PRKV:Page Restructed KV Cache for High Accuracy and Efficiency LLM Generation

Anonymous ACL submission

Abstract

As the key-value(KV) cache size scales with context length, the substantial GPU memory demand and the overhead of accessing a large KV cache in each decoding step pose challenges for deploying LLMs with long contexts. Among various sparse attention methods, offloading-based KV retrieval method preserves entire KV cache in CPU memory and dynamically retrieves most relevant KV pairs for each decoding step, which performs higher quality and effectively reduces GPU memory footprint than other methods. However, exiting offloading-based KV retrieval methods perform page-level to reduce estimation overhead, which introduces inaccurate KV selection and significant retrieval overhead. We propose PRKV, a framework that both-optimizes algorithm and system for page-level KV retrieval with KV offloading. On the algorithm side, PRKV introduces hybrid KV selection that combines static and dynamic KV selection strategies. On the system side, PRKV employs contiguous memory indexing and batched transfer optimizations to improve retrieval efficiency. Experiments demonstrate that PRKV **improves accuracy** across various scenarios and models, delivering up to **6.75×** speedup compared to SOTA KV retrieval methods.

1 Introduction

Recent advancements in large language models (LLMs) have significantly enhanced their ability to process long contexts(Wang et al., 2024; Jiang et al., 2024). However, the efficient serving of these long-context LLMs is hampered by two critical challenges related to the key-value (KV) cache, whose size scales with sequence length and inference batch size. The resulting memory-bound access not only slows down decoding but also imposes substantial GPU memory demands, often exceeding hardware limits.

To address these challenges, numerous sparse attention methods have been proposed. KV eviction reduces memory usage and access latency by discarding less important tokens (Xiao et al., 2024b; Zhang et al., 2023), but its inability to recover evicted information can degrade performance on tasks requiring long-range dependencies (Gao et al., 2025; Liu et al., 2025d). As an alternative, KV retrieval preserves the full cache in CPU memory and dynamically fetches relevant KV pairs during decoding. While this approach improves accuracy and alleviates GPU memory pressure, it introduces new challenges in efficient and accurate token selection.

Inaccurate KV selection. KV retrieval requires estimating the importance of each KV pair at every decoding step, token-level approach such as Magicpig and Infinigen (Chen et al., 2025b; Lee et al., 2024)is computationally expensive estimation, as illustrated in Figure 2. To mitigate this overhead, existing methods, such as Quest and Arkvale (Tang et al., 2024; Chen et al., 2024), perform retrieval at the page level. However, this coarser granularity results in the selection of less accurate KV pairs compared to the ground-truth top-k selection, as illustrated in Figure 10.

Significant retrieval overhead. Limited by the transfer efficiency between CPU memory and GPU memory, retrieval introduces significant latency when fetching sparse KV pairs back to the GPU during decoding.

In this paper, we aim to solve above two concerns and build an inference system for long-context LLM with high accuracy and low latency.

Reusing top-k KV pairs. Performing high-precision top-k KV selection at every decoding step is computationally expensive. However, we observe strong similarity among query vectors and high overlap in the top-k KV pairs of consecutive tokens, which enables the reuse of previously selected top-k KV pairs. This reuse strategy maintains high attention recall while incurring minimal additional overhead.

Hybrid KV selection. Although reusing previous top-k KV pairs improves selection accuracy, static selection methods fail to account for the dynamic nature of KV importance (Chen et al., 2024). To address this limitation, we propose a novel hybrid KV selection approach that combines both static and dynamic retrieval strategies during decoding. As illustrated in Figure 3b, our hybrid method achieves a higher recall rate than alternative selection strategies.

New retrieval pattern with low overhead. To reduce latency when retrieving KV pairs from CPU memory, we propose a system-level optimization that addresses two key inefficiencies. First, we note that existing head-wise retrieval methods often overlook the critical impact of the KV cache layout on access efficiency. Our analysis shows that the HND layout enables contiguous memory access for head-wise operations, making it ideal for this context. Second, we implement a batched KV transfer mechanism to maximize PCIe bandwidth utilization during retrieval. Together, these optimizations significantly reduce latency.

Based on the preceding analysis, we introduce PRKV, a framework that both-optimizes algorithm and system design for page-level KV retrieval with KV offloading. As illustrated in Figure 4, the algorithmic component of PRKV features a hybrid KV selection method for decoding steps. On the system side, the framework employs contiguous memory indexing and a batched transfer mechanism to maximize retrieval efficiency.

We conducted extensive experiments and ablation studies to evaluate PRKV. In Section 4.1, we assess various LLMs on benchmarks including LongBench V2 (Bai et al., 2024), Needle In A Haystack (Kamradt, 2023), and the reasoning benchmarks AIME (Jia, 2025) and MATH500 (Hendrycks et al., 2021). The results show that PRKV achieves the **highest accuracy** among other methods and even exceeds the accuracy of the full cache. In Section 4.2, the end-to-end results demonstrate that PRKV achieves a speedup of up to $\mathbf{6.75}\times$ over SOTA KV retrieval methods.

2 Observation

2.1 Recall Rate Definition

Attention mechanisms are core of transformer-based LLMs. For each attention head i, the general attention computation formula is as follows:

$$W_i = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{d}}), \qquad O_i = W_i V_i \quad (1)$$

Here, W_i is the attention scores across tokens, and O_i is the attention output from the i-th attention head. Sparse attention algorithms aim to retain the most important tokens and achieve the highest attention scores with a fixed budget k through different estimation methods, which can be presented as:

$$I_i = \text{argtopk}\left(f(Q_i, K_i),\ k\right) \quad (2)$$

where I_i denotes the selected subset tokens' indices in i-th attention head, and the function f is an efficient estimation method to identify the importance of the tokens. The **Attention Recall Rate** is defined as the proportion of the ground truth attention scores that the selected subset covers:

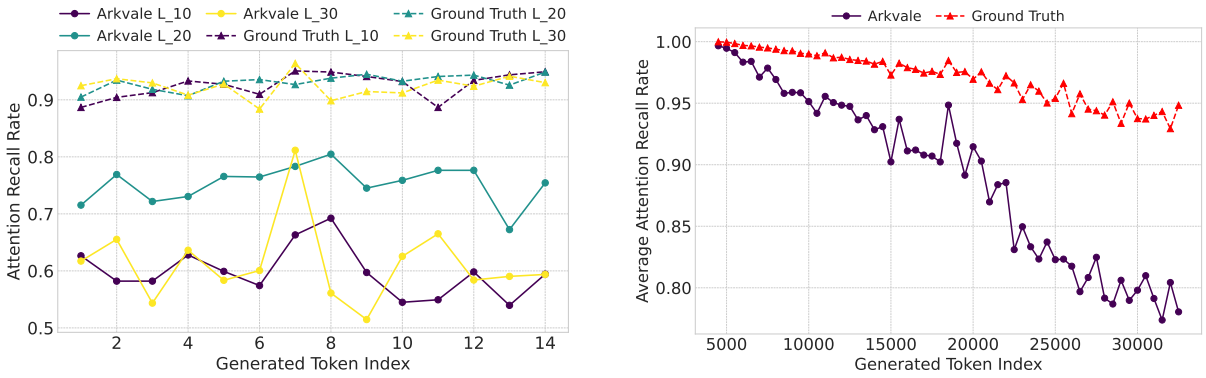
$$\mathcal{R}_i = \sum_{j \in I_i} W_{ij} \quad (3)$$

For a given KV budget k, ground truth top-k means select k KV pairs with highest ground truth attention scores, which guarantees the highest recall rate in return. A high attention recall rate demonstrates that the selected tokens encompass the most significant attention scores, which is essential to maintain high accuracy.

2.2 Long estimation time of Token-Level methods

In long-context scenarios, KV cache is often offloaded to CPU memory due to limited GPU capacity. Token-level retrieval methods, such as MagicPIG (Chen et al., 2025b), estimate token importance by scanning the full KV cache, resulting in substantial computational overhead. In contrast, group-level approaches like Arkvale (Chen et al., 2024) leverage page-level metadata stored in GPU memory to perform coarse-grained importance estimation more efficiently.

To quantify this efficiency gap, we measure the estimation overhead under a 64K context length. As shown in Figure 2, MagicPIG introduces significantly higher estimation latency, reaching up to **15×** the cost of full-cache attention implemented with FlashAttention2 (Dao, 2023), whereas Arkvale incurs considerably lower overhead. These results highlight the scalability limitations of token-level retrieval and motivate the need for more efficient KV selection strategies.



(a) Attention recall rate of different approaches using a 1K token budget on a 37K NIAH task.

(b) Attention recall rate of different approaches using a 4K token budget on an AIME task.

Figure 1: Analysis of attention recall rate between Arkvale and ground-truth Top-k on different tasks. Figure 1a shows attention recall rates of layer 10, layer 20, and layer 30 on a NIAH task with a 1K token budget and 37K input length using Llama3.1-8B-Instruct. Figure 1b shows the average attention recall rate across all layers on an AIME problem with a 4K token budget and generation length up to 32K tokens using Qwen3-8B.

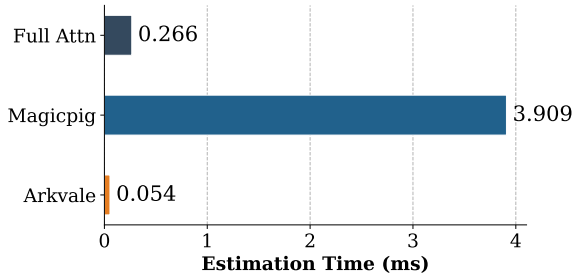


Figure 2: Estimation time of MagicPIG and Arkvale with 64K sequence length

2.3 Limitation of page-level selection

Page-level KV selection reduces estimation overhead by grouping consecutive tokens into coarse-grained pages. However, such approaches overlook the tokens within the same page can exhibit vastly different importance though they have high semantic similarities. As shown in Figure 10, important tokens selected by ground truth topk are distributed throughout the sequence, while Arkvale's page grained nature fails to select these important individual tokens, directly resulting in a **lower accuracy**.

