ShadowKV: KV Cache in Shadows for High-Throughput Long-Context LLM Inference

Hanshi Sun*[†‡], Li-Wen Chang[‡], Wenlei Bao[‡], Size Zheng[‡], Ningxin Zheng[‡], Xin Liu[‡], Harry Dong[†], Yuejie Chi[†], and Beidi Chen[†]

[†]Carnegie Mellon University
[‡]ByteDance
{hanshis,harryd,yuejiec,beidic}@andrew.cmu.edu
{liwen.chang,wenlei.bao,zheng.size,zhengningxin,liuxin.ai}@bytedance.com

October 30, 2024

Abstract

With the widespread deployment of long-context large language models (LLMs), there has been a growing demand for efficient support of high-throughput inference. However, as the key-value (KV) cache expands with the sequence length, the increasing memory footprint and the need to access it for each token generation both result in low throughput when serving long-context LLMs. While various dynamic sparse attention methods have been proposed to speed up inference while maintaining generation quality, they either fail to sufficiently reduce GPU memory consumption or introduce significant decoding latency by offloading the KV cache to the CPU. We present ShadowKV, a high-throughput long-context LLM inference system that stores the low-rank key cache and offloads the value cache to reduce the memory footprint for larger batch sizes and longer sequences. To minimize decoding latency, ShadowKV employs an accurate KV selection strategy that reconstructs minimal sparse KV pairs on-the-fly. By evaluating ShadowKV on a broad range of benchmarks, including RULER, LongBench, and Needle In A Haystack, and models like Llama-3.1-8B, Llama-3-8B-1M, GLM-4-9B-1M, Yi-9B-200K, Phi-3-Mini-128K, and Qwen2-7B-128K, we demonstrate that it can support up to 6× larger batch sizes and boost throughput by up to 3.04× on an A100 GPU without sacrificing accuracy, even surpassing the performance achievable with infinite batch size under the assumption of infinite GPU memory. The code is available at `https://github.com/bytedance/ShadowKV`.

1 Introduction

Large language models (LLMs) have increasingly demonstrated their ability to scale and handle long contexts [2, 30, 34, 49], enabling them to tackle complex tasks like multi-document question answering and information retrieval from extensive contexts of up to 1M tokens [2, 53]. However, efficiently serving these long-context LLMs presents challenges related to the key-value (KV) cache [13, 31], which stores previous key-value activations to avoid re-computation. As the KV cache scales with sequence length, its growing memory footprint and the need to access it for each token generation lead to low throughput during long-context LLM inference. To address these issues, KV cache eviction or sparse attention methods have been widely explored.

However, existing methods face three primary limitations: accuracy degradation, inadequate memory reduction, and significant decoding latency overhead. KV cache eviction strategies [66, 67] aim to reduce the memory footprint by discarding KV pairs based on specific policies, but they often result in information loss and accuracy degradation in tasks such as multi-turn conversations [47, 60]. Dynamic sparse attention methods [48] preserve all KV pairs on the GPU and accelerate inference by computing attention with selected KV pairs. However, this line of work does not mitigate the memory footprint, thereby limiting the batch size and preventing accommodation of extremely long contexts (e.g., 1M tokens). A naive solution based on sparse attention

*Work done during the internship at ByteDance.

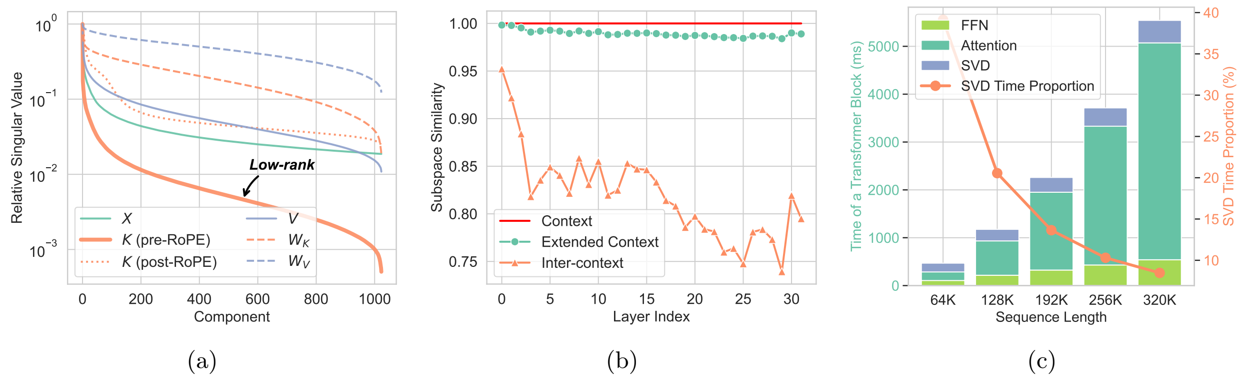


Figure 1: (a) For a sample from PG-19 [12, 40] fed into Llama-3.1-8B, the pre-RoPE keys are the most low-rank, as indicated by the sharpest decay in singular values. (b) Average similarities, defined in Section 3.1, between rank-256 truncated SVD projections of pre-RoPE keys from PG-19 sequences using Llama-3.1-8B. Similarity is measured between a length 16K "Context" and either a 16K+2K continuation on "Context" ("Extended context") or a new length 16K sequence ("Inter-context"). Pre-RoPE keys within sequences exhibit similar low-rank subspaces, while those between sequences show different patterns. (c) The relative overhead of singular value decomposition (SVD) decreases as sequence length scales for the pre-filling stage.

involves offloading the KV cache to the CPU to reduce memory usage [16, 26]. Nonetheless, this approach incurs significant overhead due to the latency of fetching the selected sparse KV pairs from the CPU during decoding.

Consequently, an ideal effective system for long-context LLM inference with sparse attention should: (i) reduce GPU memory usage, (ii) minimize inference latency, and (iii) maintain accuracy within limited sparse KV cache budgets. Fortunately, we can potentially overcome these challenges by leveraging our discovery that pre-Rotary Position Embedding [45] (RoPE) keys are exceptionally low-rank compared to the layer inputs, post-RoPE keys, values, key weight matrix, and value weight matrix, as indicated in Figure 1a. Furthermore, our analysis in Figure 1b reveals that pre-RoPE keys lack significant similarities in low-rank subspaces across different sequences, while a sequence and its continuation tend to strongly share low-rank subspaces, enabling high compression rates within each sequence. Motivated by these findings, we have developed two key insights that pave the way for the design of an applicable system, detailed in Section 3.

Low-rank Keys and Offloaded Values for Storage: In long-context LLM inference, the quadratic scaling of attention computation with sequence length makes the linear cost of low-rank decomposition during pre-filling negligible, as illustrated in Figure 1c[1]. To reduce memory footprint, we retain the low-rank pre-RoPE key cache on the GPU and offload the value cache to the CPU since the value cache does not exhibit low-rank properties, minimizing memory footprint without sacrificing accuracy. During decoding with sparse attention, we employ CUDA multi-streams to overlap the recovery of the selected key cache with the fetching of the corresponding value cache. This approach conceals key cache reconstruction and reduces data fetching overhead by 2× compared to the naive offloading strategy, thereby decreasing the latency of sparse attention during decoding.

Accurate KV Selection for Fast Decoding: To further reduce decoding latency in sparse attention, we propose an accurate KV selection method that maintains accuracy with minimal sparse budgets (1.56%). Our analysis reveals that most post-RoPE keys exhibit high cosine similarity with adjacent tokens, enabling chunk-level approximations for selecting important tokens. A minimal number of outlier chunks (0.3%), which are more challenging to approximate (Figure 3b), are stored as static cache on the GPU to preserve accuracy. As shown in Figure 2, our method outperforms the naive sparse attention approach [48] and achieves higher sparsity, accelerating decoding.

Building on these insights, we present SHADOWKV in Section 4, depicted in Figure 2, a high-throughput system for long-context LLM inference. Specifically, during pre-filling, we offload the value cache to the CPU, retaining only the low-rank pre-RoPE keys, along with compressed landmarks of the key cache and detected outliers for larger batch sizes. During decoding, landmarks are used to select chunk indices for key cache recovery and value cache fetching. We perform accurate sparse attention computation with selected KV pairs and static outliers to achieve high throughput.

[1]In practical scenarios, the key cache can be offloaded to the CPU to perform SVD asynchronously or precomputed and stored as part of the prefix cache [23].

