MiniKV: Pushing the Limits of 2-Bit KV Cache via Compression and System Co-Design for Efficient Long Context Inference

Anonymous ACL submission

Abstract

State-of-the-art 2-bit KV cache quantization techniques achieve excellent results in accelerating LLM inference while retaining accuracy on long context tasks. However, further pushing the compression ratio fails to deliver performance gains. In this work, we revisit these approaches by considering, additionally, adaptive KV methods that retain LLM accuracy with only a subset of KV states. This leads us to propose a method based on 2-bit KV cache quantization with adaptive KV policies. In addition, we take an algorithm and system co-design approach by developing hardware-friendly kernels to accelerate LLM inference while making MiniKV compatible with existing memory-efficient attention techniques such as FlashAttention, effectively translating algorithmic improvements into system performance gains. Experiments on a wide range of long context tasks show that MiniKV effectively achieves >80% KV cache compression while retaining accuracy, outperforming state-of-the-art methods while achieving excellent latency, throughput, and memory consumption improvements in long context inference.

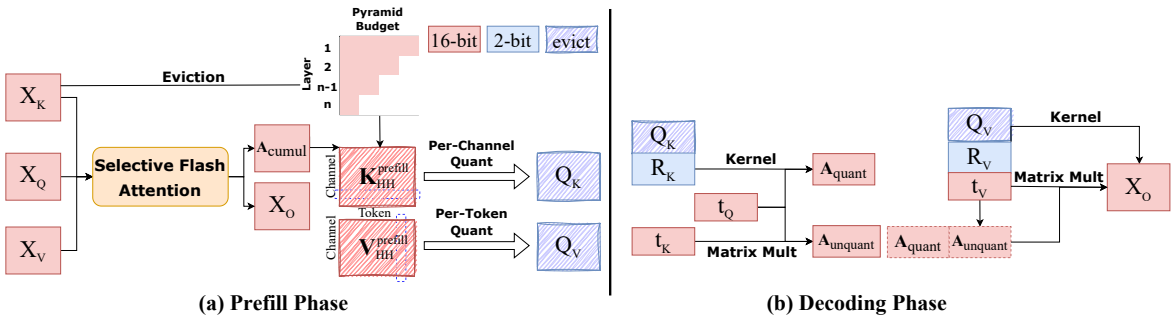
1 Introduction

Large language models (LLMs) have exhibited unique capabilities, such as instruction following, reasoning, and inference time scaling (OpenAI, 2024; DeepSeek-AI et al., 2025). However, efficiently serving LLMs is still a pressing concern. One of the main LLM inference bottlenecks is the consumption of KV cache memory, which consumes memory in addition to widely studied bottlenecks such as model sizes (Frantar et al., 2022; Lin et al., 2024).

To address this challenge, one of the prevailing practices is to quantize the KV cache (vLLM, 2025; NVidia, 2025). Studies show that FP8/INT8 or even 4-bit quantization can be achieved for KV cache compression while preserving accuracy (Sheng et al., 2023; Liu et al., 2023a; Yang et al., 2024b). State-of-the-art approaches, such as KIVI and KVQuant (Liu et al., 2024b; Hooper et al., 2024), show that the KV cache can be effectively quantized to sub 4-bit, e.g., 2 bits, while preserving most accuracy. However, further pushing down the compression ratio (e.g.,<2 bits) leads to a significant accuracy loss.

In a separate line of research in the community, numerous work have explored *adaptive KV*, where the LLM selects a small subset of KV states based on their importance (Zhang et al., 2023; Xiao et al., 2023b; Ge et al., 2023; Liu et al., 2023b). Recent advances also introduce head-specific adaptive KV (Ge et al., 2023; Xiao et al., 2024; Wu et al., 2024) and layer-specific adaptive KV (Cai et al., 2024; Nawrot et al., 2024; Wan et al., 2024) with the goal of evicting or merging KV pairs without compromising overall performance. However, following the work of (Zhang et al., 2023), few studies have included studies on how adaptive KV policies work on quantized KV cache, despite quantized KV is widely used in practice (Turganbay, 2024). Moreover, for long context inference, where KV cache memory becomes the major bottleneck, few adaptive KV work manage to achieve a compression ratio that exceeds 50% while maintaining accuracy in long context tasks (Li et al., 2024; Tang et al., 2024).

These two points of view (quantized KV and adaptive KV) consider the extreme sides of the spectrum of optimization points for KV cache memory. However, there has been very little work exploring how to consolidate these two lines of work to maximize the KV cache memory savings. The conventional wisdom is that these techniques can be *combined*. However, existing work aiming to combine 4-bit quantization and adaptive KV shows that combining these techniques leads to non-trivial interactions (Zhang et al., 2024), which



(a) Prefill Phase (b) Decoding Phase

Figure 1: An overview of MiniKV. Tensors colored red/blue indicate 16-bit/2-bit representation, and shaded tokens are evicted during inference. During the prefill phase, we employ pyramid KV with rectified token selection policy across layers to identify a sparse set of important tokens. For all the important tokens, we employ sub-channel Key quantization and per-token Value quantization to minimize the quantization errors while maintaining a compact KV cache data layout without introducing any irregular operations. To address the incompatibility issue between score-based KV pair selection policies and memory-efficient system optimizations such as FlashAttention, we develop a two-pass Triton-based *selective flash-attention kernel* to output both the representation X_O and the cumulative attention map A_{cumul}, while still keeping the memory consumption of the attention calculation linear with respect to the sequence length. During decoding, we use a *fused unpacking and multiplication* kernel to compute both the attention map between the new Query token t_Q and the quantized Keys, as well as the product between the attention map and the quantized Values.

need to be reasoned through carefully for good performance. In this paper, we address the following question: *How should 2-bit KV cache quantization techniques be combined with adaptive KV policies to maximize the inference speed of LLMs given a memory budget while retaining high model accuracy in long context inference?*

To answer the question, we revisit existing approaches on ultra low-bit quantized KV (e.g., 2-bit) and adaptive KV, together with a compression system co-design perspective, which is unexplored so far. Our findings led us to develop **MiniKV**, which effectively compresses the KV cache through a synergistic combination of 2-bit quantization and adaptive KV to achieve minimal accuracy loss in long-context tasks while maximizing the compression ratio. Specifically, on the algorithm side, we employ subchannel-wise key and token-wise value quantization, as well as pyramid KV with rectified token selection policy across layers to significantly push the KV compression ratio while keeping the algorithm still hardware-friendly without introducing any irregular computation. On the system side, we develop a two-pass Triton (Tillet et al., 2019) kernel together with native fused kernels to accelerate the inference latency while resolving the incompatibility limitation from the attention score-based eviction policy and memory-efficient attention system optimizations such as FlashAttention (Dao et al., 2022). Consequently, the resulting system maximizes the compression ratio on the KV cache while obtaining high accuracy and hardware efficiency in long context inference.

To validate the approach, we compare MiniKV with existing KV cache compression techniques such as H2O, SnapKV, and Q-Hitter, across three major models in LongBench datasets. The results show that MiniKV effectively achieves 86% KV cache compression while retaining comparable accuracy on LongBench, outperforming state-of-the-art methods. Furthermore, MiniKV enables prompt lengths up to 44K tokens and a maximum throughput that is 48% higher than its strongest baseline on a single NVIDIA A100 GPU. To our knowledge, our work is the first to show that it is possible to achieve significantly >50% KV cache reduction through compression and system co-design while achieving high batch size ≥ 1 throughput on long context tasks.

2 Related Work

Numerous efforts have been made to improve the KV cache efficiency of LLMs. Among them, quantization has been a prevailing technique employed in deployment to overcome KV memory overhead without retraining (vLLM, 2025; NVidia, 2025). Many research has shown that FP8/INT8/INT4 quantization can be achieved for KV cache while preserving accuracy (Hooper et al., 2024; Sheng et al., 2023; Liu et al., 2023a; Yang et al., 2024b; Zhang et al., 2024). However, further pushing the quantization limit to under 4-bit, e.g., 2-bit, leads to major performance loss. More recently, researchers have proposed advanced quantization techniques,

such as KIVI (Liu et al., 2024b), to quantize KV cache into 2-bit without major loss in accuracy. While being effective, it still has one major limitation: its effectiveness against adaptive KV policies and its implication on system performance has not yet been studied. Our results indicate that it is nontrivial to use 2-bit quantized KV together with adaptive KV policies in conjunction while achieving high compression ratio, accuracy, and system efficiency in long context inference, simultaneously.

