HitKV: Activation Frequency Knows Which Tokens Are Important

Sanle Zhao[1], Yujuan Tan[2*], Yu Jing[1], Zhuoxin Bai[1], Yue Niu[1], Jiayi Guo[1], Zongjie Wang[1], Ao Ren[1*]

[1]Chongqing University, China
[2]National University of Defense Technology, China
tanyujuan@nudt.edu.cn, {ren.ao, sanlezhao}@cqu.edu.cn

Abstract

The demand for long-context processing in large language models (LLMs) continues to escalate alongside rapid advancements in their capabilities. However, the intermediate attention keys and values (KV cache) employed to avoid recomputations, also grow linearly with sequence length, far exceeding the memory capacity of consumer-grade GPUs. Consequently, many studies have proposed KV cache compression methods that evict unimportant tokens based on variant attention scoring strategies. These methods typically retain the KV pairs of the top-k scoring tokens under a fixed memory budget. However, they still face several limitations. First, they disregard the activation frequency of tokens, specifically the count of times tokens achieve top-k scores in the attention distribution of following tokens. The methods based on variant attention scores may incorrectly evict some high-activation-frequency yet low final-scoring tokens. Second, the activation frequency exhibits different distribution patterns across layers and tasks. Neglecting these differences negatively impacts model performance and task adaptability. Our analysis of the actual token activation frequency and its unique characteristics across layers and task types reveals potential opportunities to address these issues. In this paper, we propose HitKV, which employs hit rates to directly characterize token activation frequencies, enabling adaptive layer-aware and task-aware KV cache eviction under the uniform memory allocation strategies. Also, HitKV can be easily integrated into layer-specific memory allocation methods. Experimental results demonstrate that HitKV maintains model performance with preserving only 3% of the KV cache, achieves high-quality generation outputs in long-text generation tasks, and delivers $4\times$ throughput improvement over baselines.

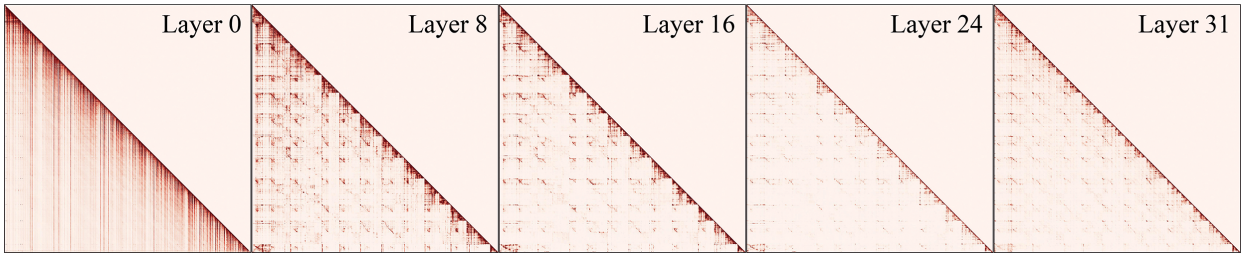
Introduction

Long-context requests for Large language models (LLMs) (Zhao et al. 2023) has become increasingly prominent, such as multi-turn dialogues (Yi et al. 2024), text summarization (Zhang et al. 2025), information retrieval (Dai et al. 2024). The long-context processing capabilities of LLMs have also evolved from several hundred tokens (Radford et al. 2018; Devlin et al. 2019) to a prevailing standard of 16K-128K tokens (OpenAI et al. 2024; GLM et al. 2024; Chang et al. 2024; Grattafiori et al. 2024), with Claude (Anthropic 2024) and Yi (AI et al. 2024) extending sequence lengths to 200K or even infinite sequence (Munkhdalai, Faruqui, and Gopal 2024; Wu et al. 2025). However, the Key-Value(KV) cache, which stores attention keys and values to avoid re-computation, also increases linearly with sequence length and quickly dominates memory usage (Chang et al. 2024; Wu et al. 2025; Li et al. 2025; Shi et al. 2024). For example, in OPT-175B, with a batch size of 16, 512-token input and 1024-token output, the KV cache reaches 1.2 TB, $3.8\times$ larger than the model weights (Sheng et al. 2023).

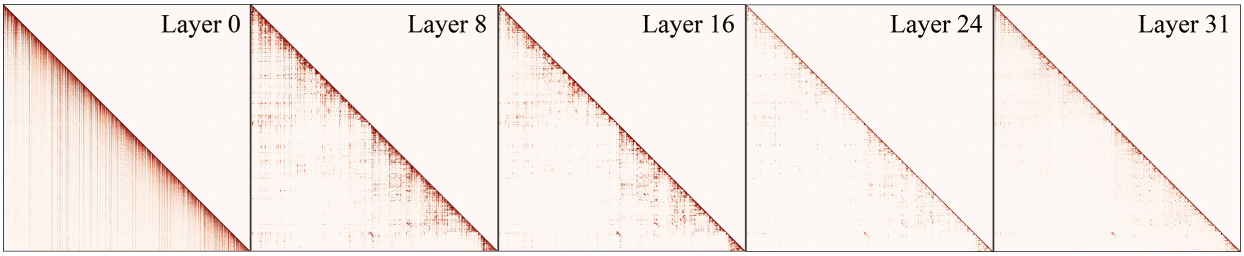
To address excessive KV cache memory consumption, recent studies have identified substantial sparsity in attention matrices during inference, where only a small subset of tokens achieve high activation values (attention scores) that critically influence generation quality (Liu et al. 2023; Zhang et al. 2023). This pattern suggests that selectively evicting unimportant tokens' KV pairs can reduce memory usage while preserving model accuracy. Researchers have proposed various metrics to identify tokens' importance, including calculating the cumulative attention scores under different windows, such as global tokens (Zhang et al. 2023), recent tokens(Li et al. 2024), and final-token (Oren et al. 2024), then retaining only the top-k scoring KV pairs per layer under a fixed memory budget. More sophisticated methods account for varying sparsity patterns across different layers as well as temporal and spatial attention variations, incorporating additional variables into their eviction criteria, such as integrating initial tokens into recent windows (Wan et al. 2024) or considering variability in attention distributions across tokens (Qin et al. 2025). These approaches typically combine important KV pair selection with layer-specific memory budget allocation strategies.