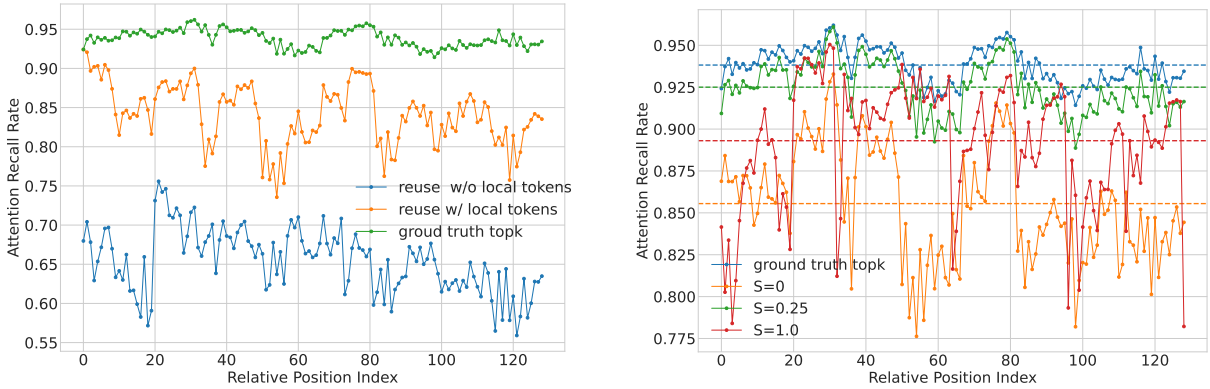
Empirical analysis further reveals the Attention Recall Rate between page-level selection and ground-truth top-k across long-context tasks. As shown in Figure 1a, Arkvale exhibits an average recall rate about **0.25** in NIAH and around **0.8** in AIME2024. Additionally, such discrepancies can introduce accumulated bias during sparse decoding, ultimately degrading model performance.

2.4 Opportunity: Locality-Aware Top-k Reuse

Prior studies show that query vectors of consecutive tokens exhibit strong temporal similarity (Su et al., 2024; Yuan et al., 2025). Leveraging this property, methods such as FreeKV (Liu et al., 2025b) reuse top-k KV pairs from previous decoding steps. Empirical analysis (Figure 3a) shows that reusing only the previous top-k tokens yields an attention recall rate of **0.65** under a KV budget of 1024. However, augmenting the reused set with a small local window (e.g., 64 tokens) increases the recall rate to a level comparable to ground-truth top-k selection. This observation indicates that combining top-k reuse with local context can substantially improve selection accuracy.

2.5 Opportunity: Hybrid sparse token selection

While top-k reuse provides fine-grained token selection, page-level retrieval offers dynamic KV selection. We explore a hybrid strategy that integrates both without token eviction. Given a KV budget k, we partition it into a static **token set** and a dynamic **page set**. The static set occupies a ratio $\mathcal{S}$ of the budget and consists of the top $k\mathcal{S}$ KV pairs reused from the previous step, while the remaining $1-\mathcal{S}$ budget is allocated to dynamically selected pages based on their importance during each decoding step. We evaluate the attention recall rate of this hybrid design on LongBench tasks (Figure 3b), starting selection from relative position 0. We compare against fully dynamic page-level



(a) (b)

Figure 3: Average attention recall rate on sampled LongBench tasks with 32K input length. (a) "reuse w/o local KVs" reuses only top-k ($k = 1024$) KV pairs starting from relative position 0; "reuse w/ local KVs" additionally includes the last 64 local KV pairs. (b) Attention recall rate under different $\mathcal{S}$ settings with a 1K budget.

selection ($\mathcal{S} = 0$, representing Arkvale) and fully static selection ($\mathcal{S} = 1$, representing SnapKV). For hybrid selection, we take $\mathcal{S} = 0.25$ as an example. Results show that hybrid selection consistently achieves higher recall than both extremes, supporting the effectiveness of combining static token reuse with dynamic page retrieval.

3 PRKV

Based on the above analysis and observations, we propose Page Restructed KV Cache (PRKV), a method combines static and dynamic KV pair retrieval in page granularity, achieving higher Attention Recall Rate and lower retrieval overhead.

3.1 Hybrid token selection

Static token-level selection. The detailed mechanism of static token-level selection algorithm is formalized in Algorithm 1 lines 8-19. As discussed in Section 2.4, the local tokens are consistently important in sparse attention computation, so PRKV keeps the local tokens at the end of context. PRKV uses an observation window at the end of sequence(i.e., the local window). It then computes the attention scores between the queries in the observation window and the keys of all preceding tokens. Finally, PRKV selects the top-k tokens with the highest attention scores as the static tokens. After identifying static tokens, PRKV relocates their corresponding KV to the beginning of KV cache to make them contiguous, while preserving the relative order of the remaining tokens and still organizing KV cache to pages. PRKV recomputes and updates page representations since the KV cache pages have been restructured(line 23-26).

To maintain compatibility with Grouped-Query Attention (GQA) and achieve more precise token selection, PRKV modifies the process of computing observation-based attention scores. PRKV first repeats the group's Key cache for each query head. Then computes the attention score for each head individually and averages these scores within the group(line 8-10).

Principles of choosing $\mathcal{S}$. Reuse of previous topk tokens provides a fine-grained selection mechanism, the choice of static token ratio $\mathcal{S}$ should strike a balance that maximizes the likelihood of overlap between important tokens. Based on this intuition, we design an online algorithm for determining $\mathcal{S}$.

In the prefill phase, after computing the observation-window attention scores S (line 10), Let $S^{(1)}$ and $S^{(T)}$ denote the attention score vectors of the first and last query tokens in the prefilling phase observe window, respectively. For each head h, we compute the overlap of their topk key indices using

$$C_h = \left|\text{topk}\left(S_h^{(1)}, k\right) \cap \text{topk}\left(S_h^{(T)}, k\right)\right| \quad (4)$$

The static ratio is then obtained by averaging the overlap across all heads, P denotes the page size.Then the static ratio is used in the subsequent selection process.

$$\mathcal{S} = \frac{P \cdot \left\lfloor \frac{E_h[C_h]}{P} \right\rfloor}{k} \quad (5)$$

Table 1 shows the static token ratio of different models, input lengths and KV budgets.

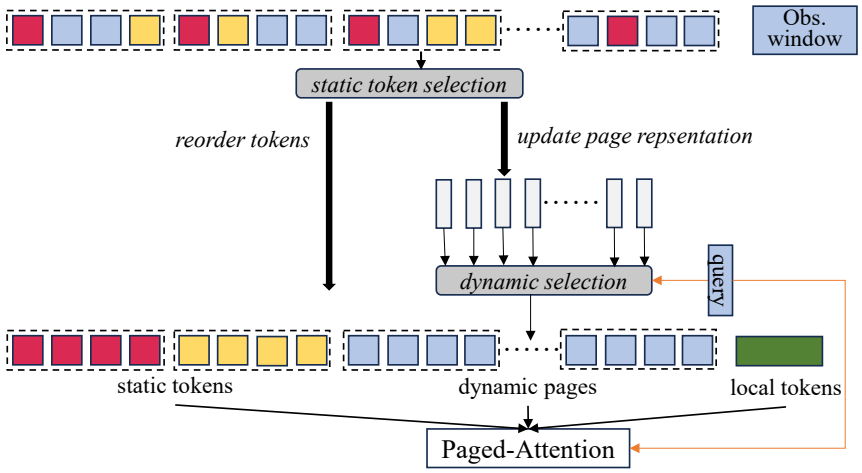


Figure 4: Algorithm design overview of PRKV.

Algorithm 1 Static Token Selection

Input: querys of observation windowQ_obs, K and V, token budget k, static ratio $\mathcal{S}$, decode step ds, update interval T
Constants: query group size gs, local window size nl
Output:
if $ds\%T \neq 0$ **then**
  return $K, V, None$
end if
# compute token importance with observation window
$K'_{pre} \leftarrow$ repeat_kv$(K[:-nl], gs)$
$S \leftarrow$ compute_score(Q_{obs}, K'_{pre})
$S \leftarrow$ Average(S)
if $\mathcal{S}$ is None **then**
  # estimate static ratio during prefill
  $S^{(1)}, S^{(T)} \leftarrow$ FirstLast(S)
  $C_h \leftarrow$ TopKOverlap$(S_h^{(1)}, S_h^{(T)}, k)$
  $C_{avg} \leftarrow$ Average(C_h)
  $\mathcal{S} \leftarrow \frac{P \cdot \lfloor C_{avg}/P \rfloor}{k}$
end if
# select static important tokens
$I_{\text{topk}} \leftarrow$ TopKIndices$(S_{mean}, k\mathcal{S})$
concat static tokens
$I_{\text{rem}} \leftarrow$ AllIndices $\setminus I_{\text{topk}}$
$P \leftarrow$ Concat$(I_{\text{topk}}, I_{\text{rem}})$
$K[:-nl] \leftarrow$ gather$(K[:-nl], P)$
$V[:-nl] \leftarrow$ gather$(V[:-nl], P)$
update page representation
$Rep_{page} \leftarrow$ update_page_representation$(K[:-nl])$
asynchronous rewrite KV
$KV_{CPU} \leftarrow Copy(K[:k], V[:k])$
return K, V, Rep_{page}

KV Budget	Qwen2.5-14B-Instruct			Llama3.1-8B-Instruct		
	32K	64K	128K	32K	64K	128K
k=512	0.25	0.25	0.25	0.625	0.5625	0.5
k=1024	0.4062	0.375	0.3438	0.6875	0.625	0.5625
k=2048	0.5	0.4844	0.4219	0.7344	0.7031	0.625
k=4096	0.6016	0.5781	0.5	0.7891	0.75	0.6875

Table 1: $\mathcal{S}$ across input lengths and KV budgets

Hybrid Sparse Attention. PRKV offloads all KV cache to CPU, keeping page representations and KV space with budget k (named $\mathbf{K}', \mathbf{V}'$) in GPU. Page representations are min and max vectors of keys in a page, allowing to approximate the maximum token importance in a page. During each decoding step, KV of static and local tokens are always in GPU. For a KV budget k, PRKV dynamically chosen a set of $\frac{k\mathcal{S}}{pagesize}$ pages and retrieval them(Section 3.2). These selected tokens' KV then participate in the sparse paged-attention computation. The detail is shown in Algorithm 2.

Algorithm 2 Hybrid Sparse Attention

Input: Q, K' and V', Page Representation P, token budget k, static ratio $\mathcal{S}$, In GPU KV pages GP
Constants: local window size nl
Output: attention out O
estimate page importance
$EScores \leftarrow QP$
$IPages \leftarrow$ TopKIndices$(EScores, k(1-\mathcal{S}))$
KV pages not in GPU
$Rpage \leftarrow IPages \setminus GP$
Retrieval pages not in GPU
$K', V' \leftarrow Retrieval(RPages)$
$O \leftarrow PagedAttention(Q, K', V')$

Periodic static token update. The importance of initially selected static tokens can decay over long decoding steps. To mitigate this, we introduce a periodic update strategy for the static token set, formalized in Algorithm 1. Given an interval T, when $ds \bmod T = 0$, we recompute the static tokens using the local window, update the page representation, and asynchronously write the updated set back to CPU memory Figure 9.

3.2 Efficient retrieval Optimization

Retrieving KV pages from CPU introduces two key overheads: inefficient indexing and data transfer. Per-head KV pages are stored non-contiguously in CPU memory, leading to costly indexing, while transferring needed pages requires multiple kernel launches and serial transfers. To mitigate these bottlenecks, we design CUDA kernels for contiguous memory indexing and batched KV transfer.