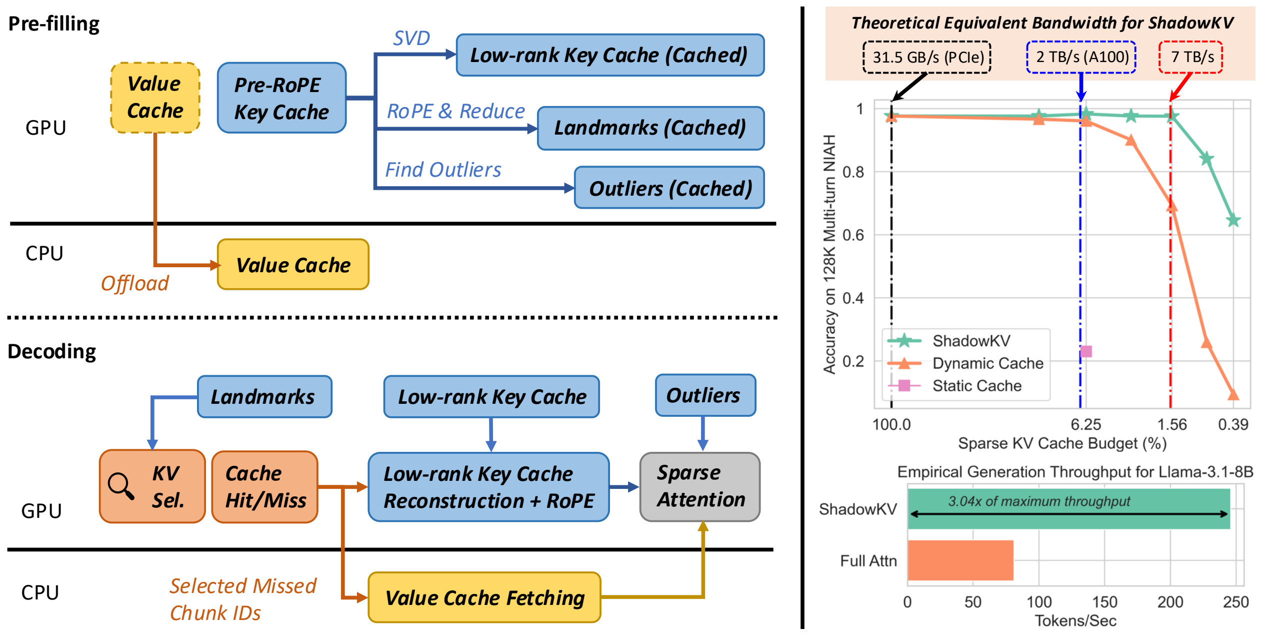


Figure 2: **Left:** ShadowKV enhances long-context LLM inference throughput by offloading the value cache to the CPU while maintaining a low-rank key cache, landmarks, and outliers on the GPU. During decoding, it employs landmarks for efficient sparse attention, reducing computation and data movement. **Right:** ShadowKV effectively utilizes a limited KV budget to achieve high accuracy, theoretically reaching over 7 TB/s equivalent bandwidth on an A100, and empirically boosts generation throughput by 3.04× for Llama-3.1-8B with on a batch of 122K contexts.

Empirically, we conduct extensive experiments and ablation studies to demonstrate the effectiveness and efficiency of ShadowKV. In Section 5.1, we evaluate across various long-context LLMs, such as Llama-3-8B-1M [15], Llama-3.1-8B [33], GLM-4-9B-1M [14], Yi-9B-200K [3], Phi-3-Mini-128K [1] and Qwen2-7B-128K [59] using benchmarks including RULER [20], LongBench [4], and Needle In A Haystack [24] with contexts up to 1M.

In Section 5.2, we demonstrate that ShadowKV can support 6× larger batch sizes and boost throughput by 3.04× compared to small batches on an A100 using Llama-3.1-8B, with each sample having a context length of 122K. We also present results across different models and context lengths, increasing throughput up to 2.97× for Llama-3-8B-1M, 2.56× for GLM-4-9B-1M, and 2.66× for Yi-9B-200K, even surpassing infinite batch size under the assumption of infinite GPU memory.

2 Related Works

Token Eviction. To reduce memory footprint, eviction-based strategies keep a fixed size of KV cache to store the critical token KV pairs and discard unnecessary tokens. StreamingLLM [57] addresses the limitations of window attention by retaining attention sinks and recent KV pairs. H_2O [67] introduces a low-cost eviction policy, updating the KV cache based on cumulative attention scores. LESS [11] accumulates evicted token information by a constant-sized low-rank cache, which allows partial access to previously evicted information, along with tokens maintained by a sparse policy. SnapKV [29] uses the local window of prompts to select important tokens for future generations. However, they suffer from performance degradation and information loss since the evicted tokens will never be recovered.

Dynamic Sparse Attention. This line of work retains all KV cache but performs dynamic sparse attention within selected KV pairs to reduce inference latency. SparQ [41] uses the norm of the query to decide an important subset of the key cache's channels to calculate a metric to select relevant tokens. Quest [48] segments tokens into pages and selects pages by approximating the highest attention within each page. Loki [43] performs principal component analysis on the key cache using a calibration dataset, selecting tokens

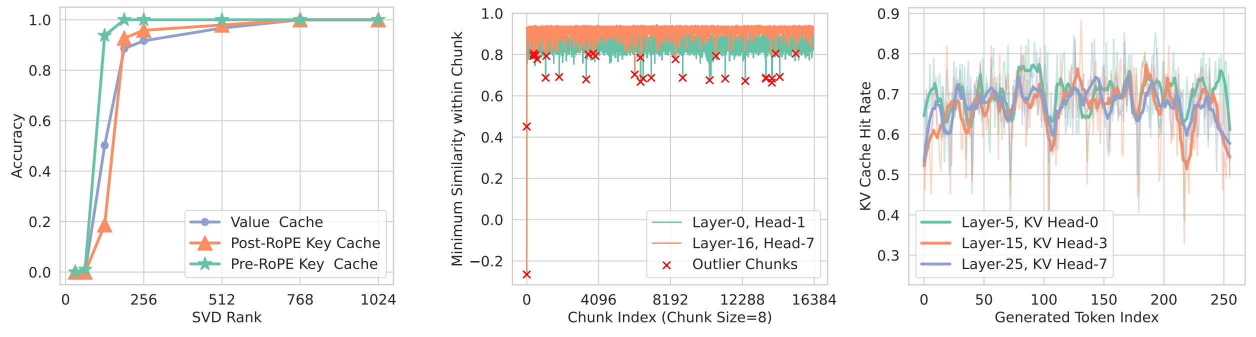


Figure 3: (a) Accuracy on the needle retrieval task across various ranks shows that the pre-RoPE key cache can be compressed by over 6 times without a drop in accuracy. (b) The number of notable outlier chunks is small, taking only 0.2-0.3%. (c) The KV cache has a high hit rate, reducing computations and data movements by over 60% for each decoding step.

based on attention scores computed in low-dimensional space. TriForce [46] combines sparse attention with speculative decoding [27] for lossless acceleration. InfiniGen [26] offloads all KV cache to the CPU and prefetches essential entries during decoding.

Quantization. Several methods have been introduced to optimize KV cache quantization [19, 56, 63], reducing memory consumption while retaining accuracy. KIVI [32] applies different quantization strategies for keys and values, quantizing the keys per-channel and the values per-token to 2-bit. Palu [5] decomposes KV weight matrices offline, caching low-rank KV projections to achieve a higher compression rate. Quantization methods reduce the KV cache bit width, which is orthogonal to our approach.

3 Observations

We present two key insights of long-context LLMs that inspire ShadowKV's design, as follows.

3.1 Low-Rank Keys and Offloaded Values for Storage

To reduce memory footprint, the low-rank nature of the KV cache has been explored by recent studies [5, 9, 58]. However, these methods focus on data-independent decomposition, either requiring training or achieving limited compression rates.

Observation. In our study, by conducting SVD on the model weights W_k, W_v, the input X, the pre-/post-RoPE key cache, and the value cache of Llama-3.1-8B, we visualize the relative singular value distributions in Figure 1a together with the accuracy in Figure 3a. As we observed, pre-RoPE keys have the lowest rank and can be compressed by $6\times$ without performance degradation.