However, while existing methods have made notable progress in reducing KV cache size by evicting tokens based on cumulative attention scores, they overlook a crucial aspect: token activation frequency—that is, how often a token appears in the top-k attention ranks of subsequent tokens. Much like the Least Frequently Used (LFU) strategy in operating systems, tokens that are frequently activated are more likely to be relevant in future generation steps. Current approaches, however, assess token importance primarily by aggregating attention scores within a fixed token window, rather than considering how often individual tokens actually attract attention

*Corresponding authors: Yujuan Tan, Ao Ren



(a) long-prompt task



(b) long-context generation task

Figure 1: Attention score distribution of the intermediate transformer layers for different task types. (a) The long-context long-prompt task uses the LongBench dataset, and (b) the long-context generation task uses the PG19 dataset, based on LlaMa3.1-8B.

over time. As a result, tokens that achieve top-k attention rankings multiple times but with modest individual values may fail to reach high rankings in the final cumulative metrics. This is particularly problematic for tokens that consistently appear near the bottom of top-k selections—despite their frequent activation, their cumulative scores remain insufficient to survive eviction policies based solely on aggregate attention values. Our experiments demonstrate that removing these frequently-activated KV pairs degrades text generation quality, increasing model perplexity as shown in Figure 2(b), and weakens long-context understanding ability, as evidenced by lower key-information retrieval performance in Figure 2(c).

Furthermore, activation frequency displays distinct distribution patterns across both layers and tasks. As shown in Figure 1, which compares long-prompt tasks (with only long input) to long-context generation tasks (with both long input and output), the differences are notable. In long-prompt tasks, most tokens maintain consistently high activation frequencies across layers, with these frequencies often increasing in deeper layers—suggesting that critical information becomes more concentrated as the model processes deeper. In contrast, long-context generation tasks show a much broader dispersion of token activation frequencies: most tokens experience dynamic shifts in activation from layer to layer, though a small subset retains stable patterns similar to those found in long-prompt tasks. These significant differences in activation frequency—both between task types and across layers within each task—highlight the need for more nuanced, context-aware approaches. Such insights motivate further investigation and point to avenues for optimizing KV cache eviction strategies.

Based on these observations, we introduce HitKV, a KV cache compression method that adaptively evicts KV pairs based on token activation frequency to improve model adaptability across tasks. Instead of traditional cumulative attention scores, HitKV uses hit rate—the frequency with which a token appears in top-k attention—to assess token importance. Within a recent-token window and under a fixed memory budget, HitKV retains tokens with the highest hit rates, as they are most likely to benefit future generation. This novel eviction metric easily integrates with layer-specific memory allocation strategies, ensuring the most important tokens are preserved within given constraints. Evaluations on multiple LLMs and benchmarks show that HitKV consistently outperforms existing methods in LongBench accuracy, throughput, perplexity, and Needle-in-a-Haystack retrieval, offering a robust and adaptive solution for KV cache compression.

Our main contributions are:

- We critically analyze the limitations of traditional attention scoring mechanisms and conduct an in-depth investigation of layer-specific and task-specific variations in token activation frequencies.
- Leveraging our activation frequency analysis, we introduce a novel hit rate metric as an eviction indicator to effectively assess token importance with efficient scalability. To the best of our knowledge, this is the first eviction strategy fundamentally based on activation frequency dynamics.
- Experimental validation demonstrates that HitKV consistently outperforms existing approaches across long-prompt and long-context generation tasks. By strategically retaining merely 3% of the key-value (KV) cache, HitKV preserves model performance while simultaneously enhancing generation quality. On our targeted hardware infrastructure, HitKV achieves a remarkable 4× reduction in decoding latency.

Related Works on KV Cache Eviction

Previous studies have proposed various strategies to reduce KV cache memory usage by identifying important tokens and selectively evicting less important ones during generation. StreamingLLM (Xiao et al. 2024) and LM-Infinite (Han et al. 2024) prioritized retaining first and last tokens, though this risks losing critical intermediate information. Subsequent work introduced more refined indicators including cumulative attention scores (Zhang et al. 2023; Li et al. 2024), mean attention scores (Ren and Zhu 2024), last-token attention scores (Oren et al. 2024), but these methods overlook attention distribution variations across layers. To address this, newer approaches like PyramidInfer (Yang et al. 2024) and PyramidKV (Cai et al. 2025) allocate layer-specific memory with a pyramid structure, D2O (Wan et al. 2024) dynamically adjusts layer memory budgets using attention density, and CAKE (Qin et al. 2025) considers spatiotemporal attention variations through spatial entropy and temporal variance indicators. However, these methods still fail to account for tokens with high activation frequencies. In this work, we employ hit rate to quantify activation frequency to enhance critical token selection for more effective KV cache management.

Observations

In this section, we elaborate on the observed attention phenomena and analyze the layer- and task- specific characteristics of tokens' activation frequencies.

Recent research shows that not all tokens contribute equally to LLM outputs during inference. While some studies use cumulative or mean attention scores to identify important tokens, these methods often overlook tokens with high activation frequency, as such tokens may rank lower in top-k scores and receive lower cumulative values than occasional outliers. However, these frequently activated tokens are numerous and crucial for model performance. As shown in Figure 2(a), eviction methods based on cumulative, mean, and recent-token attention scores remove 38%, 32%, and 16% of high-frequency tokens' KV pairs, respectively—an average of 28.6%. Removing these KV pairs significantly degrades both long-text generation quality (Figure 2(b)) and long-text understanding ability (Figure 2(c)).