Contiguous Memory Indexing. When managing KV pages in CPU memory, two common layouts are NHD and HND, with tensor shapes of $(2, page_size, num_head, head_dim)$ and $(num_head, 2, page_size, head_dim)$, respectively. These layouts differ significantly in indexing efficiency for page retrieval, which is performed per head. Retrieving a page for head i under the NHD layout requires indexing all KV pairs within a page to copy those for head i. In contrast, the HND layout allows direct indexing of all KV pairs for a specific head without search overhead. Consequently, PRKV adopts the HND layout for both CPU and GPU KV caches, utilizing contiguous memory indexing. The efficiency of this design is demonstrated in Section 4.3.

Batched KV Transfer. As shown on the left of Figure 5, the naive page-to-page transfer mode moves KV data directly from CPU memory to GPU memory. This approach requires multiple kernel launches, which is inefficient. To address this, we introduce Batched KV Transfer (right side of Figure. 5). Our method first consolidates the required KV pages into a pinned CPU buffer, then performs a single, bulk transfer to GPU memory. Finally, the data is dispatched into the GPU's KV pool.

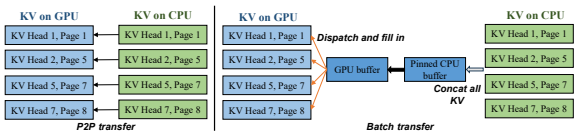


Figure 5: KV transfer optimation

4 Experiments

4.1 Accuracy Evaluation

Setup. We evaluate PRKV across various models and tasks. We conduct evaluation on LongBench V2 (Bai et al., 2024), Needle In A Haystack (Kamradt, 2023), and long reasoning tasks, including MATH500 (Hendrycks et al., 2021) and AIME24 (Jia, 2025). For LongBench V2 and NIAH, we use general models Llama-3.1-8B-Instruct(Meta Llama, 2024) and Qwen-2.5-14B-Instruct (Qwen et al., 2025). For reasoning tasks, we use Qwen3-8B and Qwen3-4B (Yang et al., 2025).

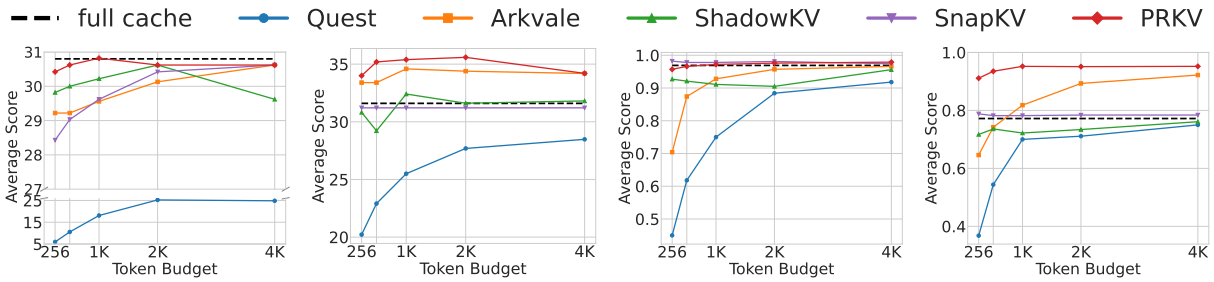
Baselines We compare PRKV against four dynamic sparse attention methods, including Quest (Tang et al., 2024), Arkvale, ShadowKV (Sun et al., 2024) and SnapKV. All methods perform dynamic sparse attention during decoding. For Quest, Arkvale and PRKV, we set their page size to 32 and local window size to 64. For PRKV the staic set ratio $\mathcal{S} = 0.25$ and update interval $T = 128$.

LongBench V2. LongBench V2 covers various difficulty levels and context lengths, ranging from 8K to 2M tokens. The average scores is shown in Figure 6a and Figure 6b. PRKV consistently achieves the highest score. While other methods experience serious performance degradation under a low KV budget, PRKV is more robust and even outperforms original full attention when tested on Qwen-2.5-14B-Instruct. The detail results of all kind of tasks are shown in Appendix B.1.

Needle In A Haystack. We evaluate retrieval accuracy on the NIAH, where a single critical sentence ("needle") is inserted into a long irrelevant context. The average results are shown in Figure 6c and Figure 6d. On Llama3.1-8B-Instruct, PRKV and SnapKV achieve the best performance, closely matching the full cache. On Qwen-2.5-14B-Instruct, PRKV consistently outperforms the full cache. Detailed heatmaps in Appendix B.2.

Reasoning Tasks. We evaluate reasoning performance on MATH500 and AIME24 using Qwen3-8B and Qwen3-4B. The maximum generation length is set to 32K with temperature 0.6 and top-p 0.95. Following prior work (DeepSeek-AI et al., 2025), we report pass@k and avg@k (Chen et al., 2021), with $k = 4$. All KV retrieval methods use a KV budget of 1024. Since SnapKV is originally designed for prefilling, we adapt it to decoding using same static token update interval as our method.

As shown in Table 2, PRKV achieves accuracy comparable to the full KV cache and consistently



(a) (b) (c) (d)

Figure 6: Average score of PRKV and baselines. (a) LongBench V2 on Llama3.1-8B-Instruct. (b) LongBench V2 on Qwen-2.5-14B-Instruct. (c) NIAH on Llama3.1-8B-Instruct. (d) NIAH on Qwen-2.5-14B-Instruct.

Methods	AIME24		MATH500	
	pass@k	*avg@k*	*pass@k*	*avg@k*
Qwen3-4B	86.67	74.17	95.40	94.20
Quest	70.00	56.67	84.80	78.50
ArkVale	76.67	62.50	85.40	84.30
ShadowKV	76.67	59.17	85.00	81.10
SnapKV	83.33	67.50	93.60	90.00
PRKV	**86.67**	**75.00**	**95.00**	**93.80**
Qwen3-8B	83.33	74.17	96.20	95.10
Quest	76.67	65.83	86.80	78.60
ArkVale	80.00	68.33	87.00	86.90
ShadowKV	76.67	67.50	86.80	82.60
SnapKV	**83.33**	67.50	93.60	93.60
PRKV	**83.33**	**73.33**	**95.40**	**94.40**

Table 2: Accuracy results of long reasoning tasks.

outperforms other compression methods across all datasets, demonstrating the effectiveness of our hybrid KV selection strategy.

4.2 Efficiency Evaluation

The efficiency experiments were conducted on an Nvidia A100 80GB GPU connected with Intel Xeon Silver 4310 CPU via PCIe Gen4. We compare PRKV with offloading methods Arkvale and ShadowKV under long-input and long-generation scenarios. We set $k = 1024$. The end-to-end latency is in the left of Figure 7. PRKV achieves a multi-fold speedup over Arkvale and ShadowKV across multiple models, demonstrating significant efficiency and achieving maximum speedup of up to up to $\mathbf{6.75}\times$ compared to ArkVale.

4.3 Ablation Results

System optimization. Since PRKV's system optimizations including contiguous memory indexing (CI) and batched KV transfer (BT) are designed to reduce recall latency, we evaluate the Time Per Output Token (TPOT) using Llama3.1-8B-Instruct and Qwen2.5-14B-Instruct with 512 generated tokens. As shown right of Figure 7, contiguous memory indexing achieve $\mathbf{1.5}\times$ reduction in TPOT. Batched KV transfer contributes most to the improvements, achieving up to a $\mathbf{7.7}\times$ TPOT reduction.

Attention breakdown. Figure 8 presents Attention breakdown of Full Attention and PRKV across sequence lengths from 32K to 128K with batchsize=4. While PRKV introduces minor computational overheads from criticality estimation and top-K filtering, its sparse attention significantly reduces latency. When the sequence length reaches 128K, Full Attention requires over 500 ms to retrieve the KV cache from CPU, whereas PRKV takes only 1 ms to retrieve important KV pages.

Update interval. We study the impact of the static token update interval T on the attention recall rate. As shown in Table 3, smaller update intervals consistently yield higher recall across models and input lengths, indicating that more frequent updates better track the evolution of token importance during decoding. However, performance degrades precipitously when $T > 128$, especially under long-context settings. Therefore, we set $T = 128$ as our default configuration to achieve an optimal balance between attention accuracy and update overhead.

T	Qwen2.5-14B-Instruct			Llama3.1-8B-Instruct		
	32K	64K	128K	32K	64K	128K
$T = 32$	0.947	0.933	0.919	0.956	0.935	0.913
$T = 64$	0.939	0.927	0.911	0.943	0.929	0.910
$T = 128$	0.927	0.915	0.902	0.934	0.916	0.901
$T = 256$	0.819	0.773	0.721	0.804	0.758	0.6875

Table 3: Update interval T ablation. Attention Recall Rate of different T across input lengths and models

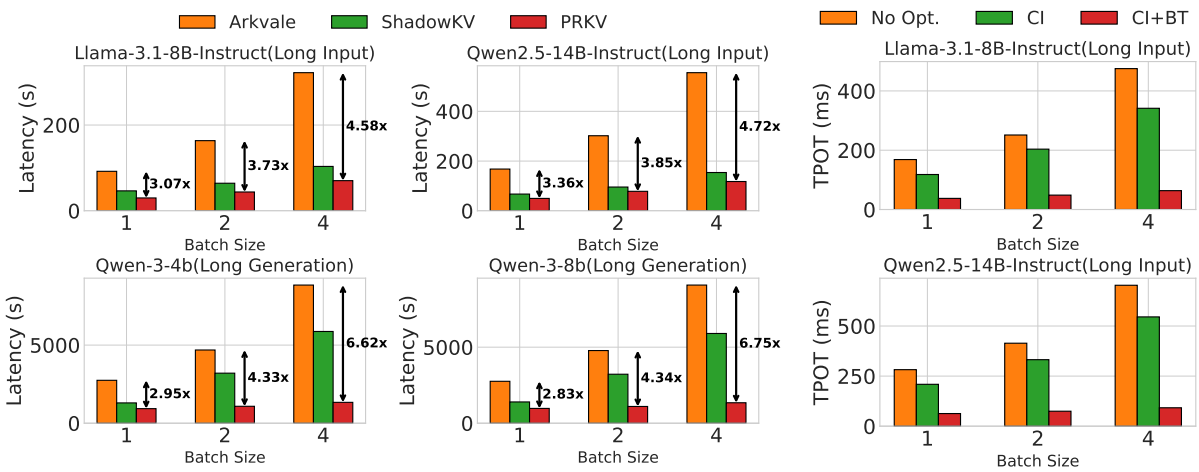


Figure 7: **Left:** End-to-end latency under long-input and long-generation scenarios. **Right:** TPOT of different system optimization designs.

