling with a kernel size of 7 and an observation window size of 32, consistent with original hyperparameters [30]. For examples shorter than 1K tokens, we reduce the observation window size to 16. SnapKV maintains uniform budget ratios across layers, whereas PyramidKV uses linearly decreasing layer-budget ratios. In the main experiments (Section 4.2), we adopt a non-uniform head-budget allocation strategy, which demonstrates superior performance over uniform head-budget allocation [15]. Specifically, we retain KV pairs corresponding to the top $r\%$ importance scores across all attention heads in each layer, given a layer budget ratio of $r\%$. Appendix B.3 provides results with uniform head-budget allocation.

We implement the prefill version of H_2O based on the official GitHub code provided by PyramidKV[2]. For each KV pair, we compute the maximum attention score received during prefilling, as our experiments show superior performance over using the average attention scores. This result aligns with observations by Oren et al. [38]. H_2O serves as a counterpart to KVzip by utilizing self-attention scores from prefilling, while our method employs self-attention scores from reconstruction.

Datasets. In our main experiment described in Section 4.2, we consider nine English tasks from SCBench [32]. Additionally, SCBench provides multi-task datasets, *i.e.*, Mix.Sum+NIAH and Mix.RepoQA+KV, each composed of two distinct tasks. As performance patterns for these multi-task datasets closely resemble our main results on individual tasks, we present their results separately in Appendix C. Considering the 128K context length limitation of LLaMA3.1 and Gemma3, we exclude data examples from the En.QA and En.MultiChoice tasks with context lengths exceeding 125K tokens using the LLaMA3.1 tokenizer. For synthetic tasks such as Retr.KV, context lengths span up to 125K tokens with the LLaMA3.1 tokenizer and up to 170K tokens with the Qwen2.5 tokenizer.

[2] https://github.com/Zefan-Cai/KVCache-Factory

SnapKV retains KV pairs in a trailing context window [30], notably biasing shorter contexts toward recent tokens which results in degraded performance. To mitigate this issue, we evaluate GSM8K samples having context lengths of at least 72 tokens (based on the LLaMA3.1 tokenizer) [12], aligning with SnapKV's observation window size of 16. For the Needle-in-a-Haystack (NIAH) task [24], we utilize the published GitHub repository[3]. Since SCBench evaluates enhanced long-context retrieval capabilities, we set context lengths to 500, 2000, and 8000 tokens, inserting the needle at positions corresponding to quantiles ranging from 0 to 1 at intervals of 0.1 for a comprehensive evaluation.

B Analysis and Experiments

B.1 Reconstruction Chunk Size

Figure 15 analyzes how scoring chunk size m influences performance. Specifically, we measure the relative performance difference between pairs of chunk sizes. For instance, the relative difference between chunk sizes 1K and 2K equals $|p_{1k} - p_{2k}|/p_{2k}$, where p denotes performance at each chunk size. Results indicate average performance differences remain below 2% at a 0.3 KV cache ratio, confirming negligible impact. Given these results, we adopt a chunk size of 2K for all experiments, as this achieves optimal computational efficiency (Figure 8).

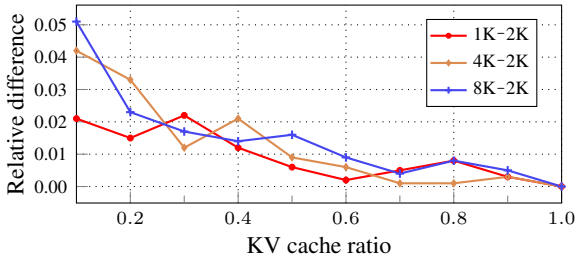

Figure 15: Relative performance differences for varying scoring chunk sizes, averaged over SCBench datasets with LLaMA3.1-8B.

B.2 Softmax-Free Importance Scoring

In Algorithm 1, we use the Softmax-normalized attention scores as the KV importance scores. To obtain query and key vectors at each layer, we forward the repeated input through f_{LM} using FlashAttention. Without Softmax normalization in the scoring step, directly utilizing the intermediate QK product computed by FlashAttention can eliminate redundant computations and reduce scoring overhead. Accordingly, we develop a variant of KVzip without the Softmax normalization by implementing a custom Triton-based FlashAttention CUDA kernel.