We also identify striking dynamic and static behaviors in low-rank keys between and within sequences, inspired by a related investigation in FFN layers [10]. Analogous to cosine similarity, we define $\mathcal{D}(\boldsymbol{H}_1,\boldsymbol{H}_2)= \langle\boldsymbol{H}_1,\boldsymbol{H}_2\rangle/r$ to be the similarity metric between low-rank subspaces of two rank-r projection matrices, $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$, where $\langle\cdot,\cdot\rangle$ is the Frobenius inner product[2]. In our case with truncated SVDs of pre-RoPE keys, let $\boldsymbol{K}_1,\boldsymbol{K}_2\in\mathbb{R}^{n\times d}$ have rank-r truncated SVDs, $\boldsymbol{\Phi}_1\boldsymbol{\Sigma}_1\boldsymbol{\Psi}_1^\top$ and $\boldsymbol{\Phi}_2\boldsymbol{\Sigma}_2\boldsymbol{\Psi}_2^\top$, respectively, where $\boldsymbol{\Phi}_1\in\mathbb{R}^{n\times r},\boldsymbol{\Sigma}_1\in\mathbb{R}^{r\times r},\boldsymbol{\Psi}_1\in\mathbb{R}^{d\times r}$, and similarly for $\boldsymbol{\Phi}_2$, $\boldsymbol{\Sigma}_2$, and $\boldsymbol{\Psi}_2$. Then, $\mathcal{D}(\boldsymbol{\Psi}_1\boldsymbol{\Psi}_1^\top,\boldsymbol{\Psi}_2\boldsymbol{\Psi}_2^\top)$ can measure the similarity between the low-rank subspaces of the two right singular matrices. Depicted in Figure 1b, pre-RoPE keys between sequences do not strongly share similar low-rank subspaces, but extensions of the same sequence do.

[2]Since $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ are projection matrices, their squared Frobenius norms are the sum of their singular values which consist of r 1's and $d-r$ 0's, i.e., $\|\boldsymbol{H}_1\|_F^2=r$. Thus, by Cauchy-Schwarz, $|\mathcal{D}(\boldsymbol{H}_1,\boldsymbol{H}_2)|\leq 1$. Additionally, $\mathcal{D}(\boldsymbol{H}_1,\boldsymbol{H}_2)\geq 0$ by the cyclic property of trace and positive semidefiniteness of projection matrices. Together, this shows $\mathcal{D}(\boldsymbol{H}_1,\boldsymbol{H}_2)\in[0,1]$, maximized or minimized when the projection matrices project onto identical or orthogonal subspaces, respectively.

Algorithm 1: ShadowKV Pre-filling

```
Input: $\boldsymbol{K}, \boldsymbol{K}^{\text{RoPE}}, \boldsymbol{V} \in \mathbb{R}^{b \times h_{kv} \times s \times d}$, SVD rank $r$, chunk size $c$, number of outlier chunks $o$
▷ Store low-rank projection of pre-RoPE key cache
$\boldsymbol{A} \in \mathbb{R}^{b \times s \times r}$, $\boldsymbol{B} \in \mathbb{R}^{b \times h_{kv} \times r \times d} \leftarrow \text{SVD}(\boldsymbol{K})$
▷ Segment post-RoPE key cache into chunks and compute the mean of each chunk
$\boldsymbol{C} \in \mathbb{R}^{b \times h_{kv} \times s/c \times d} \leftarrow \text{Reduce}(\boldsymbol{K}^{\text{RoPE}})$
▷ Compute cosine similarity within each chunk
$\boldsymbol{S} \in \mathbb{R}^{b \times h_{kv} \times s/c \times c} \leftarrow \text{CosineSimilarity}(\boldsymbol{C}, \boldsymbol{K}^{\text{RoPE}})$
▷ Find lowest cosine similarity as outliers
$\boldsymbol{I} \in \mathbb{R}^{b \times h_{kv} \times o} \leftarrow \text{ArgTopK}(-\text{Min}(\boldsymbol{S}, \text{dim}=-1), o)$
$\boldsymbol{K}^{\text{outlier}}, \boldsymbol{V}^{\text{outlier}} \leftarrow \text{Gather}(\boldsymbol{K}^{\text{RoPE}}, \boldsymbol{V}, \boldsymbol{I})$
▷ Offload the rest of values to the CPU and store the non-outlier chunks' mean as landmarks
$\boldsymbol{V}^{\text{CPU}} \leftarrow \boldsymbol{V} \setminus \boldsymbol{V}^{\text{outlier}}$, $\boldsymbol{L} \leftarrow \boldsymbol{C} \setminus \text{Gather}(\boldsymbol{C}, \boldsymbol{I})$
```

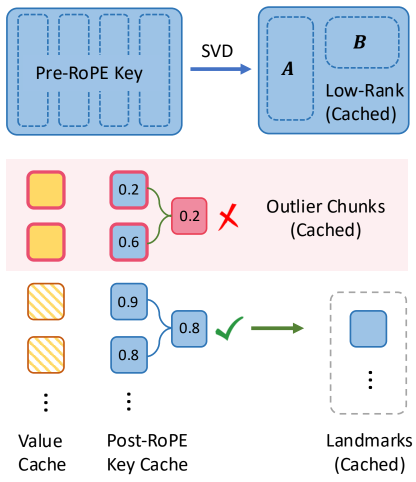


Figure 4: ShadowKV pre-filling.

Insights. Our observation of the low-rank nature in the pre-RoPE keys indicates that storing the low-rank projections is sufficient for each sequence. By keeping the low-rank key cache on the GPU and offloading the value cache to the CPU since it is not low-rank, we can largely reduce the memory footprint. During decoding, selected KV pairs can be reconstructed on-the-fly for computation.

3.2 Accurate KV Selection for Fast Decoding

To further reduce the latency overhead in sparse attention, including fetching the selected value cache from the CPU and reconstructing the corresponding key cache, an accurate KV selection method is needed to minimize the sparse KV cache budget while maintaining the accuracy.

Observation. We found most post-RoPE key cache exhibits spatial locality, with high cosine similarity to adjacent tokens, except for a few outliers. To quantify this, we conducted inference experiments on 128K contexts. We divided the post-RoPE keys into chunks of eight tokens and visualized the minimum cosine similarity between the chunk's mean and its key cache, as shown in Figure 3b. The results indicate that, apart from a few outliers, there is generally high cosine similarity, suggesting the mean values can serve as landmarks to approximate attention well within normal chunks.

Analysis. This finding suggests that for the majority of chunks, we can maintain the mean value as compressed landmarks to select minimal important KV pairs (1.56%) accurately during decoding. Outlier chunks, which may contain dense or critical information and are difficult to approximate, are retained to ensure accuracy. Given their relatively small number (0.2–0.3%), storing them on the GPU is feasible without affecting memory capacity. Furthermore, as shown in Figure 3c, considering the temporal locality of the KV cache, a cache policy [64] can be leveraged to further reduce the latency overhead by 60% during decoding with optimized kernels [35].

4 ShadowKV

In this section, we introduce ShadowKV, a high-throughput long-context LLM inference system. We first elaborate our algorithm in Section 4.1, covering both the pre-filling and decoding phases. Subsequently, in Section 4.2, we discuss the concept of theoretical equivalent bandwidth to illustrate the benefits of our approach.

4.1 Algorithm

The algorithm of ShadowKV is divided into two main phases: pre-filling and decoding. The pre-filling phase involves low-rank decomposition of the post-RoPE key cache, offloading the value cache, and constructing