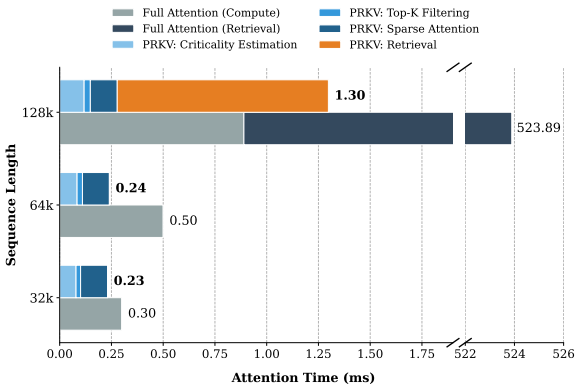


Figure 8: PRKV self-attention time breakdown compared to FlashInfer Full Attention.

5 Related Work

The KV cache enables efficient autoregressive inference but incurs substantial memory overhead as context length grows. Prior work mitigates this issue through three main directions: quantization, eviction, and retrieval-based sparse attention.

KV Cache Quantization. Quantization methods reduce KV memory footprint by approximating high-precision values with low-bit representations (Liu et al., 2024; Zhang et al., 2024; Hooper et al., 2025). For example, KIVI (Liu et al., 2024) adopts asymmetric quantization, applying channel-wise quantization for keys and token-wise quantization for values. These approaches lower storage cost and are orthogonal to sparse attention methods.

KV Cache Eviction. Eviction-based approaches maintain a fixed-size KV cache by retaining important tokens and discarding others. StreamingLLM (Xiao et al., 2024b) preserves sink tokens and a local window, while SnapKV (Li et al., 2024) identifies important tokens using an observation window at the prompt end. H2O (Zhang et al., 2023) estimates token importance from accumulated historical attention scores. However, premature eviction may degrade accuracy, as tokens initially deemed unimportant can become relevant later in decoding.

Retrieval-based Sparse Attention. Retrieval methods retain the full KV cache in CPU memory and dynamically select important KV pairs for attention computation. *Token-level retrieval* methods estimate token importance directly. RetrievalAttention (Liu et al., 2025a) and MagicPIG (Chen et al., 2025b) employ approximate nearest neighbor search and locality-sensitive hashing on CPU, respectively. InfiniGen (Lee et al., 2024) computes partial query–key interactions on GPU to guide retrieval. While precise, their importance estimation cost scales linearly with context length. *Group-level retrieval* methods operate on token groups with precomputed metadata. Quest, Arkvale, and InfLLM (Tang et al., 2024; Chen et al., 2024; Xiao et al., 2024a) partition sequences into fixed-size pages, whereas ClusterKV (Liu et al., 2025c) and RetroInfer (Chen et al., 2025a) form semantic clusters via k-means.

6 Conclusion

We propose PRKV, a hybrid sparse attention framework for efficient page-level KV retrieval with offloading. By jointly improving selection accuracy and retrieval efficiency, PRKV enables scalable long-context inference with consistent accuracy gains and substantial decoding speedups.

7 Limitations

First, the method introduces additional hyperparameters(e.g., update interval),which may require re-tuning across models, context lengths, and hardware platforms, and the best configuration may not transfer reliably. Second, our evaluation primarily focuses on the decode phase. Finally, our experiments are conducted on a specific GPU and software stack; performance and memory behavior may vary across other accelerators and deployment environments.

References

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *arXiv preprint arXiv:2412.15204*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. 2024. Arkvale: Efficient generative LLM inference with recallable key-value eviction. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Jing Liu, Bailu Ding, Xiao Yan, Jiawei Jiang, Chen Chen, Mingxing Zhang, Yuqing Yang, Fan Yang, and Mao Yang. 2025a. Retroinfer: A vector-storage approach for scalable long-context llm inference. *Preprint*, arXiv:2505.02922.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. 2025b. MagicPIG: LSH sampling for efficient LLM generation. In *The Thirteenth International Conference on Learning Representations*.

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

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Wei Gao, Xinyu Zhou, Peng Sun, Tianwei Zhang, and Yonggang Wen. 2025. Rethinking key-value cache compression techniques for large language model serving. *Preprint*, arXiv:2503.24000.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *Preprint*, arXiv:2103.03874.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2025. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Preprint*, arXiv:2401.18079.

Maxwell Jia. 2025. Maxwell-jia/aime_2024. `https://huggingface.co/datasets/Maxwell-Jia/AIME_2024`. Accessed: 2025-09-16.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *Preprint*, arXiv:2406.00515.

Greg Kamradt. 2023. Needle in a Haystack: Pressure Testing LLMs. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`. Accessed: 2025-09-16.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. Infinigen: efficient generative inference of large language models with dynamic kv cache management. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, OSDI'24. USENIX Association.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. SnapKV: LLM knows what you are looking for before generation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. 2025a. Retrievalattention: Accelerating long-context LLM inference via vector retrieval.

Guangda Liu, Chengwei Li, Zhenyu Ning, Minyi Guo, and Jieru Zhao. 2025b. Freekv: Boosting kv cache retrieval for efficient llm inference. *Preprint*, arXiv:2505.13109.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. 2025c. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. *Preprint*, arXiv:2412.03213.

Xiang Liu, Zhenheng Tang, Hong Chen, Peijie Dong, Zeyu Li, Xiuze Zhou, Bo Li, Xuming Hu, and Xiaowen Chu. 2025d. Can llms maintain fundamental abilities under kv cache compression? *Preprint*, arXiv:2502.01941.

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

Meta Llama. 2024. Llama-3.1-8B-Instruct.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

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

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2024. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: query-aware sparsity for efficient long-context llm inference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Minzheng Wang, Longze Chen, Cheng Fu, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, Yunshui Li, Min Yang, Fei Huang, and Yongbin Li. 2024. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa. *Preprint*, arXiv:2406.17419.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024a. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *Preprint*, arXiv:2502.11089.

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. In *MLSys*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

A Picture of Observation

Figure 10 shows the important token selection distribution of ground truth topk and Arkvale on a 37K NIAH task, using a 1K token KV budget. The result is from layer 5 of the 14-th generation token.

B Detail results in Evalution

In this section, we provide detail results in evalution.

B.1 LongBench V2

We report accuracy under the context length categories of short, medium and long, which has easy and hard two level problems, as well as the overall accuracy. As shown in Table 4 and Table 7

B.2 NIAH

We show the detail results of NIAH across Llama-3.1-8B-Instruct and Qwen-2.5-14B-Instruct. We test the KV budget from 256, 512, 1024, 2048, 4096. As shown form Figure 12 to Figure 20

C NHD and HND KV layouts GPU efficiency

As shown in Table 6

D Additional evalution on Qwen2.5-32B-Ins

In this section, we provide an additional evaluation on Qwen2.5-32B-Instruct using two A100 GPUs, which represents a larger-scale model. As shown in Table 7 and Figure 21, 22, 23.

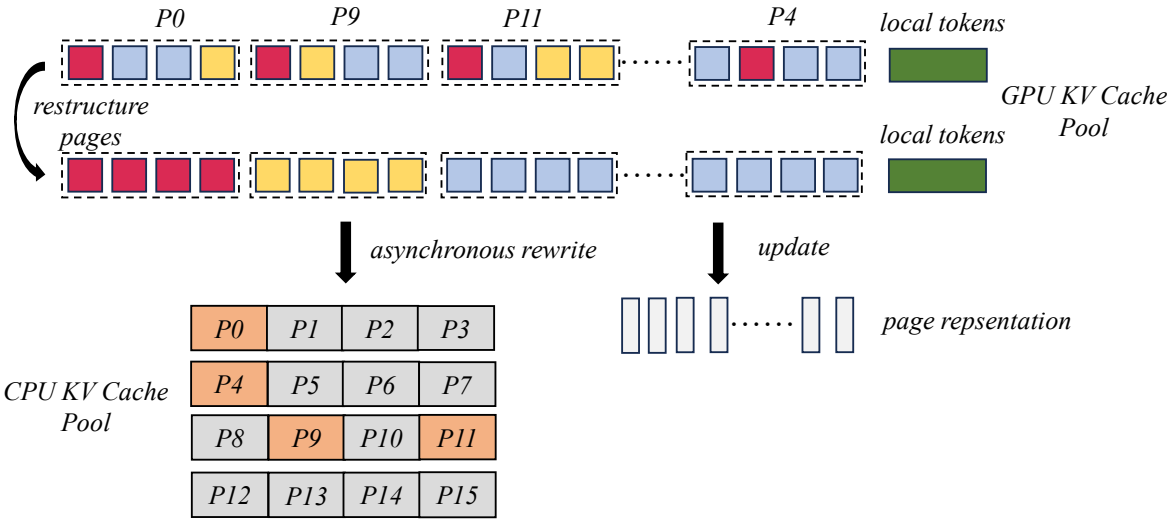


Figure 9: Detail design of recomputes the static set and restructures pages

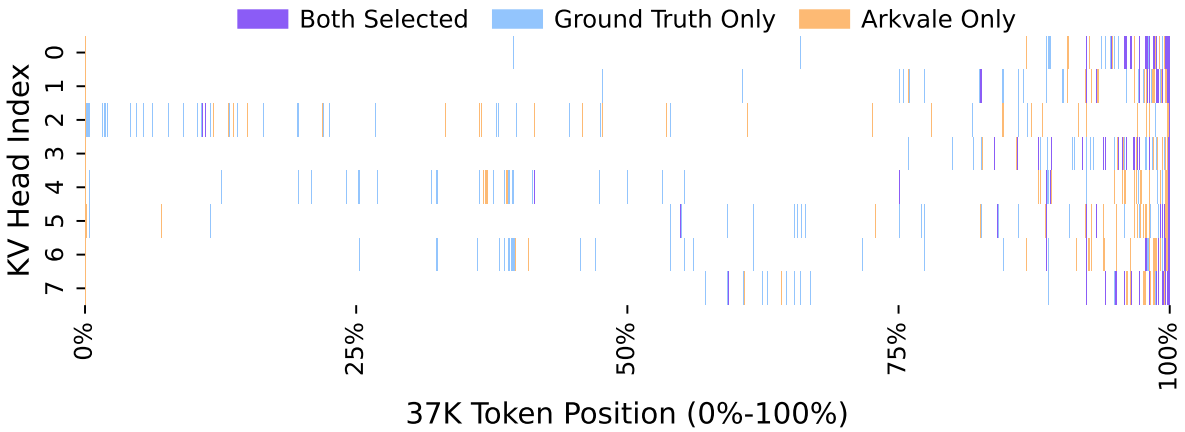


Figure 10: Important token selection distribution of ground truth top-k and Arkvale on a NIAH task.