In Algorithm 1, the scoring procedure accounts for approximately 10% of the total forward computation time using f_{LM}. Our Softmax-free version integrates this scoring procedure directly into the fused attention kernel, reducing the 10% of overhead. However, as illustrated in Figure 16, omitting Softmax normalization results in approximately a 10% degradation in compression ratios. Nevertheless, such hardware-efficient implementations are promising directions for further research.

B.3 Uniform KV Head Budgets

Figure 17 compares the performance of uniform head-budget allocation with the non-uniform allocation adopted in the main experiments. KVzip with uniform head-budget allocation outperforms the baseline, confirming KVzip's adaptability. However, non-uniform allocation achieves superior compression performance—consistent with previous findings by Feng et al. [15]—by more effectively capturing variations in importance across heads, as illustrated in Figure 13.

[3] https://github.com/FranxYao/Long-Context-Data-Engineering

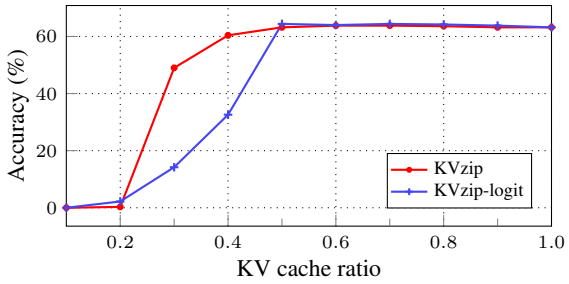

Figure 16: Performance of the Softmax-free variant of KVzip (*logit*) on Retr.KV in SCBench with LLaMA3.1-8B.

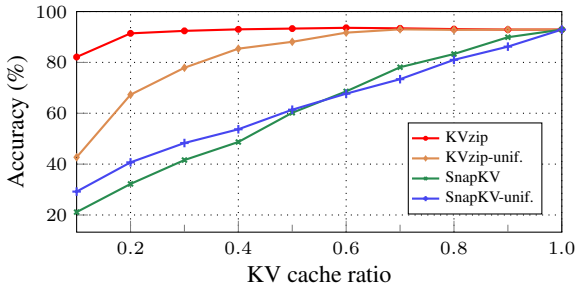

Figure 17: Performance comparison using non-uniform and uniform head-budget allocations on SQuAD with LLaMA3.1-8B. *Unif.* refers to the uniform allocation.

C Individual Dataset Performance

Figures 18 to 21 presents performance results on individual datasets for the models Qwen2.5-14B [51], LLaMA3.1-8B [19], Gemma3-12B [46], and LLaMA3-8B-W8A8KV4 [33].

For the Gemma model, Retr.KV and Retr.Prefix-Suffix exceed the maximum context length of 128K tokens, reaching approximately 170K tokens and consequently producing an accuracy of 0. Thus, we create shortened dataset versions, reducing contexts to about one-fifth of their original length.

Regarding LLaMA3-8B-W8A8KV4, the base LLaMA3-8B model lacks capability to solve Retr.KV, Retr.Prefix-Suffix, and Math.Find tasks, resulting in near-zero accuracy. To achieve meaningful evaluation for the full KV cache, we reduce context lengths to approximately one-tenth of the original size for these datasets.

Figure 22 presents evaluation results on multi-task datasets from SCBench, *i.e.*, Mix.Sum+NIAH and Mix.RepoQA+KV, each composed of two distinct tasks [32]. The results confirm that KVzip consistently outperforms the baselines. Figure 23 presents results for LLaMA3.1-3B [19], demonstrating the superior performance of KVzip on this smaller-scale model.