Furthermore, inspired by the distinct attention distribution patterns across layers and tasks, we observe that token activation frequencies also vary significantly by layer and task type (long-prompt vs. long-context generation). First, Tokens display irregular activation patterns across layers. As shown in Figure 3, mid-sequence tokens' activation frequencies and the number of high-frequency tokens both fluctuate sharply from layer to layer. Second, patterns differ by task type. In long-prompt tasks, certain tokens—especially initial and recent ones—consistently show high activation frequencies across most layers (Figure 3(b)), while the number of high-frequency tokens generally decreases in deeper layers (Figure 3(c)). In contrast, for generation tasks (Figure 3(a)), the positions of high-frequency tokens vary more across layers, but their overall count remains stable, and recent tokens always exhibit very high activation. These differences likely arise because long-prompt tasks emphasize local key information, while generation tasks require attention to the global context for quality outputs.

Based on these observations, we propose HitKV, a KV cache eviction strategy that employs hit rate to quantify activation frequency as the eviction indicator. Owing to the distinctive characteristics of activation frequency across layers and tasks, HitKV can adapt to layer- and task-specific characteristics. Furthermore, HitKV demonstrates highly scalability of which can be seamlessly integrated into layer-specific memory allocation strategy, as detailed in the next section.

HitKV

In this section, we describe HitKV in detail. The core idea is to quantify activation frequency as hit rate and leverage this indicator to identify critical KV pairs. We will also detail how HitKV adapts to layer- and task-specific characteristics and its scalability analysis.

Hit Rate

We first explain how we quantify the activation frequency as hit rate. Before detailing, we define a ***Hit***:

Definition 1*: Within the attention score distribution of a token in the recent window, if the score of the i-th token ranks within the Top-K tokens, it is recorded as one* ***Hit*** *for the i-th token. K is defined as the hit coefficient.*

Since preceding tokens cannot "see" later ones in attention mechanism, observing activation frequency over a global window inherently disadvantages later tokens. To ensure an equitable observation period, we instead measure the activation frequency over the recent window. As demonstrated by SnapKV, the recent window effectively identifies attention patterns.

To formally introduce our method, we define the following notations:

- Recent Window (L_w): The token set of recent window.
- Prompt Token (L_p): The token set of provided input. $|L_p| = |L_{pref}| + |L_w|$, where L_{pref} represents the token set before L_w
- Hit Token Set (Top-K): The set of tokens whose attention scores rank within Top-K tokens in the attention score distribution of the recent token (L_w). Top-K^j denotes the Top-k token set for the j-th token in L_w
- Hit-rate Threshold (θ): Threshold distinguishing between high and low hit-rates to evict KV pairs.

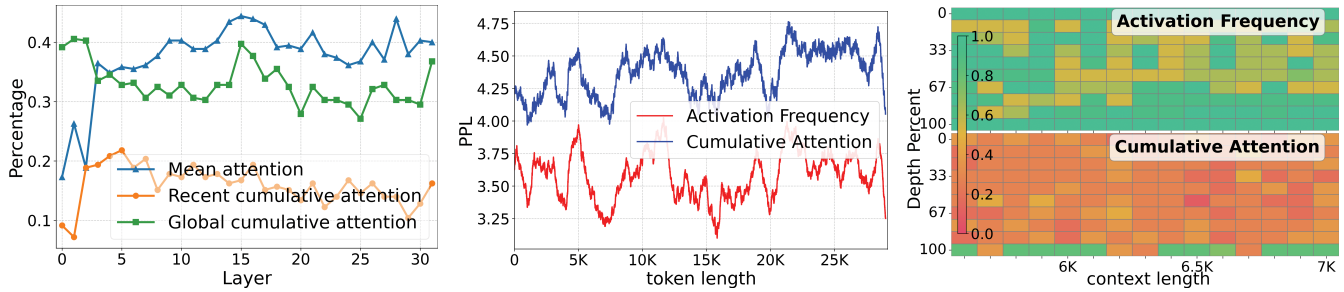
Based on these, the hit rate of the i-th token (t_i) in L_{pref} is calculated as follows:

$$T_{hr}(t_i) = \frac{\sum_{j=0}^{|L_w|} Hit_i^j}{|L_w|}, Hit_i^j = \begin{cases} 1 & \text{if } t_i \in \text{Top-}K^j \\ 0 & \text{else} \end{cases} \quad (1)$$

Hit-based KV Eviction Algorithm

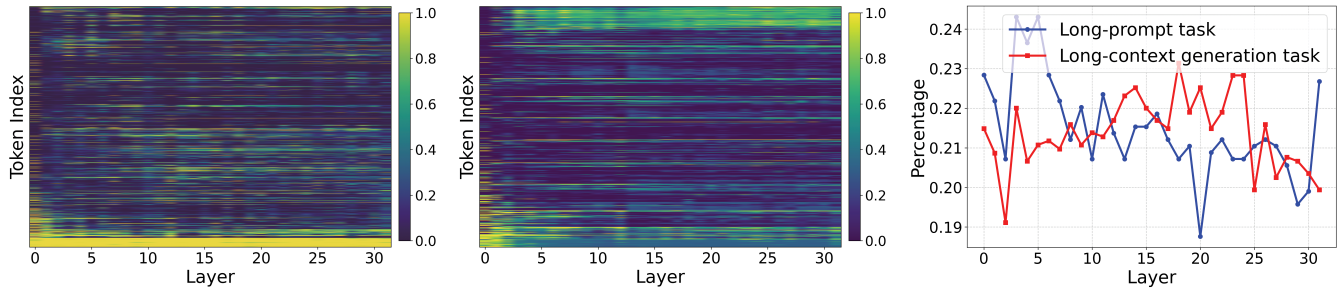
The core idea of HitKV is to identify critical tokens with high activation frequencies via hit rates. In general, HitKV comprises two key steps:

- **Compute hit rates.** Hit tokens over recent window are identified using ***Definition 1***, and their hit rates are calculated via Equation 1.