Adaptive KV policies have also gained interest within the community, leading to various algorithms (Zhang et al., 2023; Xiao et al., 2023b; Liu et al., 2023b; Ge et al., 2023; Wan et al., 2024; Wu et al., 2024; Cai et al., 2024; Yang et al., 2024a; Li et al., 2024; Liu et al., 2024a; Brandon et al., 2024; Tang et al., 2024). However, many of those works either do not focus on long context inference (Zhang et al., 2023; Xiao et al., 2023b; Liu et al., 2023b; Ge et al., 2023), where the KV cache pressure is the most prominent, or introduce irregular operations or auxiliary scores that are not hardware-friendly (e.g., batch size >1 with FlashAttention enabled) (Ge et al., 2023; Wu et al., 2024; Wan et al., 2024; Tang et al., 2024). Finally, most adaptive KV methods struggle to exceed a 50% compression rate in long context inference (Zhang et al., 2023; Li et al., 2024; Tang et al., 2024), suggesting that solely identifying important tokens may have limited improvements for adaptive KV. Complementary to this line of work, our goal is to improve the compression ratio of KV cache via revising ultra-low quantized KV (e.g., 2-bit) with adaptive KV policies, with an eye toward system co-design to maximize the performance of LLM inference. We empirically show that this path can be more memory-efficient, especially on long context tasks. We provide a detailed summary of the comparison between MiniKV and previous approaches in Appendix B.

3 Method

In this section, we first focus on the compressibility of ultra low-bit quantized KV cache by considering adaptive KV policies, with an eye toward being able to still keep the final solution hardware friendly, which leads to the proposed algorithm in MiniKV. In addition, we introduce kernel optimization that addresses the composibility issue between score-based adaptive KV and memory-efficient attention implementation such as FlashAtttention.

3.1 Revisiting 2-Bit Quantized KV with Adaptive KV Policies

3.1.1 Sub-channel Key Quantization with Persistent Context Selection

Existing KV cache quantization methods often perform per-token quantization (i.e., the scaling factor and zero point are shared by elements in the same token) (Sheng et al., 2023; Xiao et al., 2023a). However, it has been observed that outliers emerge within the channel dimension of key cache (Liu et al., 2024b; Hooper et al., 2024), requiring channel-wise quantization.

Recent works (Hooper et al., 2024; Liu et al., 2024b) observe that the data distribution within each channel shifts over generation steps, leading to inaccurate quantization. We measure and confirm the accuracy impact of inaccurate quantization on LongBench in Appendix E. To mitigate quantization error, prior work suggests fine-grained per-channel key quantization, which quantizes keys at the granularity of a small sub-channel group (e.g. 16/32 numbers). Combining these techniques with a full KV cache is straightforward because the elements within each sub-channel group remain the same during the entire LLM generation process.

However, with adaptive KV, the elements within a sub-channel group may change after each decoding step if some tokens in the group are evicted to make space for newly generated tokens. MiniKV solves this problem by enabling sub-channel key quantization via *persistent context selection*. Our design for this optimization is based on the following key observation: *Given a sufficiently large cache budget, the important tokens can be identified before generation and maintained persistently throughout the process.*

We found some recent inference optimization works that argue against persistent context selection (Tang et al., 2024). However, all of these texts show that heavy hitters do not persist when using a tiny cache budget (<5%). We empirically verify persistent context selection by measuring the fraction of heavy hitters that persist during the entire generation phase of H_2O when using a large cache budget. We observe that nearly 60-80% of the heavy hitters selected during the prefill stage persist throughout generation. Please see Appendix D for details.

Based on this observation, we choose a set of

persistent heavy hitters at the end of the prefill to quantize and not update throughout the generation phase. This allows MiniKV to avoid re-encoding a group while keeping a low quantization error with 2-bit sub-channel quantization.

3.1.2 Selectivity in Long Contexts: Heavy Hitters vs. Recent Window

Prior studies observe that the accumulated attention scores of all tokens within an attention block follow a power-law distribution and claim that maintaining a tiny subset of important tokens (e.g., as low as 5%) with the highest accumulated attention score is sufficient to maintain precision (Zhang et al., 2023). However, this observation has not been carefully examined in long contexts.

We observe that using a highly limited memory budget (e.g., 20%), existing solutions such as H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024) have a significant performance drop in long context tasks, which motivates us to revisit the selectivity of adaptive KV methods. First, we assess if the model retains performance using only the recent window (RW) or heavy hitters (HH), we examine the KV cache's selectivity towards RW/HH. The cache budget is described as the percent of the prompt tokens retained, i.e. an RW/HH budget of $(\alpha_{RW}, \alpha_{HH})$ and an input prompt of length l_{prompt} tokens indicate that $(\alpha_{RW} \cdot l_{\text{prompt}}, \alpha_{HH} \cdot l_{\text{prompt}})$ tokens are maintained as the RW and HH respectively. We fix the total cache budget to 50% and distribute it among the RW and HH, i.e., RW/HH budget of $(\alpha_{RW}, \alpha_{HH}) = (0\%, 50\%), (10\%, 40\%), (20\%, 30\%)$, and so on. Fig. 2 (left) reveals an interesting aspect of the KV cache selectivity: The model performs better on some datasets with more HH (on Passage Count) and on some with a longer RW (on TriviaQA). More importantly, using solely RW or HH leads to a catastrophic accuracy drop in certain tasks (in Lcc and TriviaQA). This indicates that to have a robustly optimized KV cache selection policy, the model needs to maintain at least a critical percentage of HH/RW (e.g., 5-10%) to avoid a significant accuracy drop.

Next, we investigate the selectivity between RW and HH by varying the KV cache budget. In particular, we fix the RW size (e.g., 10% of the prompt length) while varying the HH set size, and vice versa. Interestingly, as seen in Fig. 2 (right), we observe that there appears to be *no common trend* across datasets as to whether increasing the size of

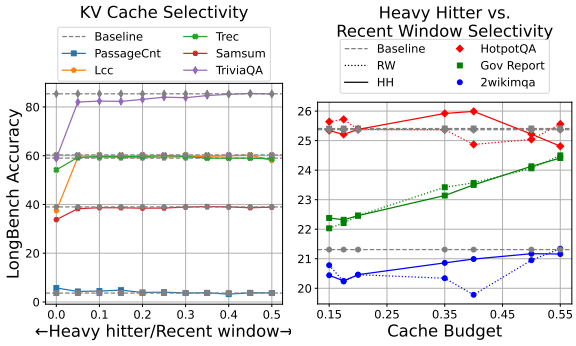


Figure 2: (Left) H_2O with different recent window/heavy hitter budget: We fix the total cache budget to 50% and vary the heavy hitter and recent window budget. (Right) H_2O with different recent window/heavy hitter budget: The heavy hitter/recent window cache budget is fixed at 10% and the recent window/heavy hitter budget is increased from 5% to 45%. The dotted/-solid lines indicate variable recent window/heavy hitter budget.

the RW vs. the HH set significantly improves the selectivity of KV states on long context tasks. In fact, either HH or RW allow adaptive KV to achieve accuracy comparable to the full KV cache baseline. Furthermore, unlike previous findings, which suggest that high levels of eviction (80-95%) do not decrease model accuracy (Zhang et al., 2023), we find that as the sequence length increases, maintaining accuracy under the same KV cache size budget becomes challenging (please see Appendix K for more details). However, low and medium levels of eviction (e.g., 50%) are still possible.

Insight. Our experiments suggest that high levels of KV cache eviction significantly degrade LLM's performance on long context tasks. However, medium levels of eviction can still retain comparable model accuracy. Even at medium levels, the model needs to maintain a critical percentage of both heavy hitters and recent window tokens.

3.1.3 Layer-Specific Selectivity: Uniform, Variance, or Pyramid?

Inspired by recent works on layer-wise KV cache compression (Cai et al., 2024; Liu et al., 2024a; Wan et al., 2024), we investigate several layer-specific KV cache selection strategies that allocate variable KV cache budgets across model layers.