Table 4: Accuracy results of LongBench v2

		Overall	**Easy**	**Hard**	**Short**	**Medium**	**Long**
Llama-3.1-8B-Instruct (base)		30.8	30.7	30.9	38.3	27.0	25.9
k=4096	Quest	24.85	22.92	26.05	30.0	21.86	22.22
	ArkVale	30.62	28.65	31.83	36.11	27.91	26.85
	ShadowKV	29.62	29.17	29.9	33.89	28.84	24.07
	SnapKV	30.42	28.65	31.51	33.89	30.23	25.0
	PRKV	**30.62**	28.12	32.15	36.67	28.84	24.07
k=2048	Quest	25.25	22.92	26.69	31.11	23.72	18.52
	ArkVale	30.13	31.49	29.27	36.05	27.04	26.0
	ShadowKV	30.62	29.69	31.19	36.11	28.37	25.93
	SnapKV	30.42	28.12	31.83	36.67	28.37	24.07
	PRKV	**30.62**	28.65	31.83	37.22	28.84	23.15
k=1024	Quest	18.09	19.27	17.36	18.89	17.67	17.59
	ArkVale	29.56	31.29	28.51	37.12	25.43	26.19
	ShadowKV	30.22	27.6	31.83	33.9	29.3	25.9
	SnapKV	29.62	28.12	30.55	35.56	27.44	24.07
	PRKV	**30.82**	30.21	31.19	35.56	29.77	25.0
k=512	Quest	10.54	9.9	10.93	12.22	9.77	9.26
	ArkVale	29.22	28.65	29.58	33.89	27.91	24.07
	ShadowKV	30.0	27.6	31.5	34.4	28.4	25.9
	SnapKV	29.03	28.12	29.58	36.67	26.05	22.22
	PRKV	**30.62**	31.25	30.23	36.11	29.77	23.15
k=256	Quest	5.96	3.65	7.4	8.33	5.58	2.78
	ArkVale	29.22	27.6	30.23	33.89	27.91	24.07
	ShadowKV	29.82	29.17	30.23	36.67	28.37	21.3
	SnapKV	28.43	25.52	30.23	34.44	25.58	24.07
	PRKV	**30.42**	28.65	31.51	33.89	30.23	25.0

Table 5: Accuracy results of LongBench v2

		Overall	**Easy**	**Hard**	**Short**	**Medium**	**Long**
Qwen-2.5-14B-Instruct (base)		31.6	35.4	29.3	37.2	30.2	25.0
k=4096	Quest	28.48	30.9	26.95	33.73	25.25	26.04
	ArkVale	34.19	41.15	29.9	37.78	34.88	26.85
	ShadowKV	31.81	33.85	30.55	41.67	27.44	24.07
	SnapKV	31.21	33.33	29.9	38.33	29.3	23.15
	PRKV	**34.2**	40.1	30.5	38.9	34.4	25.9
k=2048	Quest	27.69	28.27	27.33	35.75	23.26	23.15
	ArkVale	34.39	41.67	29.9	38.89	34.42	26.85
	ShadowKV	31.61	34.9	29.58	39.44	28.37	25.0
	SnapKV	31.21	33.33	29.9	38.33	29.3	23.15
	PRKV	**35.59**	42.71	31.19	38.89	35.81	29.63
k=1024	Quest	25.5	25.65	25.4	29.61	26.98	15.74
	ArkVale	34.59	40.62	30.87	39.44	34.42	26.85
	ShadowKV	32.41	35.94	30.23	39.44	29.77	25.93
	SnapKV	31.21	33.33	29.9	38.33	29.3	23.15
	PRKV	**35.39**	43.75	30.23	38.33	35.81	29.63
k=512	Quest	22.91	26.18	20.9	26.82	23.26	15.74
	ArkVale	33.4	38.02	30.55	37.22	33.49	26.85
	ShadowKV	29.22	31.77	27.65	36.11	27.91	20.37
	SnapKV	31.21	33.33	29.9	38.33	29.3	23.15
	PRKV	**35.19**	41.67	31.19	39.44	34.42	29.63
k=256	Quest	20.22	24.72	17.38	22.89	20.2	15.62
	ArkVale	33.4	40.62	28.94	38.33	33.02	25.93
	ShadowKV	30.82	33.33	29.26	38.89	28.37	22.22
	SnapKV	31.21	33.33	29.9	38.33	29.3	23.15
	PRKV	**34.0**	42.19	28.94	36.11	34.42	29.63

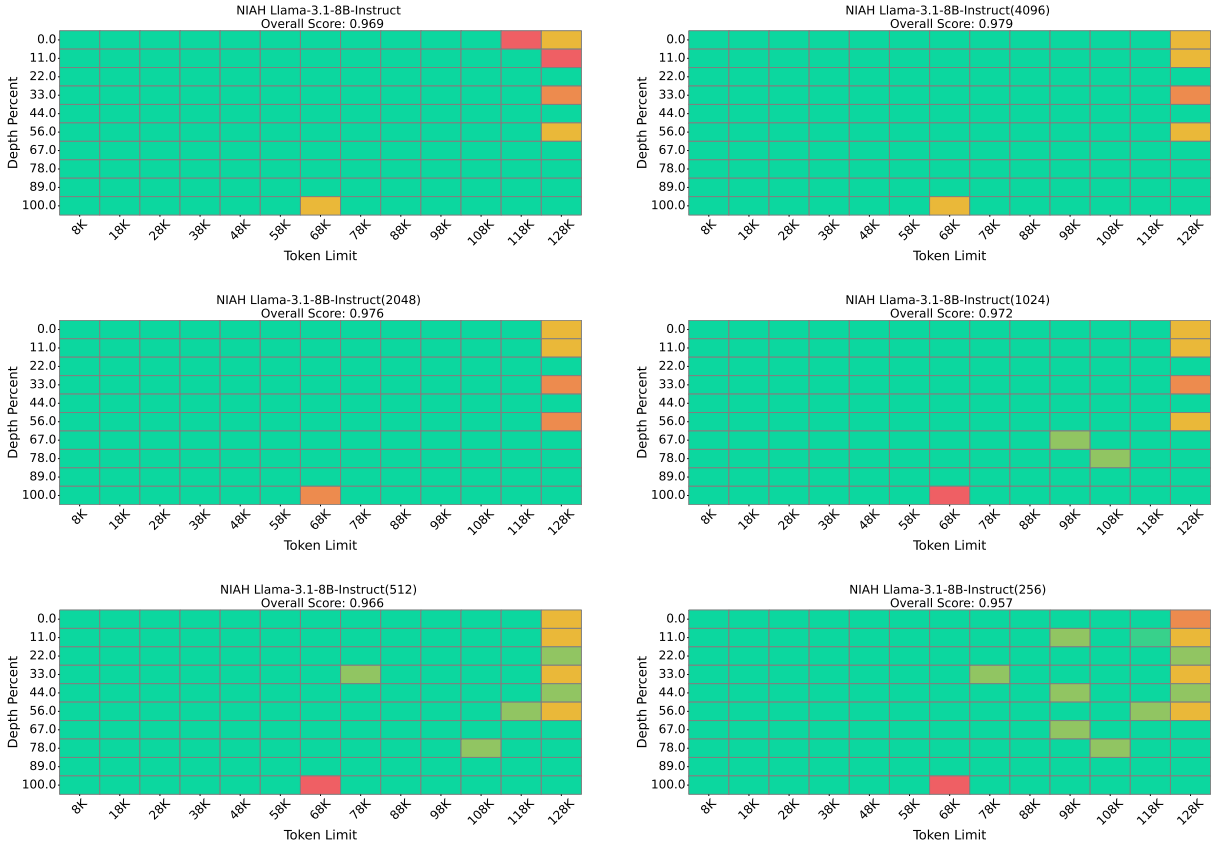


Figure 11: Compare all KV budget in PRKV with full cache in Llama-3.1-8B-Instruct