(a) Percentage of missed high-frequency tokens (b) PPL results (c) Needle in a Haystack

Figure 2: Proportion of missed high-frequency tokens and performance impact under variant attention score eviction indicators. These results are obtained by retaining KV cache of the same size (top 20%) based on activation frequency and attention score eviction indicators.



(a) Long-text generation tasks (b) Long-prompt tasks (c) Proportion of high-frequency tokens

Figure 3: Characteristic analysis of long-context generation tasks and long-prompt tasks. (a)&(b) Heatmap of token activation frequencies across layers; (c) Number variation of high-frequency tokens. Top 20% of the attention distribution of following tokens is considered an activation.

- **Update KV cache.** After computing hit rates, HitKV identifies the critical KV pairs. Since KV pairs with high hit rates exhibit higher activation frequencies, they are more likely to carry critical information. In contrast, KV pairs with low hit rates are deemed to encode secondary or redundant information. Therefore, HitKV evicts low hit-rate KV pairs with $T_{hr}(t_i) < \theta$ while retaining high hit-rate ones with $T_{hr}(t_i) \geq \theta$.

Notably, the number of retained KV pairs varies considerably across attention heads, as each attention head attends to different feature dimensions. To avoid missing outliers that may carry critical information, HitKV pads each head to a uniform length using additional KV pairs with the highest recent-cumulative attention scores. This can both minimize performance degradation under the budget constraints and maximize computational resource utilization.

Parameter Analysis

HitKV's KV retention/eviction via Top-K and hit rates inherently raises two questions:

1. How to configure K and θ for optimal model performance?
2. How to reconcile the non-deterministic KV retention count with predefined $budget$?

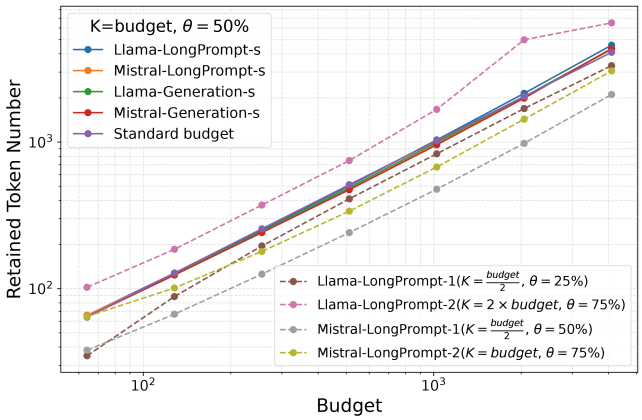


Figure 4: The average number of KV pairs retained under various parameter settings on LongBench and PG19. "Standard budget" is the standard number of KV pairs retained. It can be found that under the conditions of $\theta = 50\%$ and $K = budget$, the lines are almost exactly the same as the "Standard budget".

In fact, these two concerns converge into a single research problem: *Can coordinated configuration of K and θ simultaneously achieve budget-aligned retention and performance maximization?* Through systematic experimentation, we identify actionable empirical regularities addressing this dual objective.

As shown in Figure 4, when selecting tokens under $\theta == 50\%$ and $K == budget$, the retained counts aligns closely with budget. While it remains unclear whether this observation implies deeper theoretical significance, we empirically note that retaining tokens whose T_{hr} exceeds 50% often corresponds with practical half-and-half allocation patterns. Setting the budget as the hit coefficient K is also intuitively grounded in our design rationale. Notably, increasing both the θ and K proportionally could yield similar retention rates, but would introduce unnecessary implementation overhead. Our experiments further confirm that such complexification fails to deliver superior performance.

Scalability Analysis

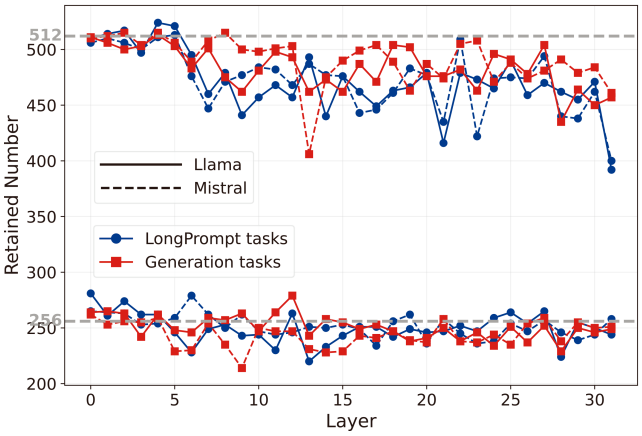


Figure 5: The number of KV pairs retained at each layer of the long-context generation tasks (PG19) and long-prompt tasks (LongBench) using the same K. The budgets are 256 and 512 KV pairs respectively.

Based on the distinct characteristics of activation frequencies across layers and tasks, HitKV can adaptively selects tokens with high activation frequencies under specified memory budgets (the hit coefficient K). Consequently, HitKV demonstrates strong scalability that it can be easily integrated into to other layer-specific memory budget allocation strategies.

First, under a uniform memory budget (same K for all layers), HitKV can effectively adapts to layer- and task-specific characteristics without complex tuning mechanisms. Figure 5 illustrates the distribution of high-hit-rate token numbers across layers and tasks under the same K.

Second, HitKV's eviction indicator can be easily integrated to layer-specific memory budget allocation strategies (different K per layer) with minimal code modifications, such as CAKE(Qin et al. 2025) and D2O(Wan et al. 2024). Each layer selects tokens with the highest activation frequencies exclusively within its allocated memory budget. We introduce P-HitKV in the *Experiments* section, which integrates HitKV's eviction indicator into PyramidKV.