- **Uniform allocation**: This policy has been used in multiple previous studies (Zhang et al., 2023; Xiao et al., 2023b; Liu et al., 2023b), where all layers have the same KV cache budget.
- **Variance-based allocation**: Similar to (Wan et al., 2024), we use the variance of the cu-

mulative attention map to determine the layer-wise KV cache budget. Lower layers exhibit smaller variances, making token eviction difficult. We examine two policy variations: *Var-prop*, allocating KV cache per layer proportionally to variance, and *Var-inv*, allocating it inversely proportional to variance.

- **Pyramid-like allocation**: Introduced in (Cai et al., 2024), this strategy adjusts the heavy hitter cache budget across layers by allocating more cache in lower layers and less in higher ones. The cache budget for the intermediate layers is determined through linear interpolation.

In our experiments, we observe that the *Pyramid* policy achieves much better accuracy than the other policies, especially with medium levels of eviction, shown in Fig. 3.

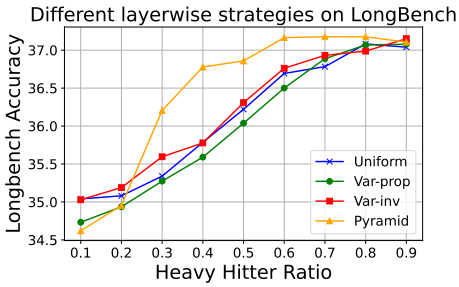


Figure 3: Performance of layer-wise KV cache allocation policies. The *Pyramid* policy works best, particularly at medium levels of eviction.

3.2 Memory-Efficient Fused Selective Attention Kernels

Despite ongoing advancements, many adaptive KV studies predominantly use attention scores as a criterion to determine which tokens should be evicted (Zhang et al., 2023; Liu et al., 2023b; Leskovec and Sosic, 2016; Cai et al., 2024).

While showing promising results in reducing the KV cache size, these *attention-score-driven* methods are not aligned with memory-efficient transformer system optimizations. In particular, these methods rely on accessing the attention matrix A in Equation 1, which has a quadratic memory complexity with respect to the sequence length. FlashAttention (Dao et al., 2022) performs the attention process without materializing the attention matrix. Therefore, to the best of our knowledge, no prior adaptive KV works with FlashAttention enabled, hindering their memory savings on long sequences.

To address the challenge, this part introduces our memory-efficient Triton kernel implementation for MiniKV, which simultaneously returns the following two outputs with linear memory complexity: (1) a weighted sum of the value tensors X_O, same as FlashAttention, and (2) cumulative attention score A_{cumul} along each column [1]. Despite being a simple task when memory is not a constraint, implementing such a kernel with linear memory complexity is challenging. The difficulty arises because A_{cumul} requires summing the attention values for each token position, i.e., along the columns of the attention matrix. FlashAttention reduces memory usage by employing row-wise tiling, which avoids storing large intermediate attention matrices. However, this row-wise tiling means that different rows are processed in parallel, leading to a race condition when summing the attention scores column-wise. To prevent this race condition, atomic add instructions are needed, which significantly slow down the kernel execution speed.

We solve this by introducing a two-pass kernel implementation. In the first-pass of the kernel, we follow FlashAttention to compute the weighted sum of the value tensors and save the intermediate LSE (Log Sum Exponential) value. To efficiently operate on data in shared memory, we tile the input query tensor into row blocks of size KBlockM. Within each row block, the key tensor is subdivided into tile blocks of size KBlockN. Each row and column block calculates the tiled attention map $P^{KBlockM \times KBlockN}$. With this product of the query and key tensors and the corresponding tile from the value tensor, we follow FlashAttention's online softmax reduction to compute the weighted V block write it back. We aggregate the LSE value per row into an additional buffer of size [batchSize, headDim, seqLen].

For the second-pass, we run different columns in parallel to compute a sequential sum of attention weights per token. As shown in Fig. 4, we iteratively recompute the QK^T value and use the LSE values to normalize it. From top to bottom, we accumulate the sum column-wise and save it to the corresponding position in A_{cumul}.

In summary, any memory buffers that we allocate over FlashAttention scale linearly with sequence length, i.e., LSE requires $O(l_{\text{query}})$ and A_{cumul} requires $O(l_{\text{key}})$ memory.

[1] Variables marked in Red/Blue indicate tensors in FP16/INT2 precision.

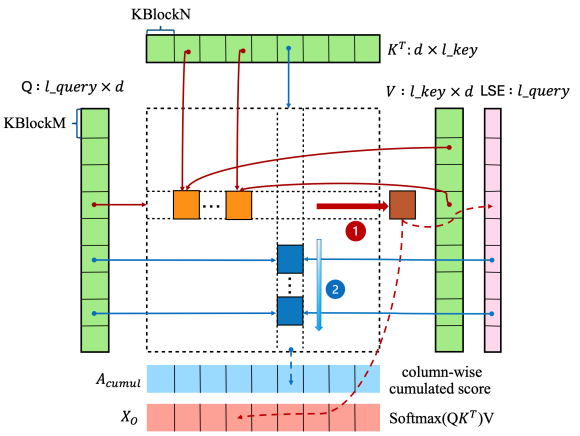


Figure 4: Two-pass kernel parallelism: In the first pass, we choose different row blocks running in parallel to compute the weighted sum of value tensors. At the same time, each row updates its max and sum and saves it to LSE. Then it switches to processing column blocks in parallel during the second pass. For each column, it recomputes QK^T and normalizes it with the corresponding LSE value. From top to bottom, each column accumulates the sum and writes the result to A_{cumul}.

3.3 MiniKV Algorithm

Based on the aforementioned observations and optimizations, we employ compression and system co-design for MiniKV, as shown in Algorithm 1. In the prefill stage, MiniKV uses the fused selective flash-attention kernel (§ 3.2) to obtain aggregated attention scores A_{cumul}. Based on the attention score, MiniKV selects the subset of KV states that has the highest attention score at the end of the prefill stage (denoted as $K_{HH}^{\text{prefill}}, V_{HH}^{\text{prefill}}$). The tokens retained are compressed to INT2 representations. We use a separate high-performance compression kernel provided by (Liu et al., 2024b) to apply bit shift to pack 16 INT2 scalar values from selected KV states into an INT32 tensor. The key/value tokens are quantized along the channel/token dimension. The results at the end of the prefill phase are the quantized key/value representation (Q_K, Q_V, stored in packed INT32 tensors) and the quantization zero-point and scale (stored in FP16 tensors).

During each decoding step, MiniKV dequantizes the quantized KV cache ($\mathbf{q}^{-1}(Q_K, Q_V)$) and uses the dequantized key states along with the new key and query token (t_K, t_Q) for attention calculation. Once the attention map (A) is obtained, the dequantized values states ($\mathbf{q}^{-1}(Q_V)$) and the new value token (t_V) are multiplied by (A) to compute the output of the attention layer (t_O). MiniKV fuses the dequantization operations with subsequent matrix multiplications to reduce kernel launch overhead and global memory accesses, leading to latency reduction.

Algorithm 1 The MiniKV Algorithm, FP16/INT2

Require: Input $X_P \in \mathbb{R}^{l_{\text{prompt}} \times d}$
$X_Q, X_K, X_V \leftarrow X_P W_Q, X_P W_K, X_P W_V$
$X_O, A_{\text{cumul}} = \text{Selective_flash_attn}(X_Q, X_K, X_V)$
$K_{HH}^{\text{prefill}}, V_{HH}^{\text{prefill}}, \#_{HH} \leftarrow \text{Heavy_hitters}(A_{\text{cumul}})$
$Q_K, Q_V \leftarrow \text{Quant}(K_{HH}^{\text{prefill}}), \text{Quant}(V_{HH}^{\text{prefill}})$
KV Cache $\leftarrow Q_K, Q_V$
procedure DECODING(KV cache, token $t \in \mathbb{R}^{1 \times d}$)
  $t_Q, t_K, t_V \leftarrow tW_Q, tW_K, tW_V$
  $Q_K, Q_V, R_K, R_V \leftarrow$ KV cache
  $R_K, R_V \leftarrow \text{Concat}([R_K, t_K]), \text{Concat}([R_V, t_V])$
  if $\text{len}(R_K) = n_r$ **then**
    $Q'_K, Q'_V \leftarrow \text{Quant}(R_K), \text{Quant}(R_V)$
    $Q_K \leftarrow \text{Concat}([Q_K, Q'_K], \text{dim} = \text{channel})$
    $Q_V \leftarrow \text{Concat}([Q_V, Q'_V], \text{dim} = \text{token})$
    $R_K, R_V \leftarrow$ None
  end if
  $A \leftarrow \text{Softmax}(\text{Concat}([\mathbf{q}^{-1}(Q_K)t_Q^T, R_K t_Q^T]))$
  $A_{\text{quant}}, A_{\text{unquant}} \leftarrow A[: -\text{len}(R_K)], A[-\text{len}(R_K) :]$
  $t_O \leftarrow A_{\text{quant}}\mathbf{q}^{-1}(Q_V) + A_{\text{unquant}} R_V$
  KV Cache $\leftarrow Q_K, Q_V, R_K, R_V$
  return t_O
end procedure

Inspired by KIVI (Liu et al., 2024b), we use a streaming buffer for both key and value states during the decoding stage, so that newly generated key/value caches are first stored in FP16 (indicated by (R_K, R_V)). These tokens are compressed every n_r step. This saves repeated kernel launch overhead for quantization while maintaining at most n_r KV tokens in FP16 during generation.