Algorithm 2: ShadowKV Decoding

```
Input: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{L}$, $\boldsymbol{V}^{\text{CPU}}$, $\boldsymbol{Q}\in\mathbb{R}^{b\times h_q\times s_q\times d}$, $\boldsymbol{K}^{\text{outlier}}$, $\boldsymbol{V}^{\text{outlier}}$, $\boldsymbol{K},\boldsymbol{V}\in\mathbb{R}^{b\times h_{kv}\times s_q\times d}$, number of chunks $n_c$, number of selected chunk budget $k$
    ▷ Compute chunk attention score
    $\boldsymbol{P}\in\mathbb{R}^{b\times h_q\times s_q\times n_c}\leftarrow\text{MatMul}(\boldsymbol{Q},\boldsymbol{L}^\top)$
    $\boldsymbol{S}\in\mathbb{R}^{b\times h_q\times s_q\times n_c}\leftarrow\text{Softmax}(\boldsymbol{P}/\sqrt{d})$
    $\boldsymbol{S}_1\in\mathbb{R}^{b\times h_q\times n_c}\leftarrow\text{sum}(\boldsymbol{S},\text{dim}=-2)$
    $\boldsymbol{S}_2\in\mathbb{R}^{b\times h_{kv}\times n_c}\leftarrow\text{max}_{\text{kv\_group}}(\boldsymbol{S}_1)$
    ▷ Select top-k chunks for each KV head
    $\boldsymbol{I}\in\mathbb{R}^{b\times h_{kv}\times k}\leftarrow\text{ArgTopK}(\boldsymbol{S}_2,k)$
    ▷ Gather values from CPU
    $\boldsymbol{V}^{\text{sparse}}\leftarrow\text{Gather}(\boldsymbol{V}^{\text{CPU}},\boldsymbol{I})$
    $\boldsymbol{V}\leftarrow[\boldsymbol{V}^{\text{outlier}};\boldsymbol{V}^{\text{sparse}};\boldsymbol{V}]$
    ▷ Recover keys from low-rank projection
    $\boldsymbol{K}^{\text{sparse}}\leftarrow\text{MatMul}(\text{Gather}(\boldsymbol{A},\boldsymbol{I}),\boldsymbol{B})$
    $\boldsymbol{K}\leftarrow[\boldsymbol{K}^{\text{outlier}};\text{RoPE}(\boldsymbol{K}^{\text{sparse}});\boldsymbol{K}]$
```

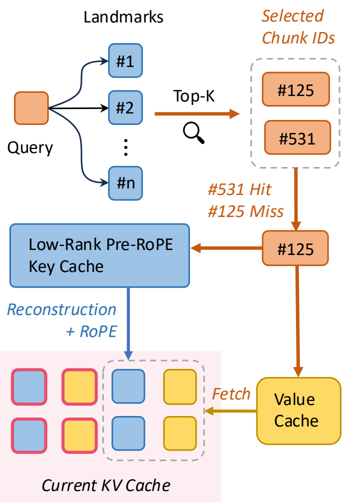


Figure 5: ShadowKV decoding phase.

landmarks to facilitate subsequent high-throughput decoding. The decoding phase includes accurate KV selection and efficient sparse KV cache reconstruction.

Pre-filling. During the pre-filling phase, we optimize GPU memory usage by performing low-rank compression on the key cache of each layer and offloading values to the CPU. Specifically, as demonstrated in Algorithm 1 and Figure 4, we apply SVD on the pre-RoPE key cache and store only the low-rank representations for each layer. Post-RoPE key cache is segmented into chunks, with the mean of each chunk computed as landmarks. By computing the cosine similarity within these chunks, we identify poorly approximated tokens as outliers. This small set of outliers is gathered and stored on the GPU as the static cache, while the remaining key cache is maintained as compact landmarks, with the corresponding values offloaded to the CPU memory.

High-throughput Decoding. For incoming queries, we first compute the approximate attention scores using the landmarks. As detailed in Algorithm 2, by identifying the top-k scoring chunk indices, the corresponding values are retrieved from the CPU, and the key cache is simultaneously reconstructed from low-rank projections, effectively concealing the construction of the key cache. Based on the insight that the KV cache has temporal locality, we build cache-aware CUDA kernels, reducing computation and value fetching by 60%. As shown in Figure 5, we conduct an index scan to detect the missed chunks and only rebuild the necessary KV pairs on-the-fly.

Based on our observations in Section 3.1, future pre-RoPE keys within a sequence reside in a shared low-rank subspace with the context. As a result, an extension of our algorithm would be to store generated tokens as low-rank states using the same projections obtained from pre-filling to reduce the memory usage for future generations[3]. For simplicity, we exclude it from the implementation.

4.2 Theoretical Equivalent Bandwidth

The benefit of ShadowKV in terms of increasing throughput can be analyzed through the concept of equivalent bandwidth. Consider each K or V vector as being M bytes in size, with a sequence length of S, a chunk size of C, a selected chunk budget of K, O outliers, and hit rate α. During KV selection, ShadowKV loads approximately $M\times S/C$ bytes using the GPU memory bandwidth B_{GPU}. For value cache fetching, it loads $M\times K\times C$ bytes using the PCIe bandwidth B_{PCIe} [42]. Since value movement and key cache reconstruction can be overlapped, we do not need to count key cache reconstruction here. Following this, ShadowKV performs standard attention computation for the top-k chunks and predefined outliers, requiring $2M\times(K+O)\times C$ bytes. The equivalent bandwidth of ShadowKV can be defined as:

[3]If $\boldsymbol{\Psi}\in\mathbb{R}^{d\times r}$ is the right singular matrix calculated from the SVD of pre-RoPE context keys $\boldsymbol{K}\in\mathbb{R}^{s\times d}$, new pre-RoPE keys $\boldsymbol{K}'\in\mathbb{R}^{s_q\times d}$ can be stored as $\boldsymbol{K}'\boldsymbol{\Psi}$ and projected back up with $\boldsymbol{\Psi}^\top$ when needed.

$$B_{\text{equivalent}} = \frac{2SB_{\text{GPU}}}{S/C+2(K+O)C+(1-\alpha)KCB_{\text{GPU}}/B_{\text{PCIe}}}$$

For example, assuming C=8, S=128K, K=256, O=48, B_{PCIe}=31.5 GB/s, and B_{GPU}=2 TB/s for A100, the equivalent bandwidth of ShadowKV is calculated as 7.2 TB/s, which is 3.6× higher than A100 memory bandwidth. This result indicates that ShadowKV theoretically achieves a high equivalent bandwidth to accelerate attention computation. System implementation is detailed in Appendix A.1.

Table 1: Performance of different models and different methods on RULER [20] evaluated at length of 128K. ShadowKV outperforms other methods with a 1.56% sparse budget.

Methods	N-S1	N-S2	N-MK1	N-MK2	N-MQ	N-MV	QA-1	QA-2	VT	FWE	Avg.
Llama-3-8B-1M	100.00	100.00	98.96	98.96	98.96	95.57	75.00	48.96	78.54	71.85	86.68
Loki	18.75	1.04	2.08	0.00	1.56	0.78	4.17	13.54	26.04	25.35	9.33
Loki (V only)	41.67	6.25	37.50	1.04	8.07	30.73	10.42	19.79	51.67	37.50	24.46
Quest	**100.00**	**100.00**	**98.96**	77.08	97.65	93.49	60.42	50.00	77.08	65.63	82.03
Quest (V only)	**100.00**	**100.00**	**98.96**	85.42	**97.92**	95.49	70.83	46.88	78.75	65.63	83.99
ShadowKV	**100.00**	**100.00**	97.92	**98.96**	96.88	**95.83**	**72.92**	**52.08**	**81.67**	**72.57**	**86.88**
GLM-4-9B-1M	100.00	100.00	94.79	87.50	99.74	93.75	67.71	55.21	97.29	72.22	86.82
Loki	71.88	27.08	22.92	2.08	9.90	11.46	28.13	27.08	31.04	54.17	28.57
Loki (V only)	96.88	55.21	56.25	18.75	51.04	50.52	45.83	39.58	72.71	59.72	54.65
Quest	**100.00**	95.83	90.62	54.17	94.01	76.30	55.21	52.08	95.83	64.58	77.86
Quest (V only)	**100.00**	96.88	93.75	72.92	95.83	83.07	56.25	53.13	96.88	65.97	81.47
ShadowKV	**100.00**	**100.00**	**95.83**	**83.33**	**98.70**	**87.76**	**69.79**	**55.21**	**97.50**	**68.06**	**85.62**
Llama-3.1-8B	100.00	100.00	98.96	91.67	98.96	95.31	82.29	47.92	68.96	71.18	85.53
Loki	68.75	32.29	32.29	20.83	42.71	28.65	41.67	33.33	24.79	29.86	35.52
Loki (V only)	95.83	36.46	57.29	62.50	77.86	70.83	69.79	39.58	35.21	37.50	58.29
Quest	**100.00**	98.96	97.92	34.38	93.49	88.54	70.83	44.79	65.63	**68.40**	76.29
Quest (V only)	**100.00**	98.96	98.96	56.25	95.83	90.63	76.04	46.88	66.25	67.36	79.72
ShadowKV	**100.00**	**100.00**	**100.00**	**83.33**	**97.92**	**92.19**	**81.25**	**48.96**	**67.08**	64.93	**83.57**
Yi-9B-200K	100.00	100.00	86.46	62.50	64.58	32.55	44.79	39.58	36.87	89.93	65.73
Loki	34.38	2.08	2.08	0.00	0.00	0.52	22.92	21.88	0.00	25.00	10.89
Loki (V only)	59.38	11.46	18.75	5.21	4.43	2.08	22.92	31.25	0.00	35.07	19.06
Quest	**100.00**	98.96	79.17	26.04	56.51	31.77	32.29	31.25	51.04	71.88	57.89
Quest (V only)	**100.00**	**100.00**	80.21	45.83	59.37	**31.90**	36.45	34.37	53.54	71.88	61.36
ShadowKV	**100.00**	**100.00**	**82.29**	**67.71**	**63.28**	31.51	**43.75**	**38.54**	**56.04**	**72.22**	**65.53**

5 Empirical Evaluation

In this section, we showcase the effectiveness and efficiency of ShadowKV. Specifically,

- In Section 5.1, we show that ShadowKV can reduce the GPU memory footprint of the KV cache by over 6× without accuracy degradation on a wide range of models and evaluation benchmarks.
- In Section 5.2, we demonstrate ShadowKV can support up to 6× larger batch sizes and increase the inference throughput by up to 3.04× without compromising model quality.
- In Section 5.3, we present extensive ablation studies that validate the effectiveness of each component of ShadowKV in optimizing GPU memory usage and enhancing performance.