Experiments

In this section, we comprehensively evaluate HitKV's performance on long-prompt tasks and long-context generation tasks across long-context processing capability, throughput, and perplexity.

Experimental Settings

Benchmark LLMs. Our experiments employ the Llama3-8B-Instruct and Mistral-v0.2-Instruct models supporting 4K-128K token contexts. Due to hardware constraints, current evaluations are limited to models with 7B-16B parameters. Performance tests on larger models will be conducted in future work.

Baseline methods. We evaluate HitKV against four methods: StreamingLLM (Xiao et al. 2024), H2O (Zhang et al. 2023), SnapKV (Li et al. 2024), and PyramidKV (Cai et al. 2025). These four methods respectively cover four types, including balancing initial and recent tokens, cumulative attention scores under global window and recent window, and layer-specific memory budgets, enabling comprehensive performance comparisons. Furthermore, we integrate HitKV's eviction indicator into PyramidKV's layer-specific memory allocation for performance comparison, denoted as P-HitKV.

Evaluation datasets. Performance is assessed using three established benchmarks: (1) LongBench (Bai et al. 2023): dedicated to long-context understanding across 16 datasets spanning 6 categories such as single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion; (2) Needle-in-a-Haystack (Kamradt. 2023): evaluating the key information extraction capability, which can reflect the LLM's basic ability to understand long contexts. (3) PG19 (Rae et al. 2019): comprising 100 book as its test set, with an average length of 70K tokens, widely used to evaluate long-range dependency modeling and long-text generation capabilities.

Parameter settings. The hyperparameter K (memory budget) ranges from 64 to 2048 tokens, and the hit frequency threshold (θ) is set to 50%. All experiments are run on NVIDIA A6000 48GB GPUs.

Evaluation on LongBench

We evaluate HitKV's long-context processing ability on the LongBench (Bai et al. 2023) dataset. LongBench is specifically designed to assess models' long-context understand ability. we integrate the hit-rate eviction indicator into PyramidKV's layer-specific memory allocation strategy to assess HitKV's scalability, denoted as P-HitKV.

Figure 6 presents the average accuracy of various methods across different memory budgets. While both PyramidKV and SnapKV employ the same eviction indicator, PyramidKV utilizes a invert-pyramid-shaped memory budget allocation strategy, leading to stronger performance in certain scenarios. However, its static memory allocation strategy reduces its