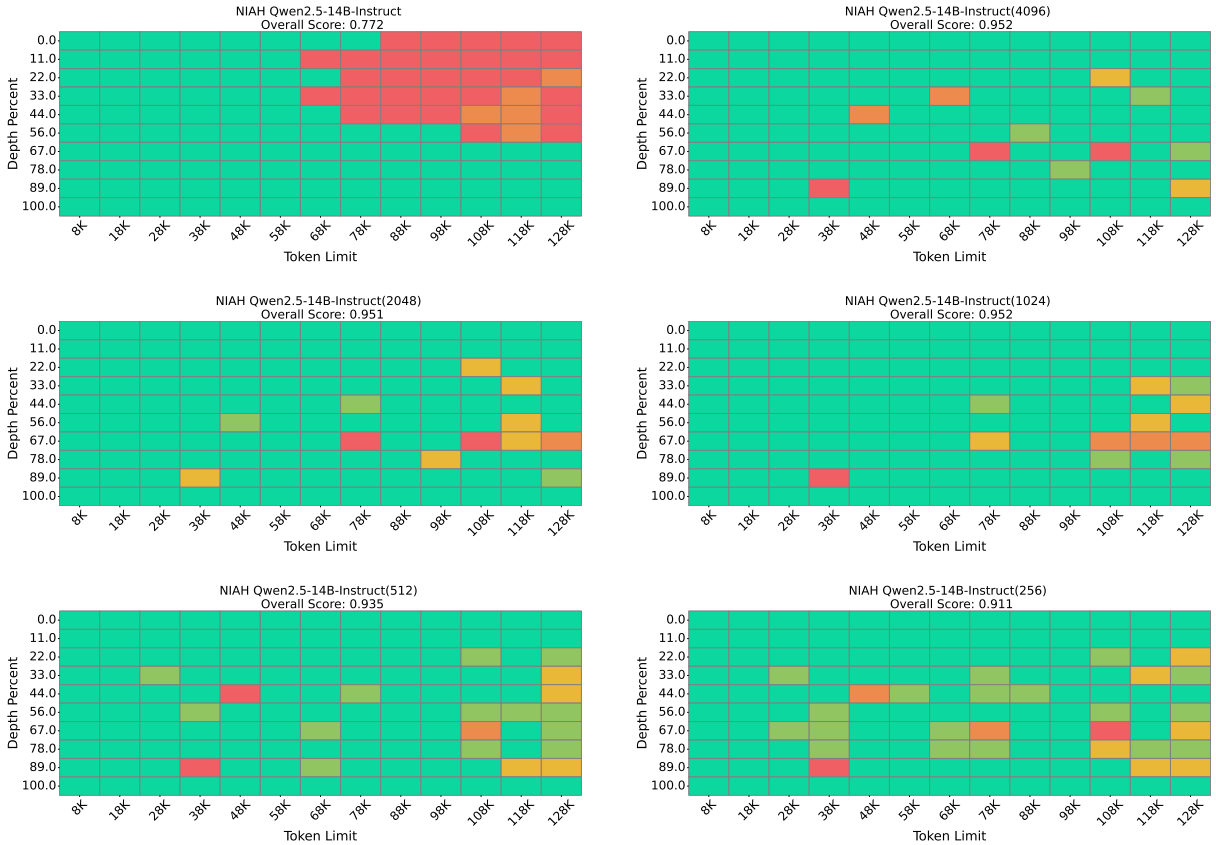


Figure 12: Compare all KV budget in PRKV with full cache in Qwen2.5-14B-Instruct

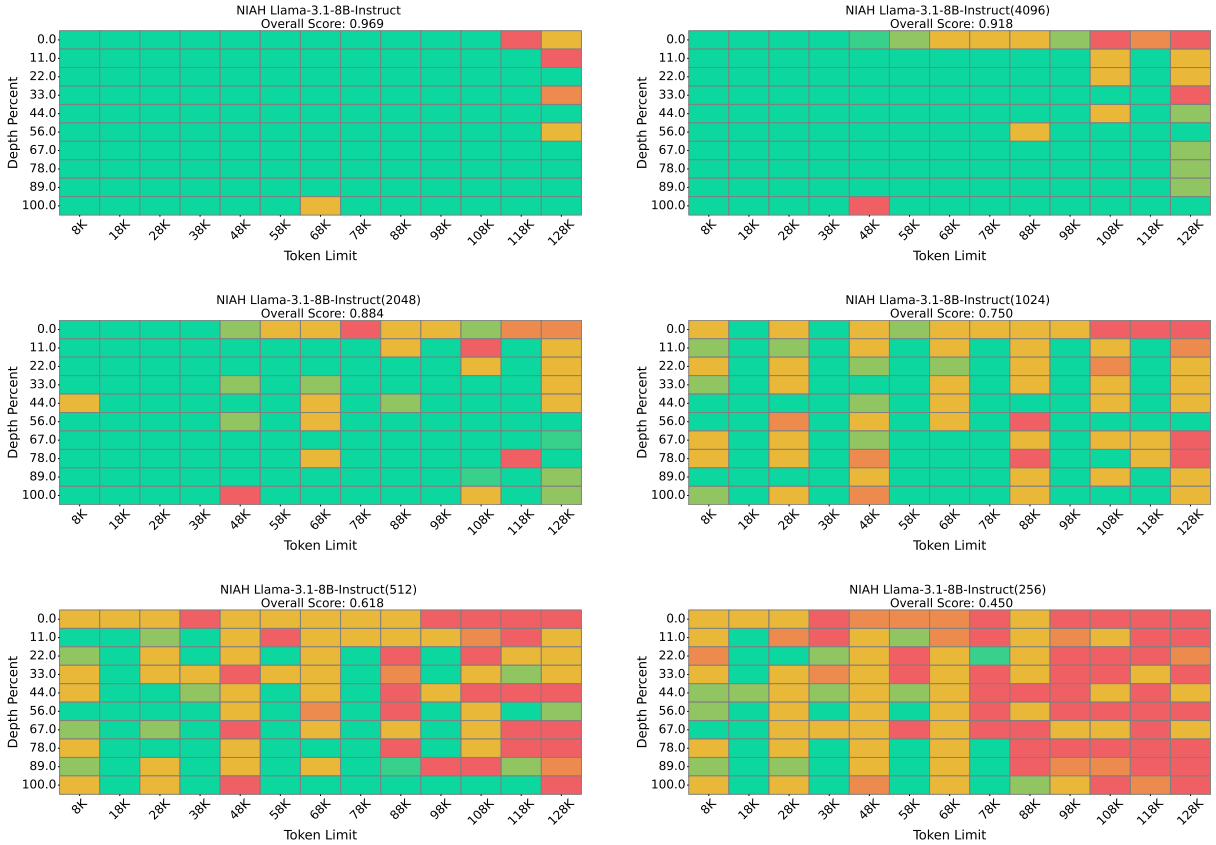


Figure 13: Compare all KV budget in Quest with full cache in Llama-3.1-8B-Instruct

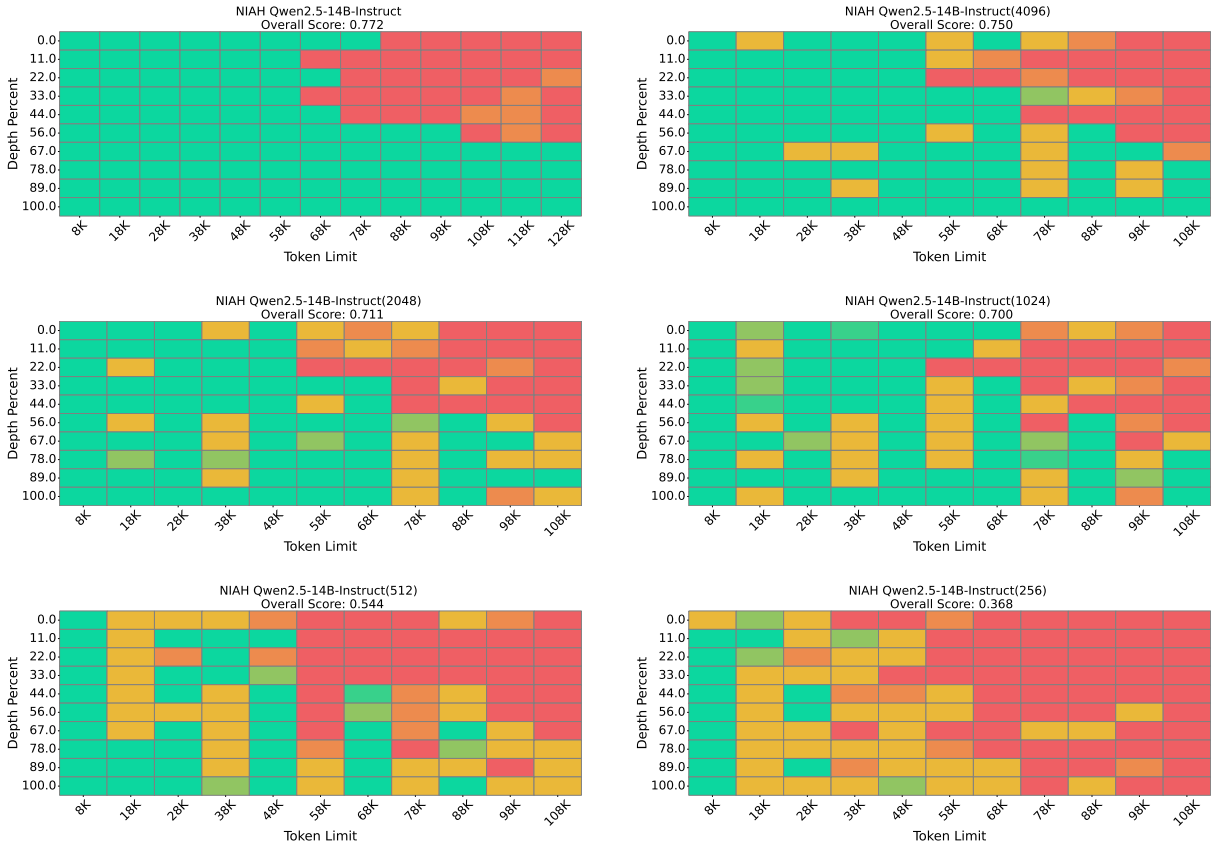


Figure 14: Compare all KV budget in Quest with full cache in Qwen2.5-14B-Instruct

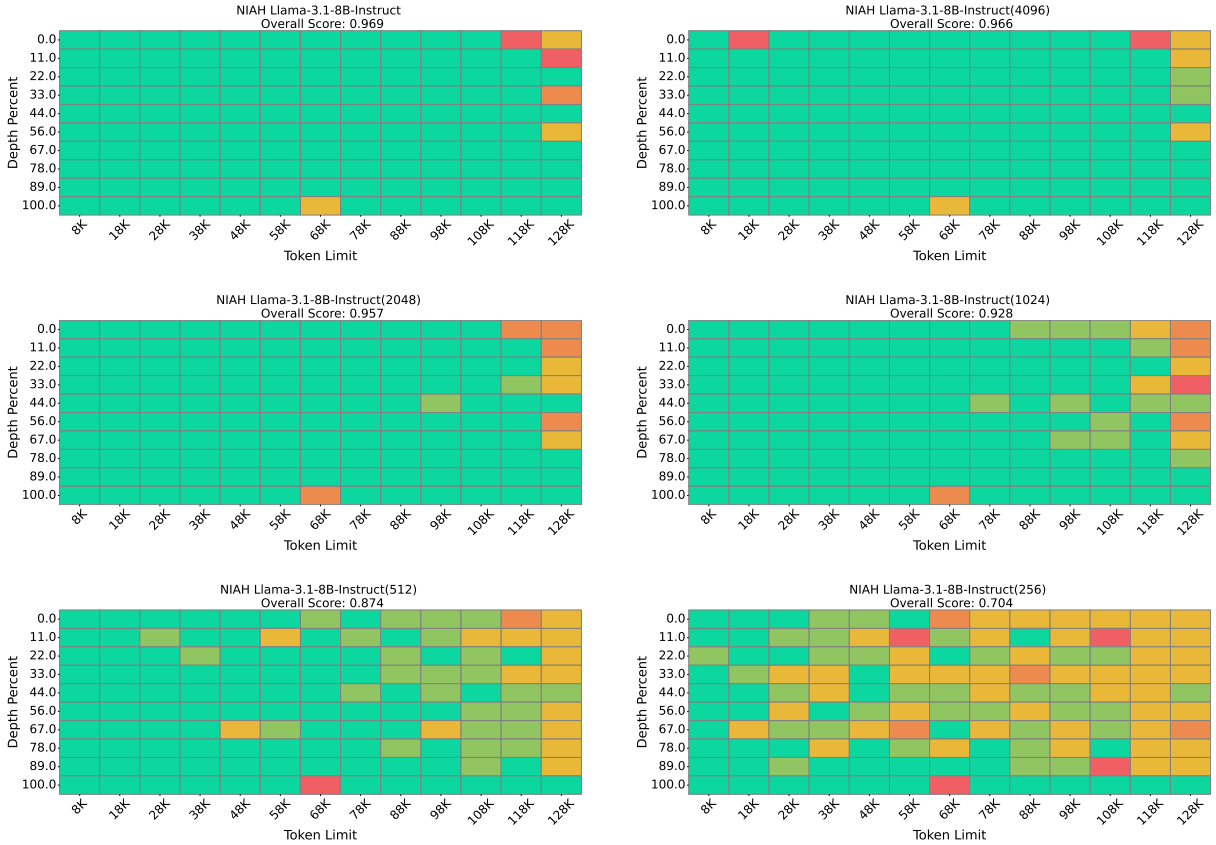


Figure 15: Compare all KV budget in Arkvale with full cache in Llama-3.1-8B-Instruct

Figure 16: Compare all KV budget in Arkvale with full cache in Qwen2.5-14B-Instruct