All details (hyperparameters, datasets, etc.), along with additional experiments, are in Appendix A.

5.1 Accuracy Evaluation

We demonstrate that ShadowKV can reduce the GPU memory usage of the KV cache by 6× while maintaining accuracy on a range of long-context tasks with a minimal sparse KV cache budget.

Table 2: Performance of various methods on different models with LongBench [4] samples exceeding 4K tokens. ShadowKV outperforms other methods and maintains the accuracy.

Methods	NarratQA	MultiFQA	HotpotQA	MuSiQue	DuRead	GovRep	SAMSum	PassRetr	LCC	Avg.
Llama-3-8B-1M	18.98	41.84	36.79	21.47	31.93	34.18	35.96	81.50	56.07	39.86
Loki	2.26	10.19	5.48	3.16	12.17	28.97	7.84	40.52	31.44	15.78
Loki (V only)	3.20	21.01	12.41	3.86	17.07	31.24	16.23	52.57	38.10	21.74
Quest	**20.13**	36.63	35.00	18.14	24.55	27.11	35.63	79.00	53.64	36.65
Quest (V only)	17.26	39.51	36.78	18.71	26.41	29.49	35.80	79.50	60.05	38.17
ShadowKV	17.17	**39.73**	**38.29**	**21.08**	**31.77**	**31.62**	**35.87**	**80.00**	**63.93**	**39.94**
GLM-4-9B-1M	25.44	51.09	58.67	39.61	32.04	29.97	40.31	99.00	58.02	48.24
Loki	5.82	30.60	22.73	9.20	30.09	30.35	22.70	98.92	40.77	32.35
Loki (V only)	10.89	44.97	45.44	23.51	32.07	**30.56**	35.34	**99.50**	50.27	41.39
Quest	23.81	44.53	56.41	35.49	23.54	21.73	37.39	87.00	43.80	41.52
Quest (V only)	26.00	46.32	57.54	36.42	24.58	24.52	37.71	93.50	46.52	43.68
ShadowKV	**26.50**	**51.31**	**59.09**	**38.87**	**32.92**	28.54	**38.70**	96.50	**58.55**	**47.89**
Llama-3.1-8B	31.56	55.10	57.65	29.46	35.26	34.45	29.84	100.00	67.31	48.96
Loki	2.31	18.89	10.64	5.47	19.30	31.16	15.91	94.88	44.60	27.02
Loki (V only)	3.93	38.59	22.85	12.96	27.43	32.22	26.43	98.25	56.11	35.42
Quest	29.70	49.04	53.96	27.18	27.16	30.43	29.85	98.50	57.35	44.80
Quest (V only)	30.02	53.97	56.39	27.06	29.06	31.65	30.23	99.00	63.89	46.81
ShadowKV	**30.93**	**55.20**	**57.32**	**29.13**	**31.85**	**32.79**	**30.40**	**99.50**	**66.03**	**48.13**
Yi-9B-200K	13.88	30.02	52.46	28.20	22.29	30.25	19.08	67.00	73.50	37.41
Loki	1.63	2.73	16.21	4.87	4.75	2.13	4.95	0.00	38.72	8.44
Loki (V only)	1.96	10.39	21.31	7.36	6.78	9.15	10.02	4.00	58.75	14.41
Quest	10.57	25.83	46.06	23.04	17.09	17.11	20.59	50.50	67.70	30.94
Quest (V only)	**14.56**	25.73	48.73	24.73	18.44	20.83	20.08	57.50	71.13	33.53
ShadowKV	12.44	**30.82**	**52.43**	**27.73**	**20.79**	**29.83**	**20.73**	**64.00**	**72.89**	**36.85**

Setup. We choose four widely used long-context models for our evaluation: Llama-3-8B-1M [15], GLM-4-9B-1M [14], Llama-3.1-8B [33], and Yi-9B-200K [3]. We evaluate our approach on three challenging long-context benchmarks: RULER [20], LongBench [4], and Needle In A Haystack [24], covering QA, multi-hop, reasoning, summarization, code completion. Needle In A Haystack is also tested on Phi-3-Mini-128K [1] and Qwen2-7B-128K [59]. We set the chunk size to 8, the rank to 160, and the number of outliers to 48 for ShadowKV.

Baselines. We include two dynamic sparse attention methods as baselines: Quest [48] and Loki [43]. For all methods, we retain exact pre-filling and perform dynamic sparse attention during decoding, where the computation cost is set to 1/16 of full attention for selecting sparse KV pairs. We include two variants for each baseline: one where all the KV cache is offloaded, and another where only the value cache is offloaded. The former has similar latency to ShadowKV but a smaller sparse budget since ShadowKV only needs to fetch the value cache from the CPU. The latter aligns with the same sparse KV cache budget but significantly increases GPU memory usage. The latter one is marked as “V only” in the table.

RULER. As shown in Table 1, ShadowKV demonstrates excellent performance on 128K contexts. With a fixed sparse budget of 1.56%, Loki and Quest experience performance degradation. In contrast, ShadowKV is more robust and even outperforms original full attention on certain tasks, such as variable tracking. For complex tasks like multi-document QA or multi-key needle retrieval, other methods suffer from significant performance degradation while ShadowKV does not.

LongBench. On LongBench, we evaluate our method with a range of realistic scenarios, including single-/multi-document question-answering, document summarization, code completion, information retrieval, etc. We only test on samples longer than 4K and set the sparse KV cache budget to 256 for this benchmark since it has shorter inputs compared to RULER. As shown in Table 2, ShadowKV outperforms other methods consistently and maintains the performance.

Needle In A Haystack. On the Needle In A Haystack dataset, as shown in Figure 6, ShadowKV shows the ability to process information at different positions across various context windows, ranging from 16K to 1M tokens. More experiments on a range of models can be found in Appendix A.3.

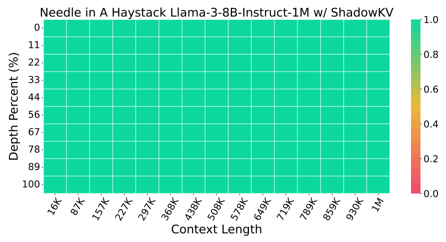


Figure 6: Needle In A Haystack.

Integrate with Efficient Pre-filling Methods. We also combined ShadowKV with a state-of-the-art efficient pre-filling method MInference [22]. As shown in Table 3, following the setting of MInference, we tested it on RULER with contexts scaling from 8K to 256K. This demonstrates that our method is compatible with pre-filling acceleration techniques. For some certain context length settings, we even see a slight performance improvement.

Table 3: Performance of different methods on RULER [20] using MInference [22] in the pre-filling stage. ShadowKV is compatible with MInference.

Methods	8K	16K	32K	64K	128K	256K	Avg.
Llama-3-8B-1M w/ MInference	89.92	88.02	82.81	**78.45**	78.12	**74.57**	81.98
ShadowKV w/ MInference	**90.47**	**88.12**	**83.28**	77.71	**78.32**	74.31	**82.04**

Multi-turn Conversation Capability. As the input context length scales, pre-filling becomes quite costly due to the quadratic growing computation time for attention, which means multi-turn conversation capability is important. To simulate multi-turn conversations, we challenged ShadowKV with a multi-turn needle retrieval task (Multi-turn NIAH). We also test two eviction-based methods in Figure 7, including SnapKV [29] and StreamingLLM [57]. The performance of SnapKV drops significantly from the second round due to the required context information being different from the first round. Since SnapKV inevitably evicted tokens based on the first-turn conversation, it cannot successfully retrieve related information for future queries. In contrast, ShadowKV can maintain accuracy in the multi-turn conversation setting.