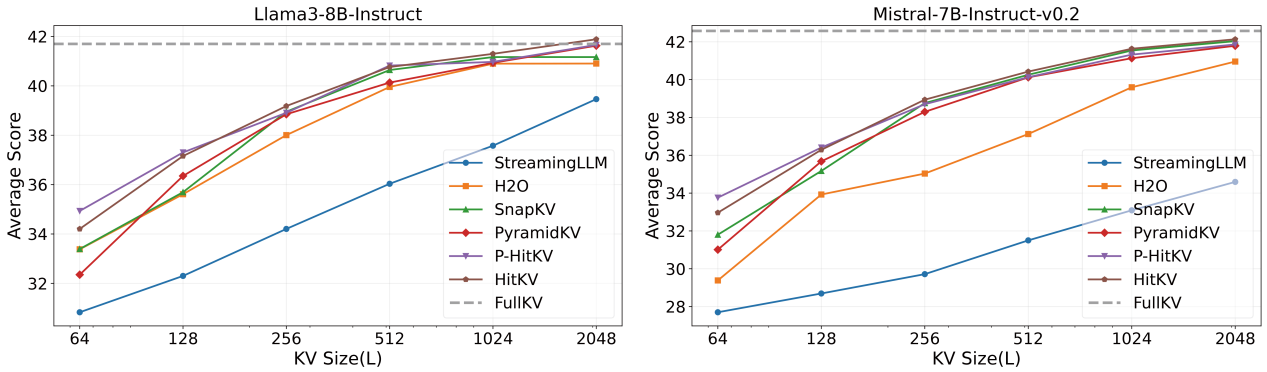


Figure 6: Average score among 16 datasets of LongBench under different memory budgets.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpQA	2wikMQA	Musique	GovRpt	QMSum	MltNews	TREC	TriQA	SAMSum	PCount	PR-en	Lcc	RB-P	
Llama3-8B-Instruct, $B_{\text{total}} = 128L$																	
FullKV	25.56	31.89	40.35	43.57	35.64	21.18	28.66	23.29	26.67	73.5	90.48	42.37	4.8	69.25	57.57	52.42	41.7
Streaming	18.13	8.51	21.31	32.86	25.85	15.51	16.76	20.43	18.16	45.0	74.59	36.35	5.75	68.5	56.94	52.17	32.3
H2O	**23.42**	14.88	24.85	35.66	28.49	18.26	**21.79**	21.86	**23.78**	46.5	88.29	38.65	**6.0**	68.92	57.88	50.56	35.61
SnapKV	21.08	13.5	31.34	36.9	29.35	19.13	19.14	21.64	20.36	44.5	87.8	37.84	5.13	68.85	58.63	**55.91**	35.69
PyramidKV	23.1	14.12	31.72	**39.36**	28.91	19.26	19.52	21.8	20.43	50.0	87.26	37.85	5.55	**69.25**	58.38	55.11	36.35
P-HitKV	22.69	**16.61**	32.39	38.29	27.97	**19.83**	19.73	**22.34**	21.08	**60.5**	88.99	38.78	5.93	69.0	59.04	53.69	**37.30**
HitKV	22.08	14.32	**33.66**	38.44	**31.68**	18.75	19.38	22.17	20.86	55.5	**89.02**	**38.84**	5.3	68.85	**61.52**	54.17	37.16
Llama3-8B-Instruct, $B_{\text{total}} = 1024L$																	
Streaming	20.51	14.91	26.73	37.3	26.7	17.1	23.1	21.04	25.54	68.67	85.99	40.25	3.33	**70.0**	**63.16**	**56.89**	37.58
H2O	25.0	**27.73**	35.31	43.29	31.54	19.73	**25.84**	22.48	**26.29**	**74.0**	**90.96**	41.04	4.43	69.67	61.93	55.24	40.9
SnapKV	**25.54**	27.3	37.81	**44.39**	**35.97**	21.35	23.88	22.82	25.37	73.33	89.97	40.4	4.24	69.67	61.51	55.13	41.16
PyramidKV	25.09	26.41	38.48	43.46	35.23	22.24	23.61	**23.46**	25.34	73.33	89.8	40.68	3.93	69.67	60.65	53.57	40.93
P-HitKV	24.92	26.49	36.23	43.75	33.47	21.43	24.38	22.84	25.87	72.5	90.34	**41.17**	**5.61**	69.5	61.0	56.08	40.97
HitKV	25.28	27.66	**38.52**	43.57	35.6	**23.18**	23.89	22.44	25.51	**74.0**	89.97	40.16	4.33	69.67	62.37	54.62	**41.3**
Mistral-7B-Instruct-v0.2, $B_{\text{total}} = 128L$																	
FullKV	26.77	33.0	49.46	43.02	27.35	18.77	32.96	24.25	27.11	71.0	86.23	42.75	2.75	86.98	54.72	54.1	42.58
Streaming	17.12	13.43	27.42	29.54	21.91	11.94	15.5	19.26	17.78	44.0	79.92	37.35	3.75	24.27	51.2	44.69	28.69
H2O	**22.72**	20.26	35.71	34.65	22.71	14.3	**25.42**	21.9	**24.54**	**63.5**	84.72	39.37	**4.92**	37.49	48.05	42.55	33.93
SnapKV	19.69	21.22	43.16	36.88	22.38	**16.0**	19.16	21.84	21.36	48.0	84.19	40.37	2.31	67.76	51.74	**46.69**	35.17
PyramidKV	19.67	21.83	44.26	**37.63**	23.38	14.08	19.41	21.96	21.19	51.0	84.62	40.06	2.92	**72.1**	50.23	46.52	35.67
P-HitKV	20.3	**22.37**	**45.06**	35.81	24.03	14.94	20.82	22.22	21.87	58.0	84.01	**40.55**	2.84	71.65	**52.18**	45.92	**36.41**
HitKV	21.65	20.68	44.35	36.33	**24.13**	15.65	20.84	**22.54**	21.64	56.0	**85.36**	39.53	2.66	70.82	51.9	46.6	36.29
Mistral-7B-Instruct-v0.2, $B_{\text{total}} = 1024L$																	
Streaming	22.35	18.07	30.59	32.08	22.31	11.82	23.94	20.48	25.62	64.0	84.71	41.35	3.27	22.9	54.71	51.25	33.09
H2O	**26.51**	29.57	45.99	37.87	25.7	18.32	**29.62**	23.57	**26.71**	70.0	**87.57**	42.39	**3.72**	60.26	54.59	51.14	39.60
SnapKV	25.38	29.64	**49.72**	40.94	25.73	**19.0**	26.08	23.59	26.07	69.5	86.48	42.14	3.19	**88.56**	**55.39**	**53.26**	41.54
PyramidKV	24.32	29.80	48.77	40.75	25.46	18.77	25.59	**24.05**	25.84	68.5	86.31	42.02	2.89	87.17	54.7	53.18	41.13
P-HitKV	24.33	29.81	49.14	**42.04**	26.52	18.04	26.78	23.51	26.19	69.0	85.98	42.6	2.89	86.62	54.61	53.10	41.32
HitKV	24.74	**30.9**	49.03	41.59	**26.96**	17.24	27.31	23.52	26.19	**70.5**	86.36	**42.41**	3.2	88.23	55.26	52.71	**41.63**

Table 1: Performance comparison on the LongBench dataset, where the best results are highlighted in **bold** and the second-best are marked with an underline.

adaptability to varying tasks. In contrast, HitKV selects high-frequency tokens at specific hit coefficient ($K = budget$), effectively adapting to layer-specific and task-specific characteristics, and demonstrates superior performance. By integrating HitKV's eviction indicator into PyramidKV (P-HitKV), we enhance PyramidKV's task adaptability, yielding significant performance gains. Table 1 presents the accuracy on 16 datasets of LongBench under memory budgets of 128 and 1,024 tokens. Both HitKV and P-HitKV achieve superior performance across most datasets, further demonstrating

HitKV's adaptivity and scalability.

Evaluation on Needle-in-a-Haystack

Method	B=64L	B = 128L	B=256L	B=512L	B=1024L
Llama3-8B-Instruct, L=8K					
Streaming	27.2	31.3	33.8	38.0	50.4
H2O	29.1	29.6	32.0	43.0	62.0
SnapKV	69.3	86.6	95.1	97.0	98.2
PyramidKV	65.8	87.4	95.7	97.2	98.9
HitKV	70.6	87.6	95.9	97.4	98.9
P-HitKV	73.2	88.0	96.2	98.6	99.0
Mistral2-7B-Instruct-v0.2, L=32K					
Streaming	20.2	27.2	30.3	33.0	35.7
H2O	28.1	33.5	40.9	49.6	63.9
SnapKV	59.5	76.5	89.6	93.9	96.2
PyramidKV	60.9	76.9	90.1	93.9	96.4
HitKV	77.0	88.8	94.1	95.8	97.5
P-HitKV	80.6	89.2	94.5	96.2	97.3

Table 2: Needle in a Haystack results.