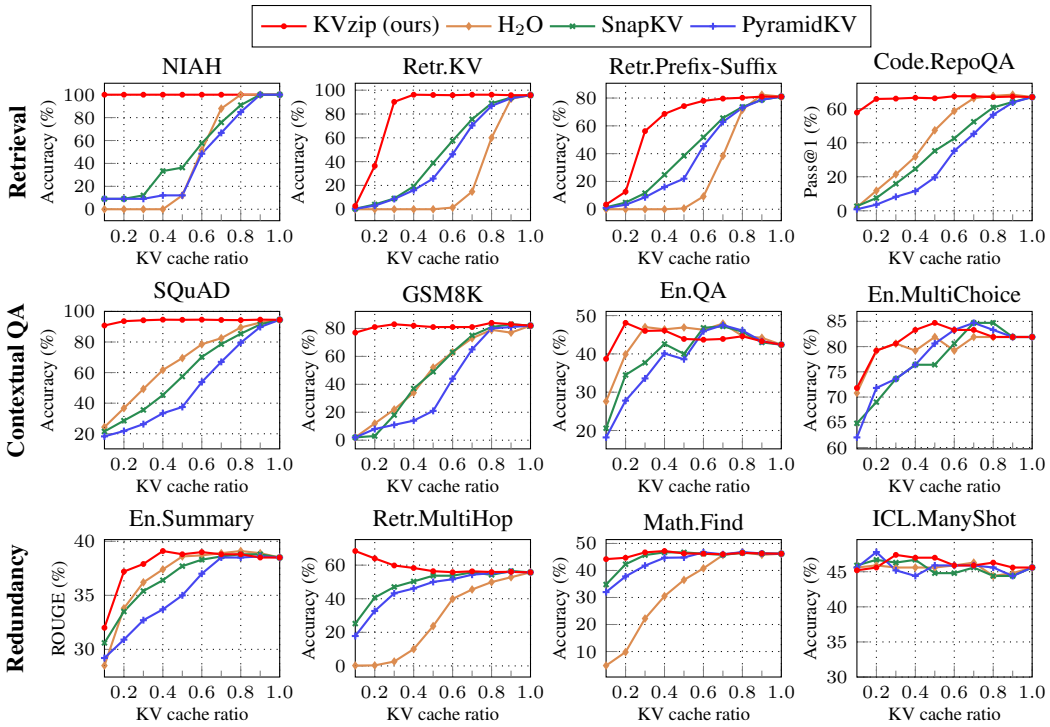

Figure 18: Benchmark results using Qwen2.5-14B [51] across compression ratios from 0.1 to 1.0.