4 Experiments

We conduct experiments to evaluate the effectiveness of MiniKV in improving accuracy preserving and inference performance.

4.1 Evaluation Methodology

Models. We compare MiniKV against state-of-the-art public LLMs, including LLaMA2-7B-chat, LLaMA2-13B-chat (Touvron et al., 2023) and Mistral-7B-Instruct-v0.2 (Jiang et al., 2023).

Datasets. We choose LongBench for evaluation (Bai et al., 2023), which has been widely adopted in state-of-the-art works (Liu et al., 2024b; Hooper et al., 2024; Li et al., 2024). Additional details on the data sets used can be found in the Appendix F.

Baselines. We compare MiniKV with the following baselines: adaptive KV (H_2O, SnapKV (Zhang et al., 2023; Li et al., 2024)), INT2 quantized KV (KIVI (Liu et al., 2024b)), adaptive + quantized KV (Q-hitter (Zhang et al., 2024)), and FullKV. Q-Hitter combines H_2O with INT4 quantization, providing a strong baseline for MiniKV.

Table 1: Performance evaluation of MiniKV on various models in a range of benchmarks in LongBench. Rows marked in brown have a similar KV cache size, while KIVI and the full model use a larger KV cache.

Models	Methods	Single-Doc QA		Synthetic		Code		Multi-Doc QA		Summarization		Few-Shot Learning			Average
		Qasper	MultifieldQA	Passage Ret.	Passage Ct.	LCC	RepoBench-P	2WikiMQA	HotpotQA	Gov Report	Multi News	TREC	SamSum	TriviaQA	
LLaMA2-7B-chat	Full Model	22.78	33.59	8.44	4.75	59.56	48.07	22.35	24.88	24.99	23.60	59.67	39.38	85.38	35.19
	KIVI	22.45	33.32	11.33	4.25	59.05	47.96	21.88	23.88	24.46	22.86	59.67	38.74	84.80	34.97
	H_2O (15%)	16.98	29.72	11.00	4.55	56.87	48.25	19.92	24.58	22.19	22.16	57.33	37.80	84.02	33.49
	SnapKV (15%)	17.41	34.53	8.67	3.59	58.48	47.52	21.00	24.91	19.04	19.74	59.33	37.92	84.72	33.60
	Q-Hitter (59%)	17.43	30.08	9.00	4.13	56.84	45.18	17.66	22.57	22.83	22.48	59.67	38.46	82.76	33.01
	MiniKV	21.01	29.23	10.00	3.82	58.38	47.99	20.91	22.97	23.45	22.54	59.00	37.94	80.95	33.71
	MiniKV Pyramid	19.92	33.96	10.00	4.12	59.72	49.29	20.69	24.62	24.16	22.90	59.00	39.15	82.89	34.65
LLaMA2-13B-chat	Full Model	13.72	28.11	20.67	5.58	49.97	47.18	12.13	15.14	26.29	23.52	64.00	40.39	86.52	33.32
	KIVI	13.56	28.16	17.33	5.05	49.21	47.18	12.80	15.27	25.24	23.07	64.33	40.24	87.07	32.96
	H_2O (15%)	11.94	25.13	15.67	4.61	48.18	44.29	13.04	14.52	23.15	22.12	59.67	39.66	83.70	31.2
	SnapKV (15%)	12.11	27.09	22.00	5.18	49.52	45.44	14.10	14.40	20.06	20.75	62.33	39.25	85.86	32.16
	MiniKV	11.24	25.13	15.00	3.62	48.43	46.10	12.74	16.16	24.26	22.84	63.33	40.79	84.33	31.84
	MiniKV Pyramid	12.79	27.32	17.00	2.79	48.94	46.25	12.66	15.47	25.06	23.14	63.67	40.35	85.33	32.37
Mistral7B-instruct	Full Model	25.79	47.97	50.83	2.98	50.69	47.22	27.44	36.44	31.84	25.82	62.67	40.49	86.29	41.2
	KIVI	25.13	46.30	50.75	3.02	51.16	46.81	26.39	35.11	31.23	25.36	62.33	40.12	86.31	40.77
	H_2O (15%)	20.20	42.55	42.84	3.00	49.66	45.95	24.27	33.04	27.43	24.33	60.33	40.45	86.20	38.4
	SnapKV (15%)	24.14	48.32	50.23	3.04	50.39	45.76	25.76	34.55	25.10	22.77	61.67	40.12	86.90	39.90
	MiniKV	22.94	45.80	49.47	3.36	49.78	45.56	24.27	33.84	29.73	25.22	61.67	39.96	86.36	39.84
	MiniKV Pyramid	23.10	45.91	48.88	3.24	50.34	45.41	25.18	34.04	29.69	25.32	61.67	40.17	86.63	39.97

Hyperparameters. We use a 50% cache budget with MiniKV, with 25% heavy hitter budget and 25% the recent window budget. The group size during token/channel-wise quantization is set to 16, i.e. 16 values along the token/channel axis share quantization zero point and scale. A residual length of $n_r = 128$ is used for both MiniKV and KIVI. The maximum prompt length is 4096 for all models with the first and last 2048 tokens taken for a longer prompt. The maximum generation length is dataset-specific. No task has a generation length of more than 512 tokens. Please see Appendix G for other evaluation details.

Hardware. We conducted experiments on NVIDIA 4×A100-40GB and 4×A40-46GB GPUs.

4.2 Enhancing KV Cache Compression Accuracy in Long Context Inference

To make a fair comparison, we compare all methods with adaptive KV policies (H_2O, SnapKV, Q-Hitter, and MiniKV) under a similar KV cache size (Appendix H). Given a prompt length of 4096 and generation length of 512, the KV cache size for MiniKV is 0.33 GB. A cache budget of $\alpha = 15\%$ results in a similar KV cache size for H_2O. A cache budget of $\alpha = 59\%$ results in a similar KV cache size for Q-Hitter. We test two strategies of MiniKV, namely MiniKV and MiniKV-Pyramid, to demonstrate the effectiveness of MiniKV. MiniKV follows a uniform cache allocation with $(25\%, 25\%)$ HH, RW budget per layer. MiniKV-Pyramid uses 25% RW budget per layer but the HH budget is distributed across layers as described in § 3.1.3.

The results are shown in Table 1. MiniKV outperforms other state-of-the-art adaptive KV methods (H_2O, SnapKV, Q-Hitter) for the same KV cache size. For LLaMA2-7B-chat, MiniKV-Pyramid achieves an average accuracy of 34.65, obtaining 98.5% of the full model accuracy 35.19. MiniKV is also able to maintain accuracy on LLaMA2-13B-chat and Mistral-7B, indicating that our approach generalizes well across datasets and model classes. While the full model and KIVI perform marginally better than MiniKV, they have much larger KV cache memory consumption. The synergistic composition of 2-bit quantized KV and layer-wise adaptive KV delivers these improvements, and it also shows the promising aspect of using both quantization and adaptive KV in conjunction to reduce the high memory footprint of the KV cache.

4.3 Setting A New Pareto Frontier

With H_2O, SnapKV, Q-Hitter, and MiniKV the user can tune the cache budget, potentially improving performance at the cost of a larger KV cache. An ideal technique would maintain performance when lowering the cache budget. We plot the performance of MiniKV against the KV cache size. The size of the KV cache is computed using the KV memory consumption analysis in Appendix H. To highlight interesting configurations, we mark the Pareto optimal front, which is the configuration that offers the smallest KV cache size for the highest performance.