To further evaluate HitKV's long-context processing capabilities, we conducte Needle-in-a-Haystack experiments to assess the key information extraction performance. We compare the accuracy of all methods under memory budgets of 64, 128, 256, 512, and 1024 tokens on Llama3-8B-Instruct (8K) and Mistral-7B-Instruct-v0.2 (32K). As shown in Table 2, HitKV and P-HitKV significantly outperform other methods across all memory budgets. Notably, HitKV and P-HitKV deliver commendable performance even under minimal memory budget ($B > 128L$), demonstrating HitKV's robust long-context information retrieval capability.

Evaluation on PG19 Datasets

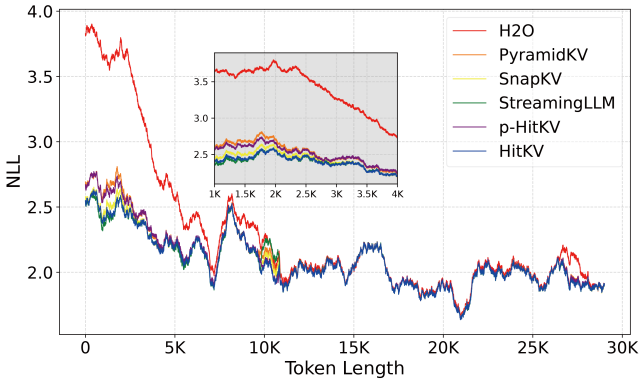


Figure 7: Long sequence modeling PPL.

To evaluate HitKV's performance on long-context generation tasks, we sampled data from PG19 (Rae et al. 2019) to model perplexity. To ensure fairness and focus on generation capacity, all methods compress only during the prefill stage. All methods compress an 11K-token prompt to 128 tokens and then evaluate the perplexity on Mistral-7B. Figure 7 shows the cumulative average negative log-likelihood (NLL) as a function of sequence length. HitKV achieves higher-quality outputs (lower perplexity). PyramidKV, due to the static layer-specific memory allocation, loses task adaptability and thus underperforms on long-context generation tasks. Although P-HitKV offers some adjustability, its generation quality remains lower than other methods. While StreamingLLM demonstrate competitive perplexity, its long-context understand ability significantly underperforms. StreamingLLM's stronger perplexity may be attributed to the high activation frequency of recent tokens in long-context generation tasks as we described in our observations. As the sequence length increase, all methods' perplexities converge, since they then operate on nearly identical context.

Evaluation on Throughput

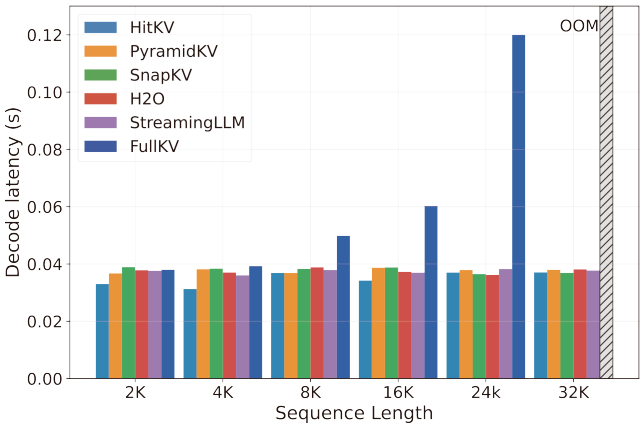


Figure 8: Decoding latency on A6000 48GB GPU

We evaluate time overhead of HitKV's hit-rate eviction indicator by analyzing decoding latency on the Mistral-7B-Instruct-v0.2. All methods maintain a fixed KV cache memory budget of $B_{total} = 128L$. As shown in Figure 8, HitKV achieves comparable inference performance to the baselines, indicating no additional overhead from the eviction indicator. Furthermore, the latency of the full-cache method increases significantly with longer input sequences, due to the additional computational and I/O overhead. In contrast, HitKV maintains relatively stable decoding latency owing to its fixed memory budget. HitKV achieves $4\times$ inference acceleration at 24k input length. For longer input sequences, HitKV is expected to deliver even greater speedups, though hardware memory limitations prevented us from validating this experimentally.

Conclusion

This paper proposes HitKV, a KV cache compression method that quantifies activation frequency as the hit-rate eviction indicator. HitKV can adapt to various tasks and be readily integrated with layer-specific memory allocation strategies. Evaluated on the LongBench, Needle-in-a-Haystack, and PG19 datasets, HitKV demonstrate the superior performance on long-prompt tasks and long-context generation tasks, and significantly reduces memory usage while maintaining high accuracy and output quality.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant No.62472058 and No.62572085, the Natural Science Foundation of Shandong Province under Grant No.ZR2024LZH013, and the State Key Lab of Processors in Institute of Computing Technology under CAS Grant No.CLQ202510.

References

AI, .; :; Young, A.; Chen, B.; Li, C.; Huang, C.; Zhang, G.; Zhang, G.; Li, H.; Zhu, J.; Chen, J.; Chang, J.; Yu, K.; Liu, P.; Liu, Q.; Yue, S.; Yang, S.; Yang, S.; Yu, T.; Xie, W.; Huang, W.; Hu, X.; Ren, X.; Niu, X.; Nie, P.; Xu, Y.; Liu, Y.; Wang, Y.; Cai, Y.; Gu, Z.; Liu, Z.; and Dai, Z. 2024. Yi: Open Foundation Models by 01.AI. arXiv:2403.04652.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

Bai, Y.; Lv, X.; Zhang, J.; Lyu, H.; Tang, J.; Huang, Z.; Du, Z.; Liu, X.; Zeng, A.; Hou, L.; Dong, Y.; Tang, J.; and Li, J. 2023. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. arXiv:2308.14508.