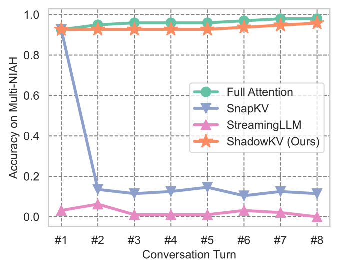


Figure 7: Multi-turn NIAH.

5.2 Efficiency Evaluation

To demonstrate the efficiency of ShadowKV, we deploy it into real-world large batch serving scenarios. By measuring the throughput during decoding across different models on A100, we show that ShadowKV can support up to 6× larger batch sizes and boost throughput by up to 3.04×.

Baselines. The baseline selects the largest batch size that can fit entirely on the GPU with full attention. We also include results for the same batch size of ShadowKV and the infinite batch size, assuming infinite GPU memory capabilities[4]. We set the sparse budget to 1.56% for ShadowKV.

Results. As shown in Table 4, ShadowKV demonstrates significant throughput improvements for various models on an A100, surpassing even those with infinite GPU memory. Notably, ShadowKV supports batch sizes up to 6× larger and enhances throughput by up to 3.04× compared to full attention, even surpassing infinite batch size assuming infinite GPU memory. While the gains for GLM-4-9B-1M and Yi-9B-200K are slightly lower, the improvements still reach up to 2.56× and 2.66× respectively, highlighting ShadowKV's adaptability even with fewer KV heads.

[4]For the equivalent ShadowKV batch size, we evaluate a single Transformer block with FlashAttention and then project the number to the entire model. For the infinite batch size, we leverage A100's theoretical memory bandwidth (2 TB/s) for attention computations.

Table 4: Generation throughput (tokens/s) on an A100. The gray text in brackets denotes batch size.

Model	Context	Full Attention	ShadowKV	Gain	Full Attention (Inf)
Llama-3-8B-1M	60K	160.62 (8)	**455.14** **(48)**	2.83×	168.72 (48) / 273.07 (Inf)
(8 KV heads)	122K	80.77 (4)	**239.51** **(24)**	2.97×	83.05 (24) / 134.30 (Inf)
	244K	40.37 (2)	**119.01** **(12)**	2.95×	52.00 (12) / 67.15 (Inf)
Llama-3.1-8B	60K	160.93 (8)	**472.77** **(48)**	2.94×	168.72 (48) / 273.07 (Inf)
(8 KV heads)	122K	80.78 (4)	**245.90** **(24)**	3.04×	83.05 (24) / 134.30 (Inf)
GLM-4-9B-1M	60K	241.05 (12)	**615.89** **(50)**	2.56×	266.24 (50) / 436.91 (Inf)
(4 KV heads)	122K	122.67 (6)	**293.40** **(25)**	2.39×	158.83 (25) / 214.87 (Inf)
	244K	61.13 (3)	**136.51** **(12)**	2.23×	78.84 (12) / 107.44 (Inf)
Yi-9B-200K	60K	204.81 (10)	**544.36** **(42)**	2.66×	271.21 (42) / 364.09 (Inf)
(4 KV heads)	122K	101.44 (5)	**260.03** **(21)**	2.56×	133.53 (21) / 179.06 (Inf)
	244K	46.74 (2)	**118.55** **(10)**	2.54×	65.79 (10) / 89.53 (Inf)

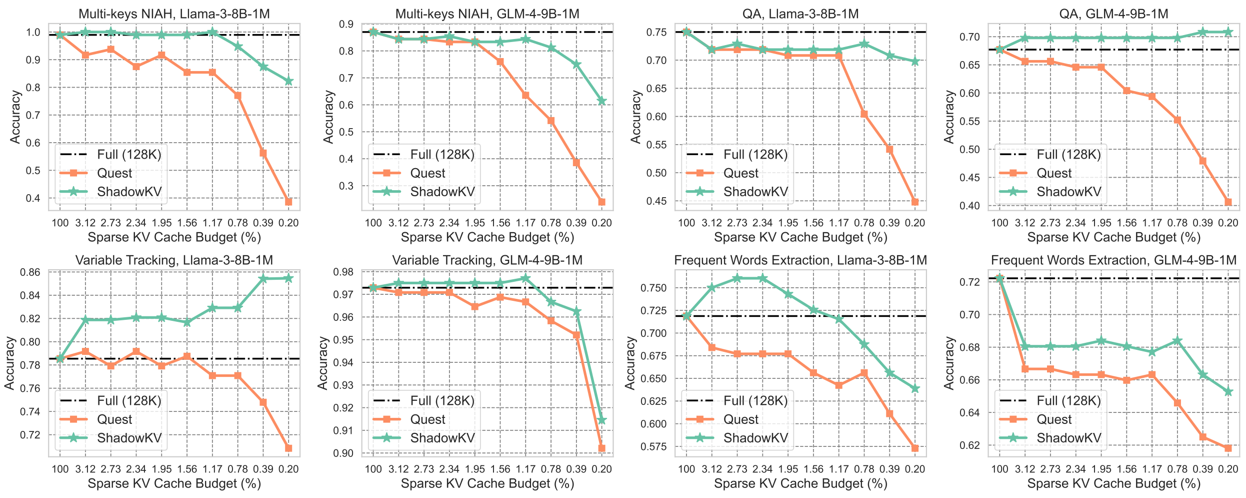


Figure 8: Comparison results between the models with full cache, our ShadowKV, and Quest.

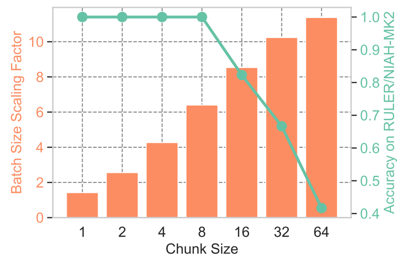
5.3 Ablation Results

We present extensive ablation studies of ShadowKV, focusing on three key points: (1) sparse KV cache budget variations, (2) chunk size selections, and (3) pre-RoPE key cache rank choices.

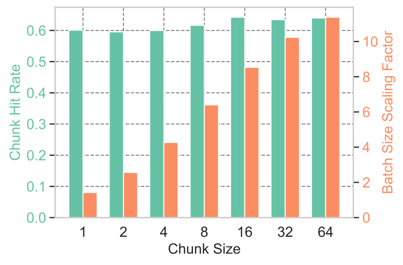
Sparse KV Cache Budget. We examine ShadowKV's performance across various tasks with different sparse budgets, as illustrated in Figure 8. ShadowKV consistently surpasses Quest under the same sparse budgets and achieves higher throughput. On most tasks, it maintains accuracy with just a 1.56% sparse budget compared to full attention and even improves slightly on some tasks.

Chunk Size. As shown in Figure 9a, increasing the chunk size allows for larger batch sizes. However, accuracy declines when the chunk size exceeds eight. Meanwhile, the chunk size choice has minimal impact on the chunk hit rate, which remains around 60%, as illustrated in Figure 9b.

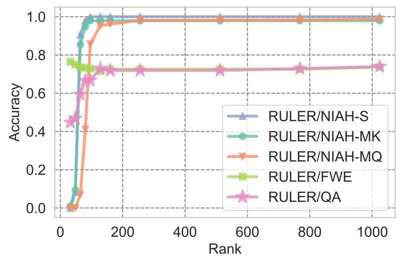
Rank of Pre-RoPE Keys. We assess ShadowKV's performance across various tasks using different ranks for pre-RoPE keys. As illustrated in Figure 9c, accuracy increases with the rank up to approximately 160, after which it stabilizes near full-rank performance. Interestingly, the trends vary across tasks, and in some cases, low-rank approximations achieve better performance.



(a)



(b)



(c)

Figure 9: (a) Impact of chunk size on batch size and accuracy. (b) Minimal effect of chunk size on hit rate. (c) Accuracy trends across different ranks with Llama-3-8B-1M on different tasks.