Fig. 5 shows the performance vs KV cache size curve for two datasets (Qasper and Lcc), the remaining plots can be found in the Appendix I. MiniKV achieves the pareto optimal compression strategy across all 6 major task categories on LongBench (single/multi-doc QA, LC understanding, code completion, summarization and few-shot

learning). These results validate the effectiveness of MiniKV with varying KV cache sizes.

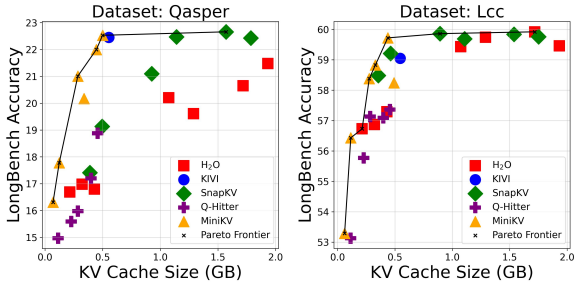


Figure 5: Algorithm Performance vs KV Cache Size: The Pareto frontier (the black curve) indicates the optimal compression strategy across a range of KV cache sizes. MiniKV lies on the Pareto frontier across all 6 task categories.

4.4 System Performance Results

We evaluate the system performance of the LLaMA2-7B-chat model on a single NVIDIA A100 GPU with 40GB of memory. We utilize FlashAttention kernels for KIVI and the Full Model while employing our customized kernel introduced in § 3.2 for MiniKV. H_2O and Q-Hitter do not support FlashAttention.

Speeding up end-to-end latency. LLM inference is predominantly constrained by the memory bandwidth required to retrieve the model states. MiniKV reduces latency through a compression and system co-design approach, which reduces the number of KV pairs loaded for each next-token prediction by revising 2-bit KV quantization combined with adaptive KV policies, while at the same time maintaining hardware friendly execution using high-performance memory-efficient kernels compatible with system optimizations such as FlashAttention. As a result, as shown in Fig. 6 (left), MiniKV has a lower latency than its baselines, especially in long sequences (e.g., >10k). We include a detailed latency breakdown analysis in Appendix J.

Achieving high throughput. As shown in Fig. 6 (right), MiniKV outperforms all its baselines in throughput, measured as the number of tokens processed per second, due to its lower latency and ability to support larger batch sizes and longer sequence lengths.

Effectively reducing peak memory usage. We benchmark peak memory usage, i.e., the maximum memory occupied by all model tensors during the generation. The memory savings achieved by KV cache compression can be rendered ineffective if peak memory usage exceeds the total GPU memory. We evaluate the impact of batch size and prompt length on peak memory usage in Fig. 7 (left). MiniKV demonstrates the lowest peak memory consumption compared to its baselines. H_2O goes out-of-memory at batch size 16 as it materializes the intermediate attention score matrix while KIVI maintains the full KV cache and therefore has a higher memory consumption.

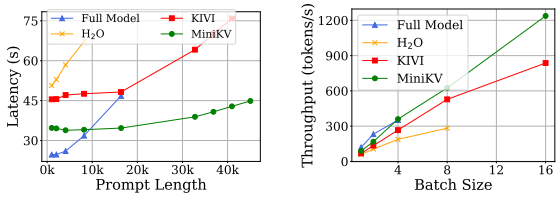


Figure 6: Left: Latency (s) for batch size = 1 and generation length = 1024. Right: Throughput (tokens/s) for prompt length = 2048 and generation length = 1024.

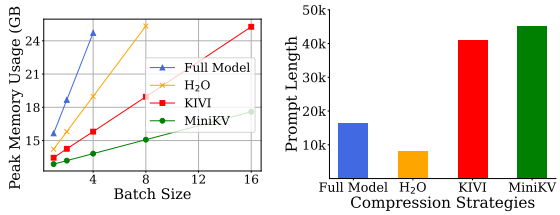


Figure 7: Left: Peak memory usage (GB) vs batch size for prompt = 2048 and generation length = 1024. Right: Maximum prompt length supported by MiniKV and its baselines for batch size = 1.

Enhancing maximum processable prompt. MiniKV's lower memory consumption becomes more apparent with longer prompt lengths. Fig. 7 (right) shows that MiniKV can process prompts 10% longer than its strongest baseline KIVI. Additionally, MiniKV's selective flash-attention kernel allows significantly longer sequence lengths when compared to H_2O.

5 Conclusion

In this work, we revisit KV cache optimization via compression and system co-design to accelerate the inference of LLM. Our empirical analysis indicates that it is challenging to directly compose state-of-the-art 2-bit quantized KV with existing adaptive KV policies while preserving both accuracy and system efficiency on long context tasks under a high compression ratio. To address this issue, we develop MiniKV to bridge the gap between ultra low-bit KV quantization and adaptive policies, as well as the gap between the compression algorithm and hardware. Evaluation on a wide range of datasets and models shows that MiniKV preserves long context accuracy while significantly improving the efficiency of LLM inference.

6 Limitations

MiniKV is promising in optimizing the KV cache. However, we identify several limitations and opportunities that can become future avenues of research to achieve an even higher compression ratio and generalizable compression.

1. **Combination with model optimizations.** While we mainly focus on KV cache optimization (which provides significant benefits on its own), MiniKV can also be combined with other optimization techniques, such as model compression (Frantar et al., 2022; Xiao et al., 2023a). This would further improve the computational and memory efficiency of LLMs.
2. **Extensible design.** While we use H_2O and KIVI as an example, our approach is compatible with other KV optimization techniques, such as StreamingLLM (Xiao et al., 2023b) and KVQuant (Hooper et al., 2024).
   Given that MiniKV combines H_2O and KIVI, we also explored the possibility of combining SnapKV and KIVI. This combination should be viable in theory, as it involves only changing the eviction strategy during the prefill phase. However, we find that doing so leads to a severe drop in performance, with LongBench scores dropping from 35 to 32 points. Further experiments show that the tokens retained by SnapKV tend to be more sensitive to 2-bit quantization than those selected by H_2O. This highlights the need for a more robust and generalizable approach to combining eviction and quantization, and a framework to determine when such combinations are effective.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *CoRR*, abs/2308.14508.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelly. 2024. Reducing transformer key-value cache size with cross-layer attention. *CoRR*, abs/2405.12981.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. Pyramidkv: Dynamic KV cache compression based on pyramidal information funneling. *CoRR*, abs/2406.02069.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Daya Guo DeepSeek-AI, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323.

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. *CoRR*, abs/2310.01801.

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. *CoRR*, abs/2401.18079.

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 preprint arXiv:2310.06825*.

Jure Leskovec and Rok Sosic. 2016. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM TIST*, 8(1):1:1–1:20.

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. *arXiv preprint arXiv:2404.14469*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024a. Minicache: Kv cache compression in depth dimension for large language models. *CoRR*, abs/2405.14366.

Liyuan Liu, Jialu Liu, and Jiawei Han. 2021. Multi-head or single-head? an empirical comparison for transformer training. *CoRR*, abs/2106.09650.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023a. LLM-QAT: data-free quantization aware training for large language models. *CoRR*, abs/2305.17888.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023b. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. *CoRR*, abs/2402.02750.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M. Ponti. 2024. Dynamic memory compression: Retrofitting llms for accelerated inference. *CoRR*, 2403.09636.

NVidia. 2025. Introducing New KV Cache Reuse Optimizations in NVIDIA TensorRT-LLM. https://tinyurl.com/4zbvwpcz. Accessed: 14-Feburary-2025.

OpenAI. 2024. Introducing OpenAI o1 . https://openai.com/o1/.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 31094–31116. PMLR.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*.

Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2019, Phoenix, AZ, USA, June 22, 2019*, pages 10–19. ACM.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Raushan Turganbay. 2024. Unlocking Longer Generation with Key-Value Cache Quantization. https://huggingface.co/blog/kv-cache-quantization. Accessed: 14-Feburary-2025.

vLLM. 2025. Quantized KV Cache. https://docs.vllm.ai/en/stable/features/quantization/quantized_kvcache.html. Accessed: 14-Feburary-2025.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5797–5808. Association for Computational Linguistics.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, and Mi Zhang. 2024. D2o: Dynamic discriminative operations for efficient generative inference of large language models. *CoRR*, abs/2406.13035.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. 2024. Retrieval head mechanistically explains long-context factuality. *CoRR*, abs/2404.15574.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023a. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. *CoRR*, abs/2410.10819.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023b. Efficient streaming language models with attention sinks. *CoRR*, abs/2309.17453.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024a. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *CoRR*, abs/2405.12532.