Cai, Z.; Zhang, Y.; Gao, B.; Liu, Y.; Li, Y.; Liu, T.; Lu, K.; Xiong, W.; Dong, Y.; Hu, J.; and Xiao, W. 2025. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. arXiv:2406.02069.

Chang, Y.; Wang, X.; Wang, J.; Wu, Y.; Yang, L.; Zhu, K.; Chen, H.; Yi, X.; Wang, C.; Wang, Y.; Ye, W.; Zhang, Y.; Chang, Y.; Yu, P. S.; Yang, Q.; and Xie, X. 2024. A Survey on Evaluation of Large Language Models. *ACM Trans. Intell. Syst. Technol.*, 15(3).

Dai, S.; Xu, C.; Xu, S.; Pang, L.; Dong, Z.; and Xu, J. 2024. Bias and Unfairness in Information Retrieval Systems: New Challenges in the LLM Era. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, 6437–6447. New York, NY, USA: Association for Computing Machinery. ISBN 9798400704901.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805.

GLM, T.; :; Zeng, A.; Xu, B.; Wang, B.; Zhang, C.; Yin, D.; Zhang, D.; Rojas, D.; Feng, G.; Zhao, H.; Lai, H.; Yu, H.; Wang, H.; and Sun, J. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. arXiv:2406.12793.

Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; Yang, A.; Fan, A.; Goyal, A.; and Hartshorn, A. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.

Han, C.; Wang, Q.; Xiong, W.; Chen, Y.; Ji, H.; and Wang, S. 2024. LM-Infinite: Simple On-the-Fly Length Generalization for Large Language Models.

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

Li, H.; Li, Y.; Tian, A.; Tang, T.; Xu, Z.; Chen, X.; Hu, N.; Dong, W.; Li, Q.; and Chen, L. 2025. A Survey on Large Language Model Acceleration based on KV Cache Management. arXiv:2412.19442.

Li, Y.; Huang, Y.; Yang, B.; Venkitesh, B.; Locatelli, A.; Ye, H.; Cai, T.; Lewis, P.; and Chen, D. 2024. SnapKV: LLM Knows What You are Looking for Before Generation. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *Advances in Neural Information Processing Systems*, volume 37, 22947–22970. Curran Associates, Inc.

Liu, Z.; Wang, J.; Dao, T.; Zhou, T.; Yuan, B.; Song, Z.; Shrivastava, A.; Zhang, C.; Tian, Y.; Re, C.; and Chen, B. 2023. Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time. In Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; and Scarlett, J., eds., *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, 22137–22176. PMLR.

Munkhdalai, T.; Faruqui, M.; and Gopal, S. 2024. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 101.

OpenAI; Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; Avila, R.; Babuschkin, I.; Balaji, S.; Balcom, V.; Baltescu, P.; Bao, H.; Bavarian, M.; and Belgum, J. 2024. GPT-4 Technical Report. arXiv:2303.08774.

Oren, M.; Hassid, M.; Yarden, N.; Adi, Y.; and Schwartz, R. 2024. Transformers are Multi-State RNNs. arXiv:2401.06104.

Qin, Z.; Cao, Y.; Lin, M.; Hu, W.; Fan, S.; Cheng, K.; Lin, W.; and Li, J. 2025. CAKE: Cascading and Adaptive KV Cache Eviction with Layer Preferences. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

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

Rae, J. W.; Potapenko, A.; Jayakumar, S. M.; Hillier, C.; and Lillicrap, T. P. 2019. Compressive Transformers for Long-Range Sequence Modelling. *arXiv preprint*.

Ren, S.; and Zhu, K. Q. 2024. On the Efficacy of Eviction Policy for Key-Value Constrained Generative Language Model Inference. arXiv:2402.06262.

Sheng, Y.; Zheng, L.; Yuan, B.; Li, Z.; Ryabinin, M.; Chen, B.; Liang, P.; Ré, C.; Stoica, I.; and Zhang, C. 2023. FlexGen: high-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Shi, L.; Zhang, H.; Yao, Y.; Li, Z.; and Zhao, H. 2024. Keep the Cost Down: A Review on Methods to Optimize LLM' s KV-Cache Consumption. arXiv:2407.18003.

Wan, Z.; Wu, X.; Zhang, Y.; Xin, Y.; Tao, C.; Zhu, Z.; Wang, X.; Luo, S.; Xiong, J.; and Zhang, M. 2024. D2O: Dynamic Discriminative Operations for Efficient Generative Inference of Large Language Models. *CoRR*, abs/2406.13035.

Wu, J.; Yang, S.; Zhan, R.; Yuan, Y.; Chao, L. S.; and Wong, D. F. 2025. A Survey on LLM-Generated Text Detection: Necessity, Methods, and Future Directions. *Computational Linguistics*, 51(1): 275–338.

Xiao, G.; Tian, Y.; Chen, B.; Han, S.; and Lewis, M. 2024. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453.

Yang, D.; Han, X.; Gao, Y.; Hu, Y.; Zhang, S.; and Zhao, H. 2024. PyramidInfer: Pyramid KV Cache Compression for High-throughput LLM Inference. In Ku, L.; Martins, A.; and Srikumar, V., eds., *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, 3258–3270. Association for Computational Linguistics.

Yi, Z.; Ouyang, J.; Liu, Y.; Liao, T.; Xu, Z.; and Shen, Y. 2024. A Survey on Recent Advances in LLM-Based Multi-turn Dialogue Systems. arXiv:2402.18013.

Zhang, Y.; Jin, H.; Meng, D.; Wang, J.; and Tan, J. 2025. A Comprehensive Survey on Process-Oriented Automatic Text Summarization with Exploration of LLM-Based Methods. arXiv:2403.02901.

Zhang, Z.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C.; Wang, Z. A.; and Chen, B. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 34661–34710. Curran Associates, Inc.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).