6 Conclusion

We present ShadowKV, a high-throughput inference system for long-context LLM inference. ShadowKV optimizes GPU memory usage through the low-rank key cache and offloaded value cache, allowing for larger batch sizes. It reduces decoding overhead by accurate sparse attention, boosting throughput while maintaining accuracy. Our empirical experiments demonstrate ShadowKV can support up to 6× larger batch sizes and enhance throughput by up to 3.04× on an A100 across various long-context models, including Llama-3.1-8B, Llama-3-8B-1M, GLM-4-9B-1M, and Yi-9B-200K. ShadowKV holds great promise for improving long-context LLM inference.

References

[1] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] 01. AI, :, Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, Kaidong Yu, Peng Liu, Qiang Liu, Shawn Yue, Senbin Yang, Shiming Yang, Tao Yu, Wen Xie, Wenhao Huang, Xiaohui Hu, Xiaoyi Ren, Xinyao Niu, Pengcheng Nie, Yuchi Xu, Yudong Liu, Yue Wang, Yuxuan Cai, Zhenyu Gu, Zhiyuan Liu, and Zonghong Dai. Yi: Open foundation models by 01.ai, 2024.

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

[5] Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, and Kai-Chiang Wu. Palu: Compressing kv-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*, 2024.

[6] Weize Chen, Ziming You, Ran Li, Yitong Guan, Chen Qian, Chenyang Zhao, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. Internet of agents: Weaving a web of heterogeneous agents for collaborative intelligence. *arXiv preprint arXiv:2407.07061*, 2024.

[7] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[8] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35: 16344–16359, 2022.

[9] DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

[10] Harry Dong, Beidi Chen, and Yuejie Chi. Prompt-prompted adaptive structured pruning for efficient llm generation. In *First Conference on Language Modeling*, 2024.

[11] Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. *arXiv preprint arXiv:2402.09398*, 2024.

[12] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

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

[14] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. Chatglm: A family of large language models from glm-130b to glm-4 all tools, 2024.

[15] Gradient. Llama-3-8b-instruct gradient 4194k (v0.1), 2024. URL `https://huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k`.

[16] Jiaao He and Jidong Zhai. Fastdecode: High-throughput gpu-efficient llm serving using heterogeneous pipelines. *arXiv preprint arXiv:2403.11421*, 2024.

[17] Nan He, Hanyu Lai, Chenyang Zhao, Zirui Cheng, Junting Pan, Ruoyu Qin, Ruofan Lu, Rui Lu, Yunchen Zhang, Gangming Zhao, et al. Teacherlm: Teaching to fish rather than giving the fish, language modeling likewise. *arXiv preprint arXiv:2310.19019*, 2023.

[18] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Hanyu Dong, and Yu Wang. Flashdecoding++: Faster large language model inference on gpus. *arXiv preprint arXiv:2311.01282*, 2023.

[19] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 2024.

[20] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[21] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[22] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *arXiv preprint arXiv:2407.02490*, 2024.

[23] Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-throughput llm inference with shared prefixes. *arXiv preprint arXiv:2402.05099*, 2024.

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

[25] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[26] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic kv cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024.

[27] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[28] Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. Compressing context to enhance inference efficiency of large language models. *arXiv preprint arXiv:2310.06201*, 2023.

[29] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.

[30] Hao Liu, Wilson Yan, Matei Zaharia, and Pieter Abbeel. World model on million-length video and language with ringattention. *arXiv preprint arXiv:2402.08268*, 2024.

[31] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024.

[32] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024.

[33] Meta AI. Introducing Llama 3.1, 2024. URL `https://ai.meta.com/blog/meta-llama-3-1/`. Accessed: 2024-08-21.

[34] Microsoft. Microsoft bingchat, 2024. URL `https://www.bing.com/chat`.

[35] NVIDIA. Cuda toolkit, 2024. URL `https://developer.nvidia.com/cuda-toolkit`. Accessed: 2024-09-25.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[37] Yingzhe Peng, Xu Yang, Haoxuan Ma, Shuo Xu, Chi Zhang, Yucheng Han, and Hanwang Zhang. Icd-lm: Configuring vision-language in-context demonstrations by language modeling. *arXiv preprint arXiv:2312.10104*, 2023.

[38] Yingzhe Peng, Chenduo Hao, Xu Yang, Jiawei Peng, Xinting Hu, and Xin Geng. Learnable in-context vector for visual question answering. *arXiv preprint arXiv:2406.13185*, 2024.

[39] QwenTeam. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

[40] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

[41] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference. *arXiv preprint arXiv:2312.04985*, 2023.

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

[43] Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *arXiv preprint arXiv:2406.02542*, 2024.

[44] Zezheng Song, Jiaxin Yuan, and Haizhao Yang. Fmint: Bridging human designed and data pretrained models for differential equation foundation model. *arXiv preprint arXiv:2404.14688*, 2024.

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

[46] Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*, 2024.

[47] Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024.

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

[49] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[50] Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, Matt Nicely, Duane Merrill, Dustyn Blasig, Fengqi Qiao, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Manish Gupta. CUTLASS, January 2023. URL `https://github.com/NVIDIA/cutlass`.

[51] Vijay Viswanathan, Chenyang Zhao, Amanda Bertsch, Tongshuang Wu, and Graham Neubig. Prompt2model: Generating deployable models from natural language instructions. *arXiv preprint arXiv:2308.12261*, 2023.

[52] Haixin Wang, Xinlong Yang, Jianlong Chang, Dian Jin, Jinan Sun, Shikun Zhang, Xiao Luo, and Qi Tian. Parameter-efficient tuning of large-scale multimodal foundation model. *Advances in Neural Information Processing Systems*, 36, 2024.

[53] Minzheng Wang, Longze Chen, Cheng Fu, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, et al. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa. *arXiv preprint arXiv:2406.17419*, 2024.

[54] T Wolf. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[55] Chaojun Xiao, Zhengyan Zhang, Chenyang Song, Dazhi Jiang, Feng Yao, Xu Han, Xiaozhi Wang, Shuo Wang, Yufei Huang, Guanyu Lin, et al. Configurable foundation models: Building llms from a modular perspective. *arXiv preprint arXiv:2409.02877*, 2024.

[56] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[57] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2023.

[58] Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. Think: Thinner key cache by query-driven pruning. *arXiv preprint arXiv:2407.21018*, 2024.

[59] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[60] June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*, 2024.

[61] Zihao Ye, Ruihang Lai, Bo-Ru Lu, Lin Chien-Yu, Size Zheng, Lequn Chen, Tianqi Chen, and Luis Ceze. Cascade inference: Memory bandwidth efficient shared prefix batch decoding, 2024. URL `https://flashinfer.ai/2024/02/02/cascade-inference.html`. Accessed: 2024-09-25.

[62] Yixiao Yuan, Yangchen Huang, Yu Ma, Xinjin Li, Zhenglin Li, Yiming Shi, and Huapeng Zhou. Rhyme-aware chinese lyric generator based on gpt. *arXiv preprint arXiv:2408.10130*, 2024.

[63] Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*, 2024.

[64] Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *arXiv preprint arXiv:2407.12820*, 2024.

[65] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞bench: Extending long context evaluation beyond 100k tokens, 2024.

[66] Yichi Zhang, Bofei Gao, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, Wen Xiao, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

[67] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[68] Chenyang Zhao, Xueying Jia, Vijay Viswanathan, Tongshuang Wu, and Graham Neubig. Self-guide: Better task-specific instruction following via self-synthetic finetuning. *arXiv preprint arXiv:2407.12874*, 2024.

A Experiment Details

In this section, our goal is to provide the details of the system implementation (mentioned in Section 4.2), experiment settings, and additional experiments (mentioned in Section 5).

A.1 System Implementation.

We implement the framework based on PyTorch [36, 54] and dedicated kernels [50]. FlashAttention [7, 8, 18] is used for attention computation and some efficient fused kernels in Flashinfer [61] and vLLM [25] are used, including layer norm. To reduce memory movement and kernel launch overhead, we fuse some operations into CUDA kernels, including attention approximation, key cache low-rank reconstruction, value cache fetching, cache mechanism, etc. We leverage multi-streams to overlap the reconstruction of key cache and value cache fetching. We set the rank of pre-RoPE key cache to 160, chunk size to 8, and sparse KV cache budget to 1.56% for most cases.

A.2 Dataset Details

LLMs are widely used in various fields [6, 17, 21, 28, 37, 38, 39, 44, 51, 52, 55, 62, 68], and we select three long-context benchmarks, detailed below.