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

Zhenyu Zhang, Shiwei Liu, Runjin Chen, Bhavya Kailkhura, Beidi Chen, and Atlas Wang. 2024. Q-hitter: A better token oracle for efficient llm inference via sparse-quantized kv cache. *Proceedings of Machine Learning and Systems*, 6:381–394.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

A Formal Problem Formulation

We introduce a general formulation of the co-compression of the KV cache via quantization and selection. For a given LLM Φ with H layers, we denote its key states and value states at a layer h as $\mathcal{K}_h \in \mathbb{R}^{n\times d}$ and $\mathcal{V}_h \in \mathbb{R}^{n\times d}$, respectively. Let $Q_h \in \mathbb{R}^{1\times d}$ denote the query state. Then, the output $\mathcal{O}_h$ for each attention head of Φ is:

$$\mathcal{O}_h = \mathcal{A}_h\mathcal{V}_h,\ \mathcal{A}_h = softmax\left(\frac{Q_h\mathcal{K}_h^T}{\sqrt{d}}\right) \quad (1)$$

Then the co-compression problem can be formulated as:

Definition 2.1 (KV Cache Co-Compression Problem, informal).
$\forall\ \mathcal{K}_h$ and $\mathcal{V}_h$, where $h \in \{0, 1, .., H-1\}$, find the quantizer $\mathcal{Q}_b[\cdot]$ with b quantization bits, the selection policy $\mathcal{S}_h[\cdot]$ with C selective KV cache size, such that $|\mathcal{O}_h - \mathcal{O}_h^*| \leq \epsilon$, where $\mathcal{O}_h^*$ represents the output for each attention head of Φ with $\mathcal{S}_h[\cdot]$ and $\mathcal{Q}_b[\cdot]$, and ϵ is an acceptable small positive value.

B Comparison of MiniKV with Alternative Methods

We provide a detailed summary of the comparison between MiniKV and previous approaches in Table 2.

C Additional Results on Attention Distribution on Long-Context Understanding Tasks

Researchers have always been interested in exploiting the underlying structure of the attention mechanism to improve inference efficiency (Liu et al., 2021; Voita et al., 2019; Wu et al., 2024).

While prior studies show that attention scores are largely sparse (Zhang et al., 2023; Xiao et al., 2023b; Liu et al., 2023b), we observe that the attention distribution has more diverse patterns on long sequences. Fig. 8 shows that attention distribution of LLaMA2-7B-chat on a sample from the HotpotQA dataset.

We observe distinctive patterns: (i) the attention distribution at the lower layers has a wide coverage over sequence lengths and is more dispersed, and (ii) attention becomes more narrowly focused on a small subset of tokens and starts to exhibit block-wise sparse attention as the tokens move to the higher layers. We consistently observe this pattern across datasets in LongBench.

D Persistent Context Selection Analysis

We analyzed a sample prompt from the Lcc dataset to show that the heavy hitters selected in the pre-fill phase persist across generations Fig. 9. The green positions indicate that the 150 heavy hitters currently retained by the H_2O algorithm, while the white ones represent evicted tokens. It is evident that while different heads have different importance distributions, the important tokens largely do not vary across different generation steps.

E Token-Wise Quantization Of The KV Cache

A prevalent approach to compress the KV cache is by quantization. However, directly applying quantization to selective KV imposes challenges. Prior studies find that KV states contain outliers (Liu et al., 2023a; Xiao et al., 2023a), and per-token quantization is needed to avoid accuracy degradation. Fig. 10 shows that while applying INT8 and INT4 per-token quantization to both key and value caches helps maintain the accuracy of selective KV on LongBench, further reducing it to INT2 results in a significant accuracy drop, because 2-bits can not fully capture the dynamic range of KV token distributions. This motivates using channel-wise quantization as in KIVI (Liu et al., 2024b) and KVQuant (Hooper et al., 2024).

F Dataset Details

We seek a dataset that covers a broad range of long-context understanding tasks. For this reason, we choose LongBench, which covers six major task categories and in total 13 datasets (Bai et al., 2023): Qasper(F1) and MultiFieldQA(F1) are single doc QA tasks; Passage Retrieval(accuracy) and passage count(accuracy) are synthetic datasets to test the model's tendency to forgot information over a long context understanding; LCC(similarity) and RepoBench-P(similarity) are code completion tasks; 2WikiMultihopQA(F1) and HotpotQA(F1) are multi doc QA tasks; GovReport(Rouge) and MultiNews(Rouge) are summarization tasks; TREC(accuracy), SAMSum(Rouge) and TriviaQA(F1) are few-shot learning tasks.

G Evaluation Details

Decoding Strategy All models generate responses using deterministic greedy decoding across all tasks to ensure a fair comparison and reproducibility.

Table 2: Comparison with previous KV cache optimization methods for LLM inference.

Approach	Eviction-based KV	Quantization	Training-free	Long Bench
AttentionSink (Xiao et al., 2023b)	✓		✓	
FastGen (Ge et al., 2023)	✓		✓	
ScissorHands (Liu et al., 2023b)	✓	4-bit	✓	
H2O (Zhang et al., 2023)	✓	4-bit	✓	
FlexGen (Sheng et al., 2023)		4-bit	✓	
LLM-QAT (Liu et al., 2023a)		4-bit		
Q-Hitter (Zhang et al., 2024)	✓	4-bit	✓	
KVQuant (Hooper et al., 2024)		4-bit	✓	✓
KIVI (Liu et al., 2024b)		2-bit	✓	✓
MiniKV	✓	2-bit	✓	✓