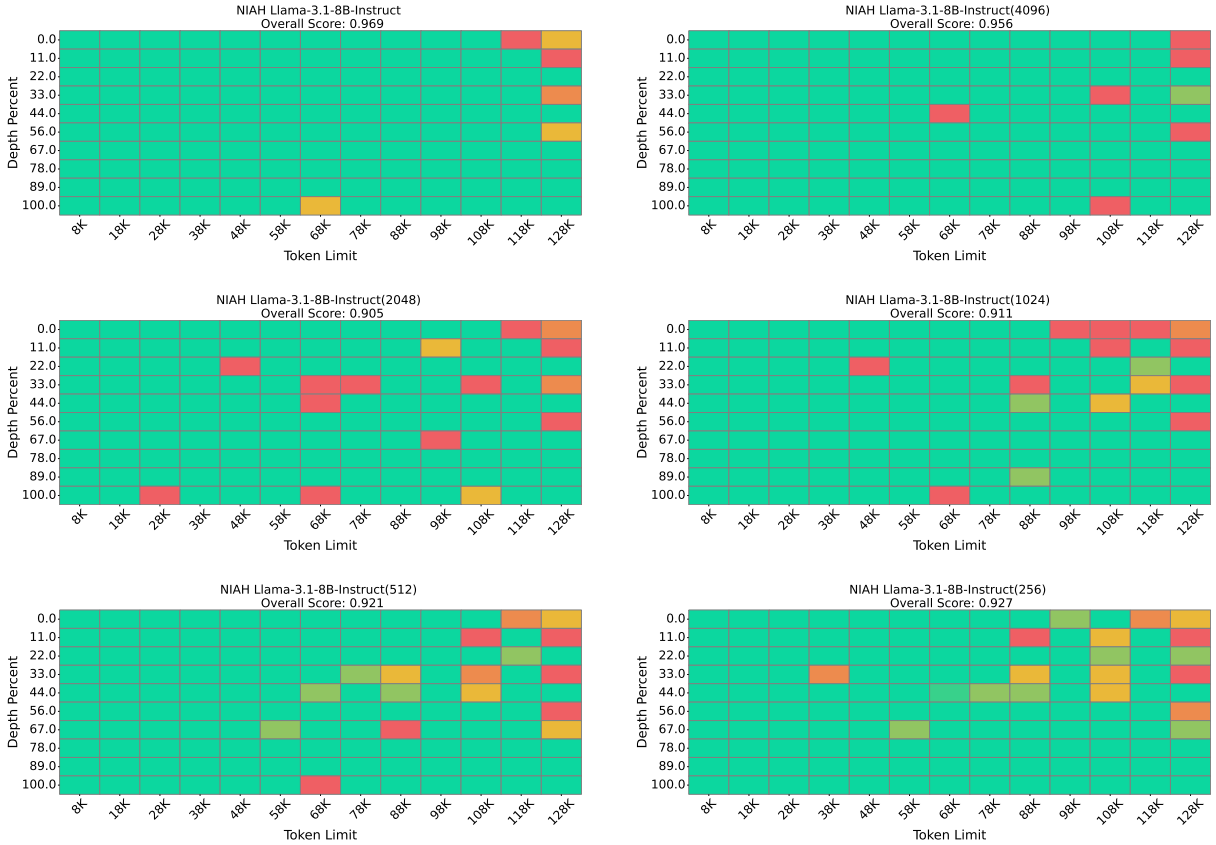


Figure 17: Compare all KV budget in ShadowKV with full cache in Llama-3.1-8B-Instruct

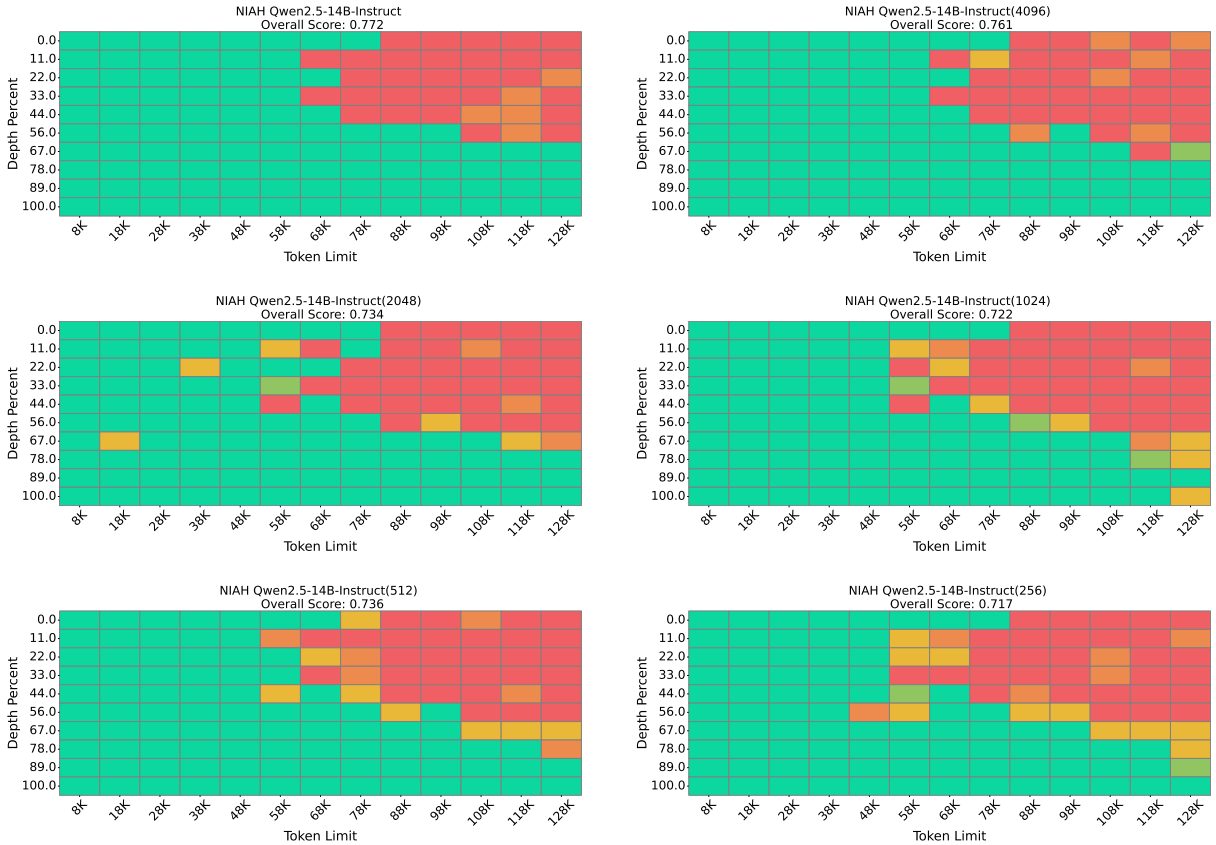


Figure 18: Compare all KV budget in ShadowKV with full cache in Qwen2.5-14B-Instruct

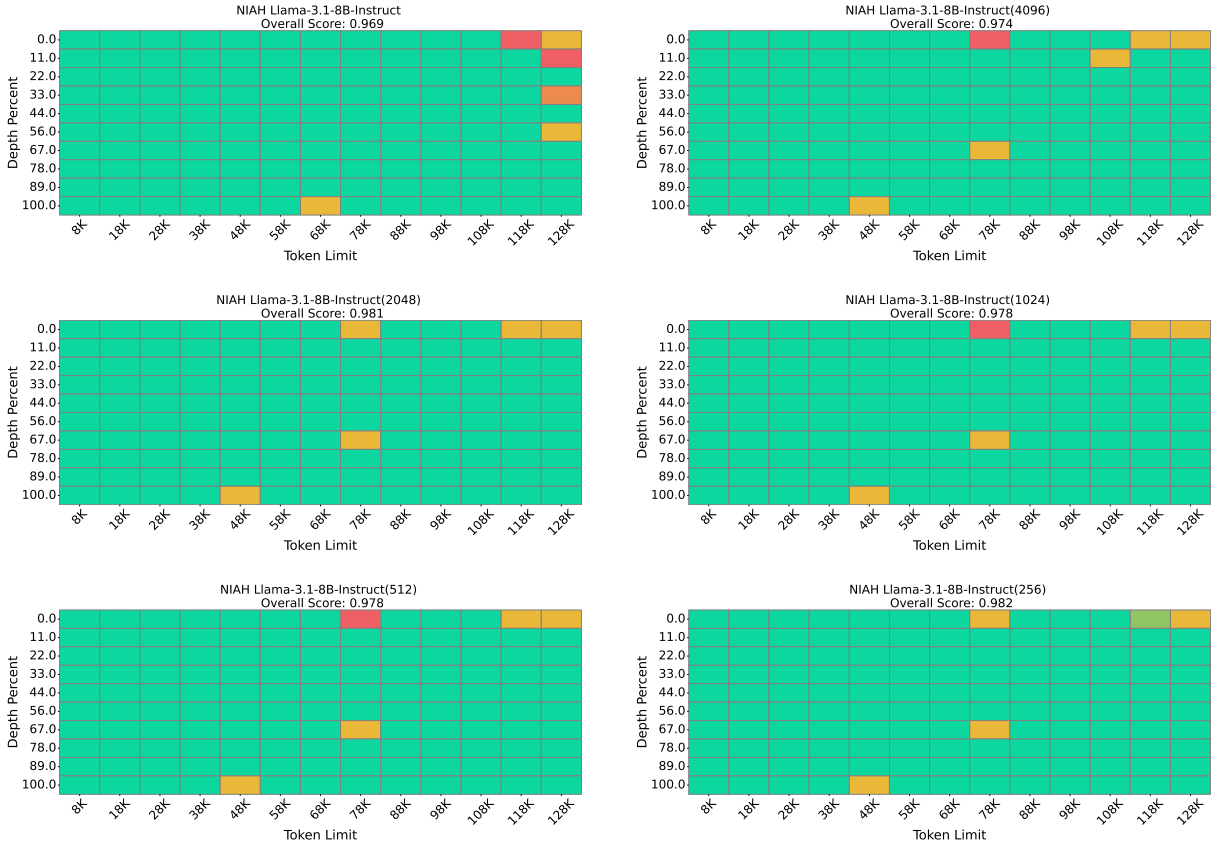


Figure 19: Compare all KV budget in SnapKV with full cache in Llama-3.1-8B-Instruct