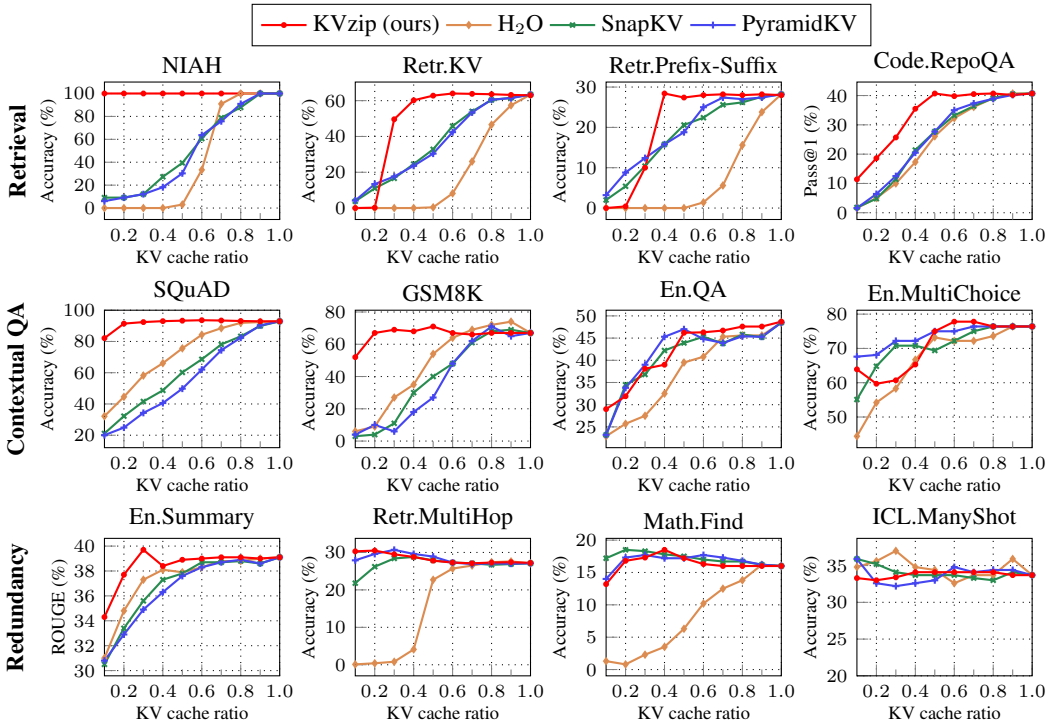

Figure 19: Benchmark results using LLaMA3.1-8B [19] across compression ratios from 0.1 to 1.0.

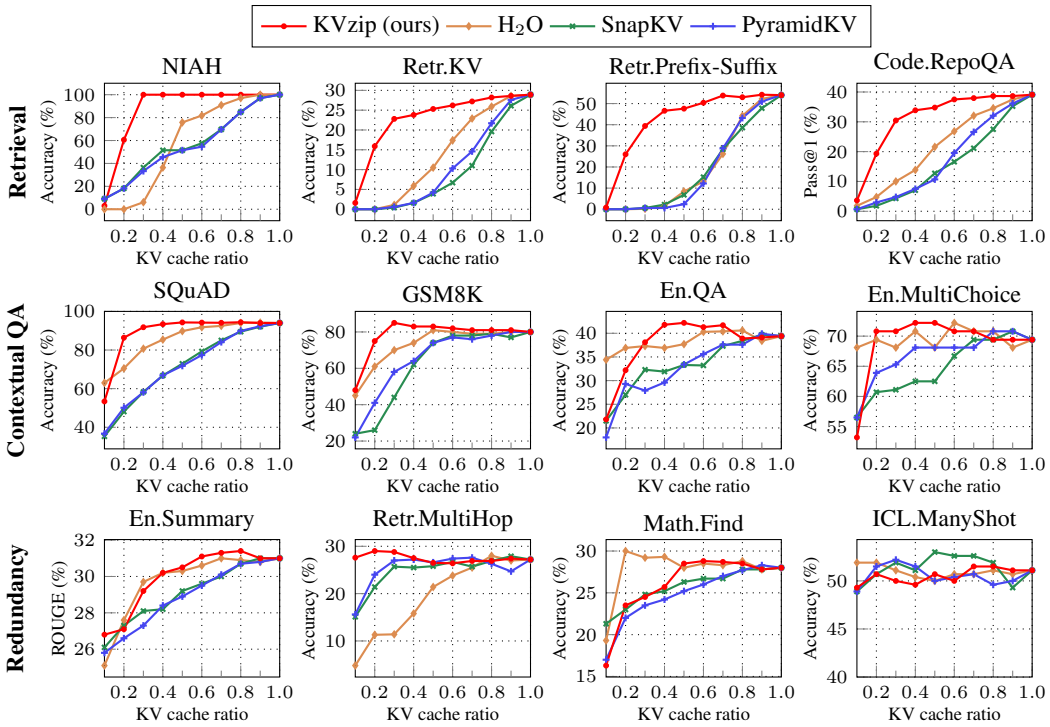

Figure 20: Benchmark results using Gemma3-12B [46] across compression ratios from 0.1 to 1.0.