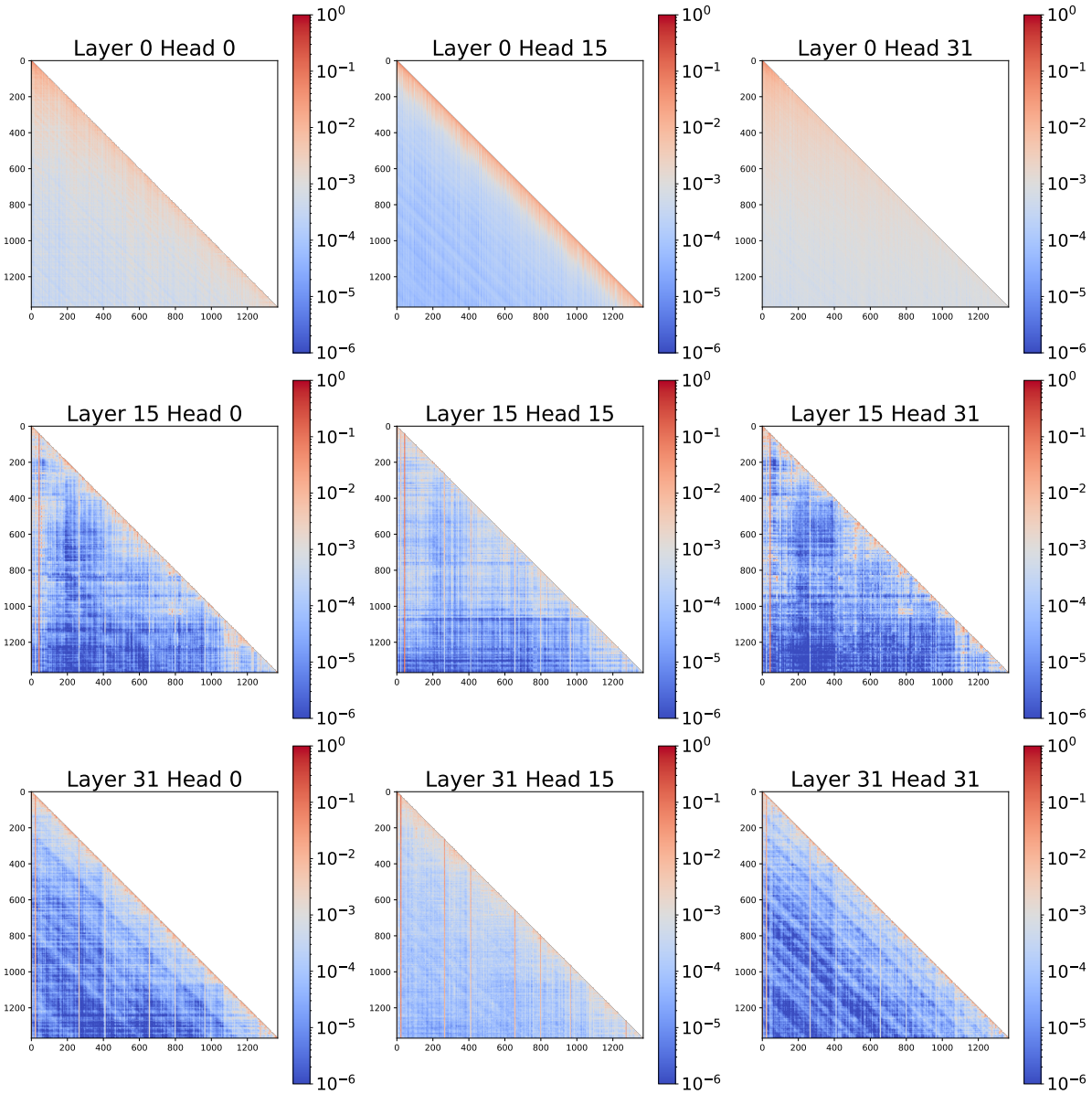


Figure 8: The attention distribution of LLaMA2-7B over the HotpotQA dataset in LongBench.

LongBench Truncation Strategy: we ensure that the model consistently selects the first 2000 and last 2000 tokens, regardless of changes to truncation settings or special tokens. This ensures stable score calculations across tests.

Pyramid-like Allocation Details Inspired by PyramidKV(Cai et al., 2024), we adjust the heavy hitter cache budget across layers by allocating more

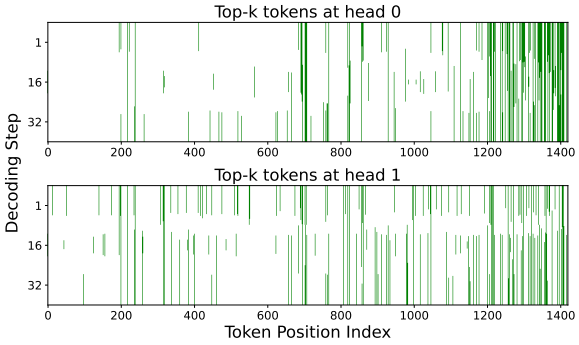


Figure 9: Top-k tokens with the highest cumulative attention score on the Lcc dataset from LongBench. Green tokens mark the heavy hitters retained by the H_2O algorithm. Here, we choose $k = 150$.

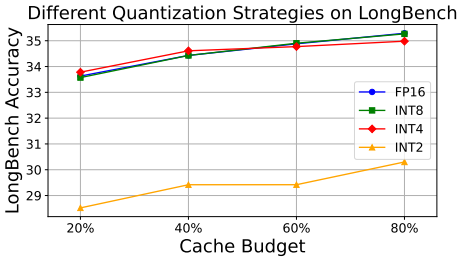


Figure 10: Performance of per-token quantized H_2O on the LongBench dataset. INT8/4 quantization can maintain performance across cache budgets. However, INT2 quantization suffers from a catastrophic drop in performance.

cache in lower layers and less in higher ones. The token allocation across layers follows a linear function. Specifically, considering the average heavy budget size is x, we choose a hyper-parameter pyramid depth d to adjust the ratio. The bottom-most layer has a heavy budget size of x/d, and the top-most layer has a heavy budget size of $2x - x/d$ with intermediate layers linearly interpolated between these values. We choose pyramid depth $d = 7$ for our experiments.

H KV Cache Compression Ratio Analysis

Given a model with (H) layers, hidden dimension (d), number of attention heads (n_{heads}), and a prompt and generated sequence of length $(l_{\text{prompt}}, l_{\text{gen}})$ the KV cache size for different techniques is shown below:

1. **Full model**: All tokens are stored in FP16 format. Therefore the KV cache has size $= 2 \times (H \times d) \times (l_{\text{prompt}} + l_{\text{gen}}) \times 2$ bytes.
2. **H_2O**: Given a cache budget of $(\alpha_{HH}, \alpha_{RW})$ for the heavy hitters and recent window the KV cache has size $= 2 \times (H \times d) \times (l_{\text{prompt}}) \times (\alpha_{HH} + \alpha_{RW}) \times 2$ bytes
3. **KIVI**: With a group size of 16, i.e., 16 scalars quantized from FP16 to INT2 format, the memory required by a group is 16 scalars $\times 2$ bits = 4 bytes. The quantization zero-point and scale are saved in FP16 format and require 2×2 bytes. In total, the group requires 8 bytes. Hence, the KV cache has $(H \times d) \times (l_{\text{prompt}} + l_{\text{gen}})$ bytes.
4. **Q-Hitter**: The Q-hitter paper performs INT4 token quantization per attention head. Therefore, the (d/n_{heads}) scalars which would be stored in FP16 are now stored in 4-bit precision. The quantization metadata is the zero-point and scale, both in FP16 precision. Therefore, the compression factor for Q-Hitter is $(d/n_{heads} * 16)/(d/n_{heads} * 4 + 2 * 16)$. For the Llama-7B-chat model this number is $(4096/32 * 16)/(4096/32 * 4 + 32) = 3.76\times$. Hence, the KV cache size is $2 \times (H \times d) \times (l_{\text{prompt}}) \times (\alpha_{HH} + \alpha_{RW}) \times 2/3.76$ bytes
5. **MiniKV**: The prompt tokens are evicted with a cache budget of α_{HH}, α_{RW} and all generated tokens are retained. All tokens are stored in 2-bit precision. Similar to KIVI, each group of 16 scalars and their quantization metadata requires 8 bytes in total. Hence, the size of the KV cache is $= (H \times d) \times (\alpha_{HH} + \alpha_{RW}) \times (l_{\text{prompt}}) + (H \times d) \times (l_{\text{gen}})$ bytes.

Given a certain prompt and output length, the uncompressed baseline and KIVI have a fixed KV cache size. However, H_2O, Q-Hitter, and MiniKV can tune the cache budget $(\alpha_{HH}, \alpha_{RW})$ to modify the KV cache size.

For prompt length 4096 and generation length 512 the full model's and MiniKV's KV cache consume 2.4GB and 0.33GB respectively. Therefore, MiniKV leads to an $(1 - 0.33/2.4) = 86\%$ reduction in KV cache size.

I Performance against KV cache size

As discussed in § H, the KV cache size depends on the prompt and generation length. Each dataset in LongBench has a different maximum generation length, therefore we make separate plots for each dataset with prompt length 4096 and the generation length as the dataset-specific maximum generation length. Figure 11 and 12 show the performance vs KV cache size curve. MiniKV achieves the optimal compression strategy across all six major task categories on LongBench (single/multi-doc QA, LC understanding, code completion, summarization, and few-shot learning). These results validate the

effectiveness of MiniKV with varying KV cache sizes.

J End-To-End Latency Breakdown

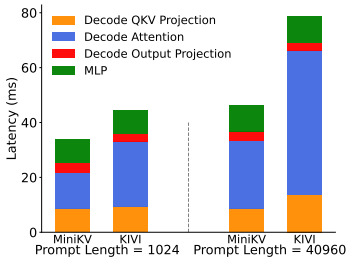


Figure 13: Per token latency breakdown for the decoding phase. Generation length = 1024 and batch size = 1.

We analyze the breakdown of latency associated with each computation in the standard decoder layer of the transformer architecture for MiniKV and KIVI during the decoding phase. We particularly look at latencies for projections of the input vector into query, key, and value vectors, attention computation, and output projection. We also measure the time spent in the MLP layer. We present the latency breakdown as the total latency for each computation component divided by the generation length.

As shown in Fig. 13, MiniKV achieves a lower end-to-end latency than KIVI. This improvement primarily arises during attention computation as well as projection of Query, Key and Value. Specifically, the inference time is dominated by KV cache loading time when processing long contexts. Therefore, MiniKV's smaller KV cache results in reduced KV load times from the GPU's HBM.

K KV Cache Eviction on Long-Context Tasks

Fig. 14 shows that with 50% KV cache size, the LLM can still obtain comparable accuracy (e.g., <1 point) as the full KV cache. However, high levels of KV eviction (e.g., 80-95%) hurts LLM's performance on long context tasks significantly.