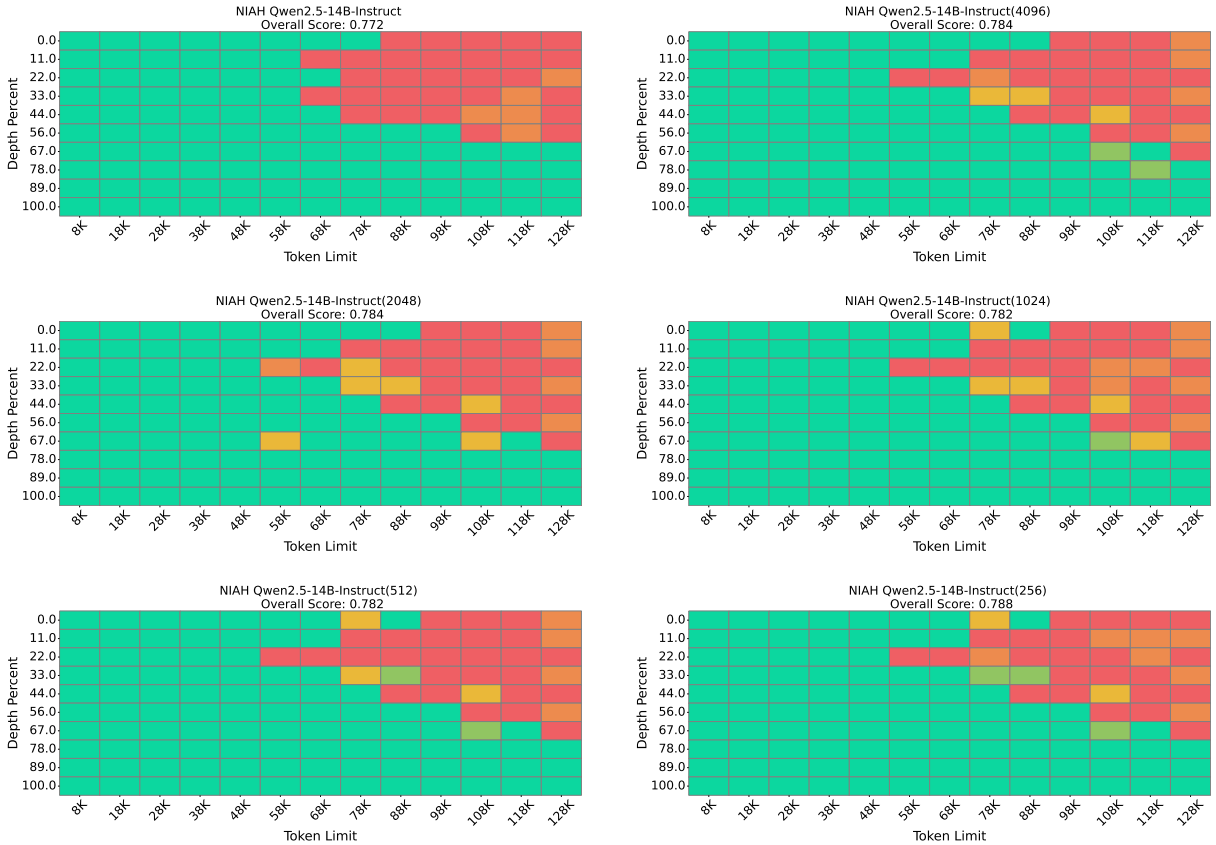


Figure 20: Compare all KV budget in SnapKV with full cache in Qwen2.5-14B-Instruct

Table 6: Latency comparison of NHD vs. HND

KV Size	Prefill NHD (ms)	Prefill HND (ms)	Decode NHD (ms)	Decode HND (ms)
4,096	1.0358	1.2567	0.0359	0.0320
8,192	4.9863	4.8339	0.0331	0.0320
16,384	19.1479	19.0742	0.0568	0.0565
32,768	76.4648	77.2803	0.0971	0.0956
65,536	325.6366	317.4488	0.1809	0.1777
131,072	1548.4492	1546.6578	0.3658	0.3981

Table 7: Larger size model results of LongBench v2

		Overall	**Easy**	**Hard**	**Short**	**Medium**	**Long**
Qwen-2.5-32B-Instruct (base)		39.36	40.62	38.59	42.22	38.6	36.11
	Quest	35.39	30.9	26.95	33.73	25.25	26.04
	ArkVale	37.77	38.54	37.3	41.11	37.67	32.41
k=4096	PRKV	**38.77**	40.1	37.94	42.22	38.14	34.26
	Quest	33.8	35.42	32.8	37.78	33.95	26.85
	ArkVale	37.38	38.02	36.98	41.11	36.74	32.41
k=2048	PRKV	**37.77**	38.54	37.3	41.11	37.21	33.33
	Quest	33.4	34.9	32.48	37.78	33.02	26.85
	ArkVale	36.18	36.98	35.69	41.11	34.88	30.56
k=1024	PRKV	**36.78**	36.98	36.66	41.11	35.81	31.48

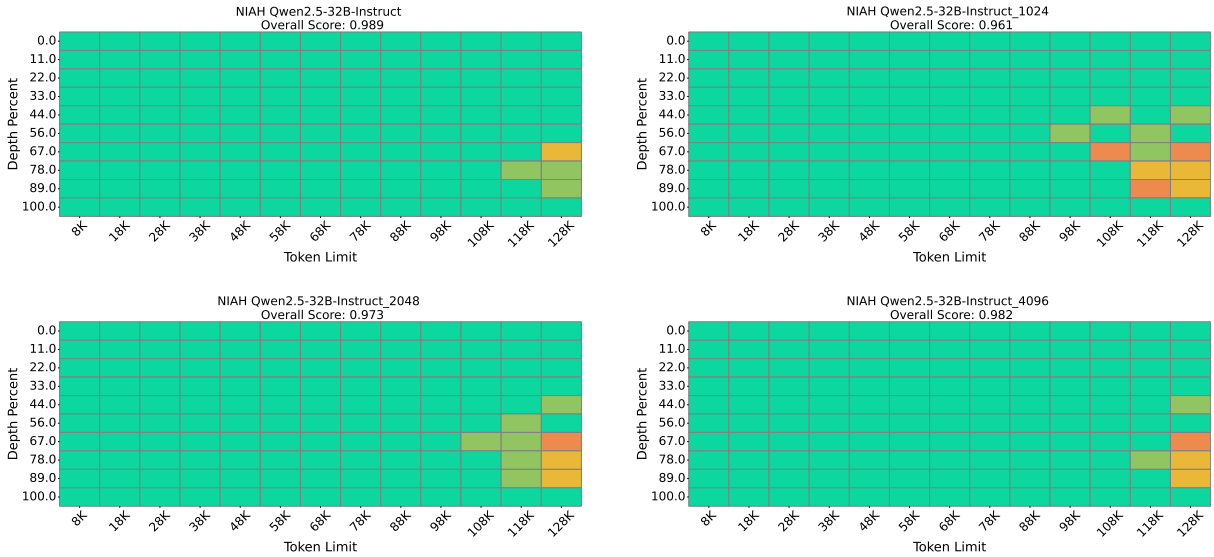


Figure 21: Compare all KV budget in PRKV with full cache in Qwen2.5-32B-Instruct

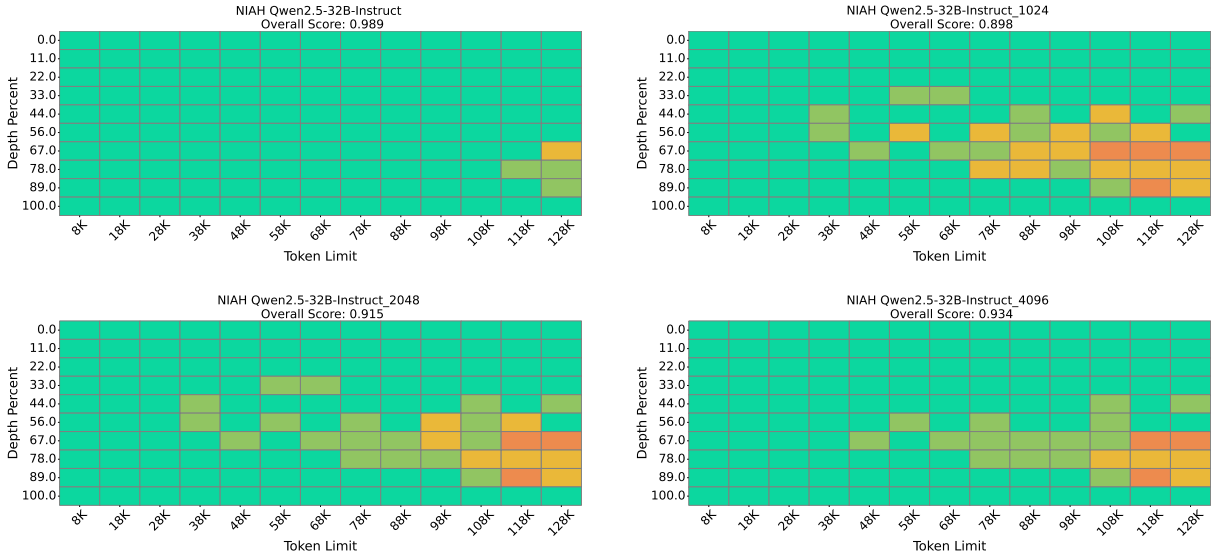


Figure 22: Compare all KV budget in Quest with full cache in Qwen2.5-32B-Instruct

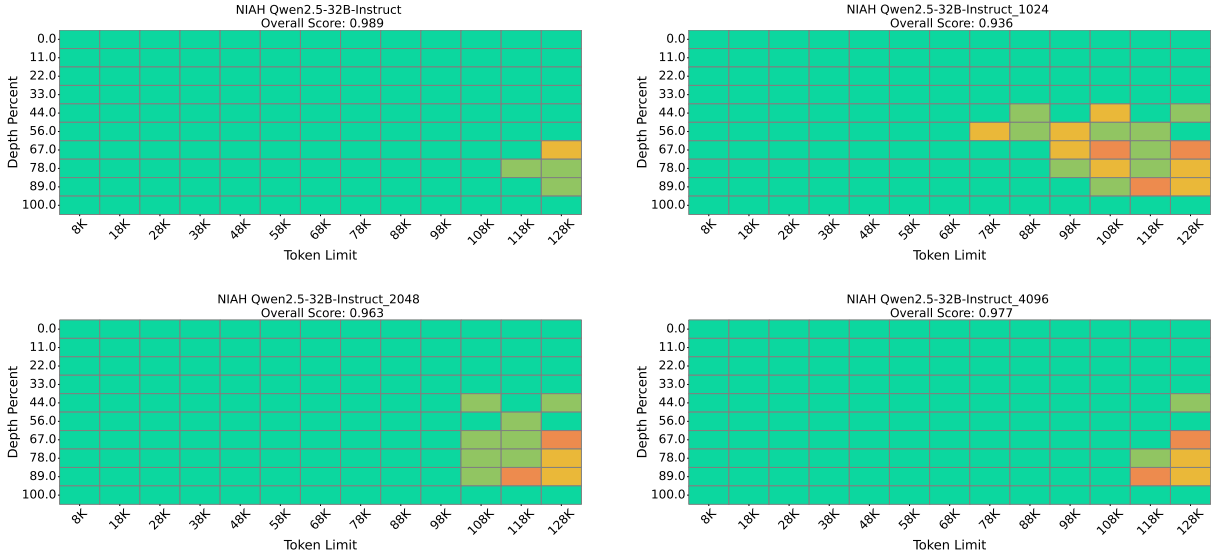


Figure 23: Compare all KV budget in Arkvale with full cache in Qwen2.5-32B-Instruct