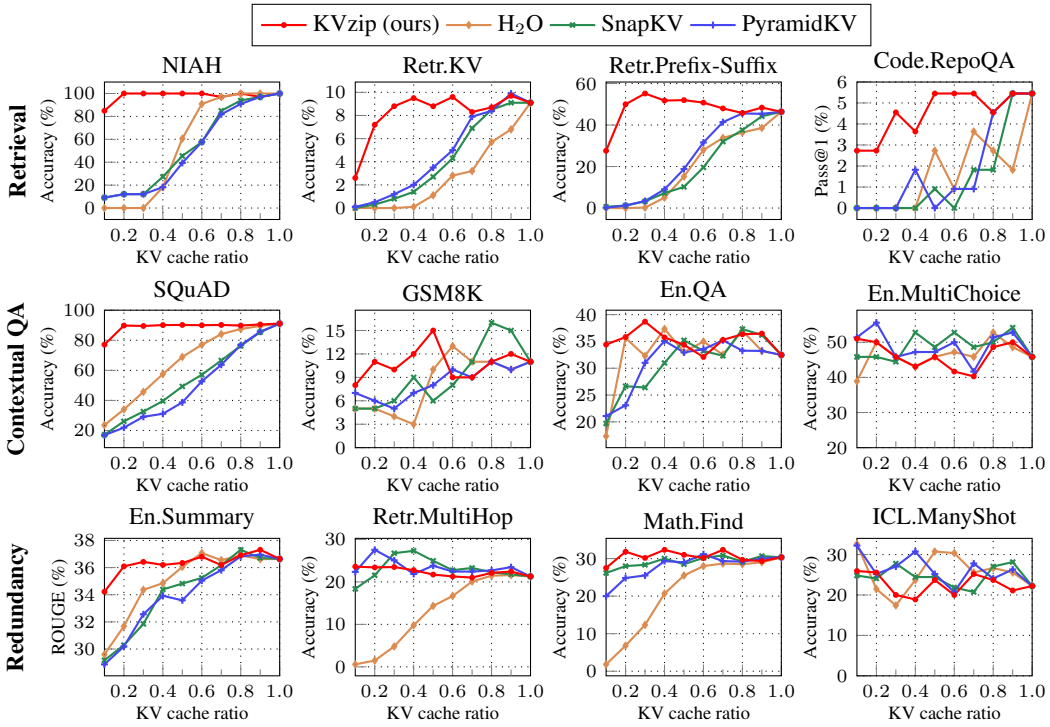

Figure 21: Benchmark results using LLaMA3-8B-W8A8KV4 [33] across compression ratios from 0.1 to 1.0.

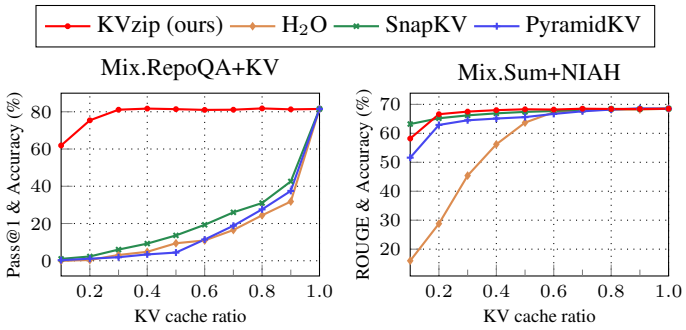

Figure 22: Benchmark results on SCBench multi-task datasets using Qwen2.5-7B [51] across compression ratios from 0.1 to 1.0.

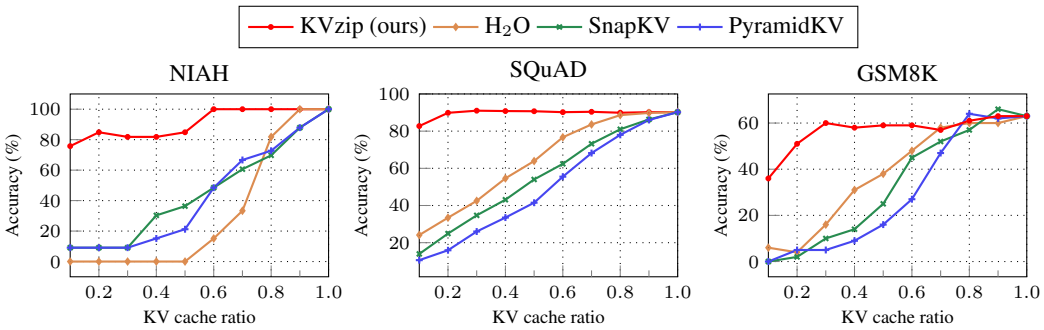

Figure 23: Benchmark results for LLaMA3.1-3B [19] across compression ratios ranging from 0.1 to 1.0. The evaluation focuses on shorter contexts, as LLaMA3.1-3B lacks the capability to solve SCBench tasks, resulting in near-zero accuracy.