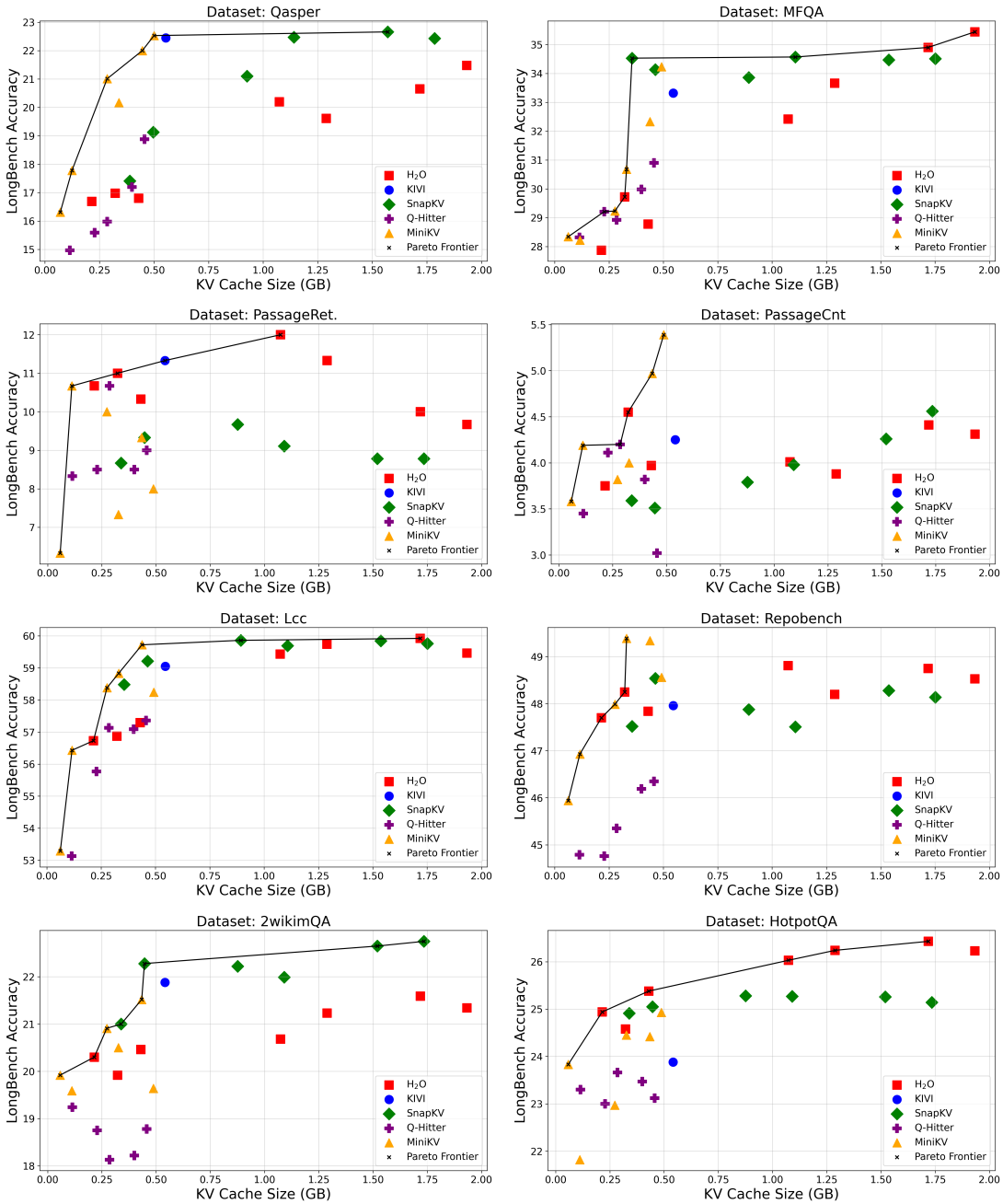


Figure 11: Performance Versus KV Cache Size: MiniKV offers the best performance for the smallest KV cache size across all 6 task categories.

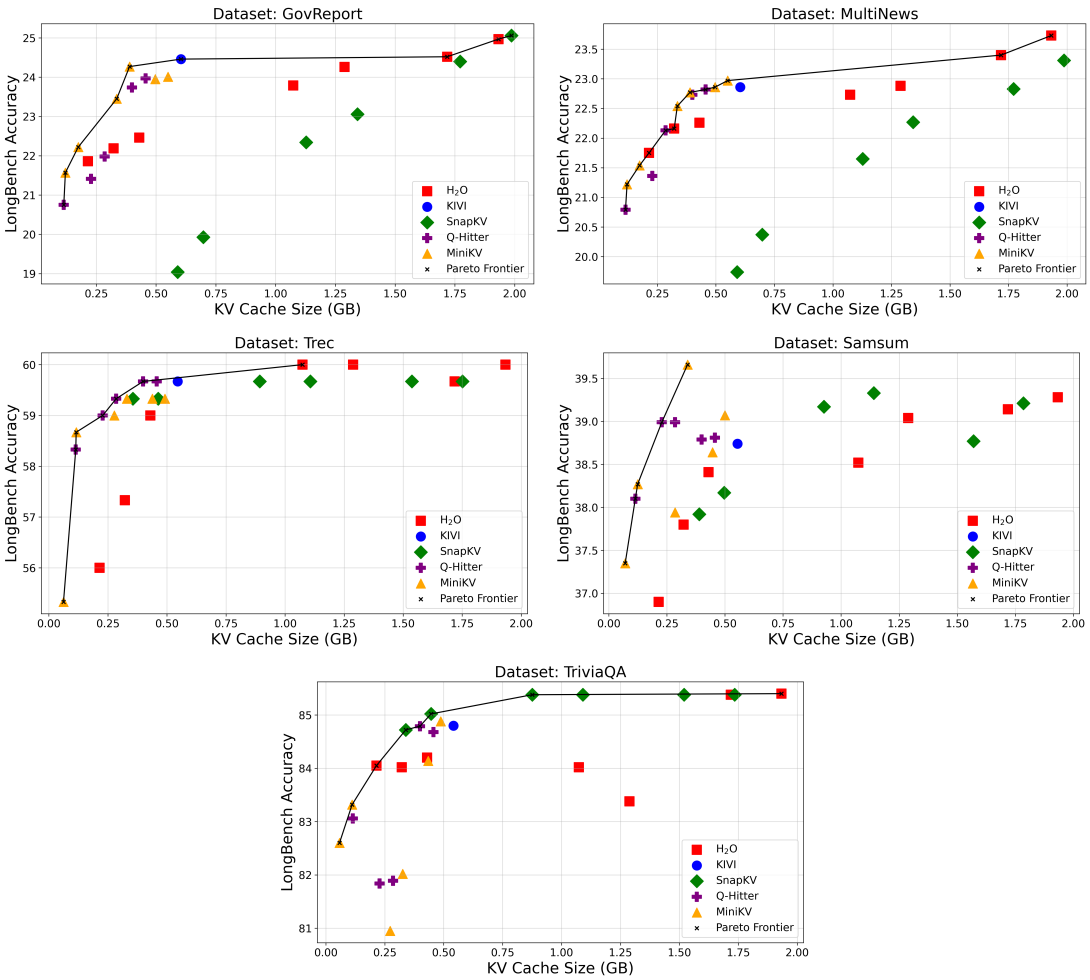


Figure 12: Performance Versus KV Cache Size: MiniKV offers the best performance for the smallest KV cache size across all 6 task categories.

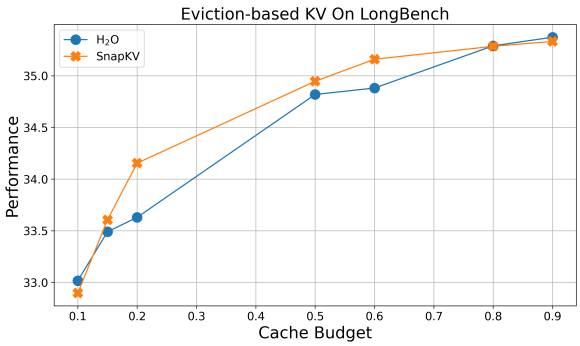


Figure 14: Eviction-based KV on LongBench: High levels of KV eviction (e.g., 80-95%) hurts LLM's performance on long context tasks significantly.