- RULER [20] consists of 13 complex tasks and supports adjustable context lengths, including retrieval, multi-hop tracking, aggregation, and QA tasks. For the test with MInference [22], we set up test sets scaling from 8K to 256K for evaluation.
- LongBench [4] is a challenging long-context benchmark that assesses the performance of LLMs in extended contexts. Featuring Chinese and English languages, LongBench encompasses 6 main categories and 21 diverse tasks, evaluating LLM capabilities across crucial long-text applications like single-/multi-document QA, summarization, code completion, etc.
- Needle In A Haystack [24] is a long-context retrieval benchmark testing LLM's performance with context window scales up to 1M tokens where information placed at various positions. We tested the retrieval capabilities of six long-context LLMs based on their context length.

A.3 Needle In A Haystack

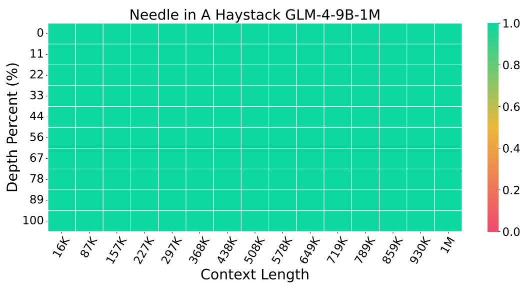
In addition to the Needle In A Haystack results for Llama-3-8B-1M shown in Figure 6, we also present results for GLM-4-9B-1M, Llama-3.1-8B, Yi-9B-200K, Phi-3-Mini-128K, and Qwen2-7B-128K, shown in Figure 10. Compared to full attention, using ShadowKV has minimal impact on the ability to understand semantic information across different context windows and needle depths. There is even a slight performance improvement for Yi-9B-200K.

A.4 InfiniteBench

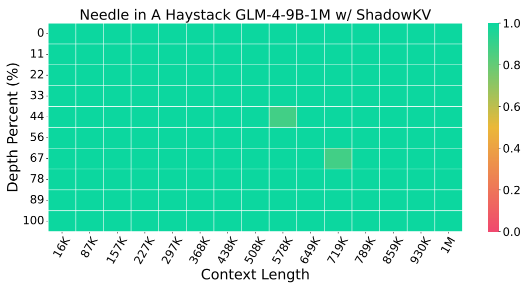
InfiniteBench [65] is a challenging long-context benchmark that consists of 10 tasks, including QA, coding, dialogue, summarization, and retrieval, with an average length of 214K.

Table 5: Accuracy of different methods with different models on InfiniteBench [65].

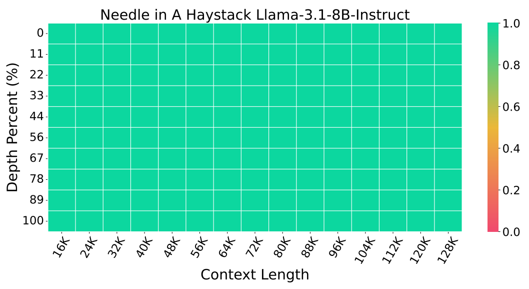
Methods	En.Sum	En.QA	En.MC	En.Dia	Zh.QA	Code.Debug	Math.Find	Retr.PassKey	Retr.Num
Llama-3-8B-1M	23.05	18.14	65.06	10.50	12.47	24.36	37.14	100.00	100.00
ShadowKV	21.50	17.73	64.63	10.50	12.45	23.86	37.43	100.00	100.00
GLM-4-9B-1M	28.61	9.25	68.12	39.50	11.77	30.20	40.00	100.00	100.00
ShadowKV	23.22	8.48	68.56	32.50	11.27	30.46	40.00	100.00	100.00
Llama-3.1-8B	26.42	14.48	66.38	16.00	12.92	21.07	34.00	100.00	99.66
ShadowKV	24.23	13.83	66.38	16.50	12.76	21.07	34.00	100.00	94.41
Yi-9B-200K	8.88	10.61	61.57	5.50	13.88	21.57	23.71	100.00	99.66
ShadowKV	8.92	10.06	59.39	6.00	13.89	20.56	24.29	100.00	99.83



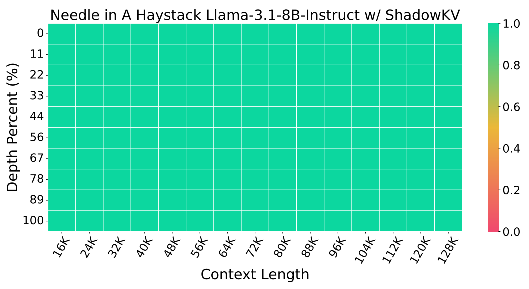
(a) GLM-4-9B-1M



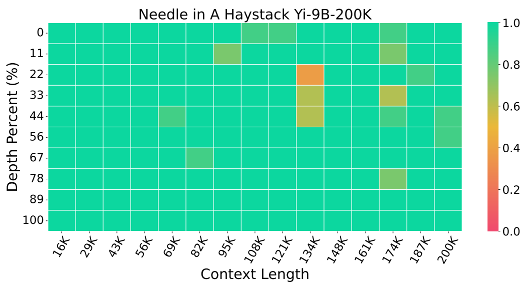
(b) GLM-4-9B-1M w/ ShadowKV



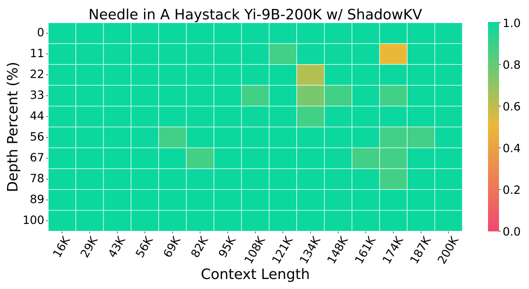
(c) Llama-3.1-8B-Instruct



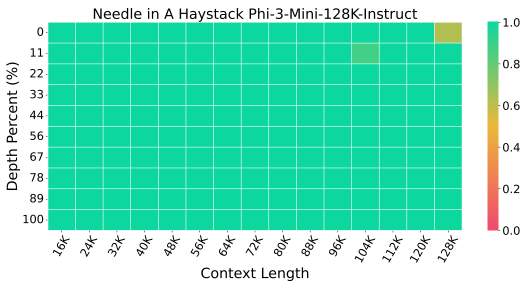
(d) Llama-3.1-8B-Instruct w/ ShadowKV



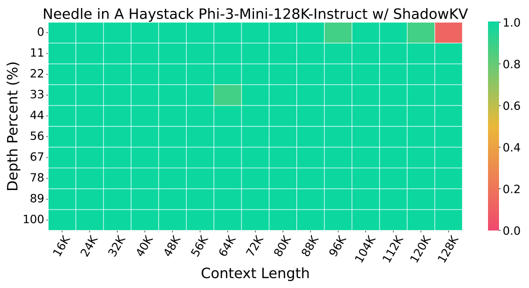
(e) Yi-9B-200K



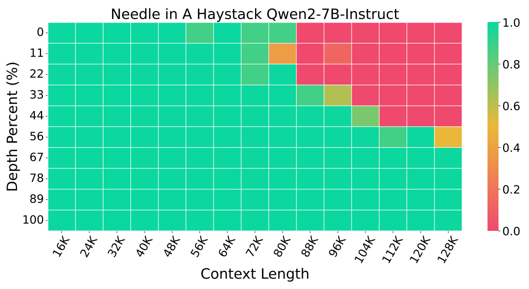
(f) Yi-9B-200K w/ ShadowKV



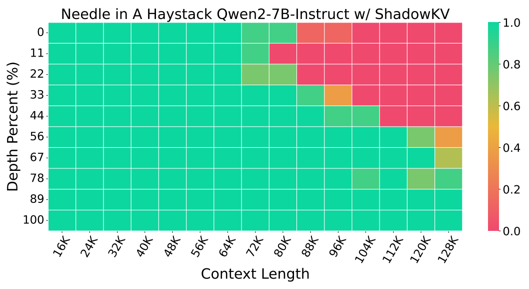
(g) Phi-3-Mini-128K



(h) Phi-3-Mini-128K w/ ShadowKV



(i) Qwen2-7B-128K



(j) Qwen2-7B-128K w/ ShadowKV

Figure 10: Needle In A Haystack [24] results using GLM-4-9B-1M [14], Llama-3.1-8B-Instruct [33], Yi-9B-200K [3], Phi-3-Mini-128K [1], and Qwen2-7B-128K [59].