ZigZagKV: Dynamic KV Cache Compression for Long-context Modeling based on Layer Uncertainty

Meizhi Zhong[1*], **Xikai Liu**[2], **Chen Zhang**[2], **Yikun Lei**[2], **Yan Gao**[2], **Yao Hu**[2], **Kehai Chen**[1 †], **Min Zhang**[1]

[1]Institute of Computing and Intelligence, Harbin Institute of Technology, Shenzhen, China

[2] Xiaohongshu Inc.

meizhi.zhong.1999@gmail.com, chenzhang9702@outlook.com,

{chenkehai,zhangmin2021}@hit.edu.cn,

{xikai,zhizhu,yadun,xiahou}@xiaohongshu.com

Abstract

Large Language models (LLMs) have become a research hotspot. To accelerate the inference of LLMs, storing computed caches in memory has become the standard technique. However, as the inference length increases, growing KV caches might lead to out-of-memory issues. Many existing methods address this issue through KV cache compression, primarily by preserving key tokens throughout all layers to reduce information loss. Most of them allocate a uniform budget size for each layer to retain. However, we observe that the minimum budget sizes needed to retain essential information vary across layers and models based on the perspectives of attention and hidden state output. Building on this observation, this paper proposes a simple yet effective KV cache compression method that leverages layer uncertainty to allocate budget size for each layer. Experimental results show that the proposed method can reduce memory usage of the KV caches to only ∼20% when compared to Full KV inference while achieving nearly lossless performance.

1 Introduction

Large language models (LLMs) (Radford et al., 2018; Touvron et al., 2023; Zhang et al., 2023a; Brown et al., 2020; Huang et al., 2024) have been employed across a wide range of natural language processing tasks, including code completion (Rozière et al., 2023) and question answering (Kamalloo et al., 2023; Jiang et al., 2021; Su et al., 2019). To accelerate inference, it is common practice to store precomputed key and value hidden states in memory as a KV cache. However, as input lengths increase during long-context modeling (Peng et al., 2023; Peng and Quesnelle, 2023; Fu et al., 2024; Zhong et al., 2024; Zhang et al., 2024a), the size of the KV caches grows proportionally, leading to memory consumption and out-of-memory issues. For instance, maintaining a KV caches for 100K tokens in GPU memory for the LLaMA-2 7B model requires over 50GB of memory, compared to less than 1GB for a 2K context.

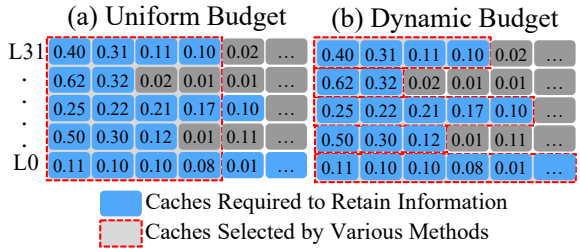


Figure 1: Comparison of the proposed method (ZigZagKV, Right) with previous PartialKV inference methods (Left). Most existing PartialKV methods allocate a uniform cache size per layer, whereas ZigZagKV dynamically adjusts the cache size based on layer uncertainty. "L0" and "L31" refer to Layer 0 and Layer 31, respectively. The numbers in brackets represent importance scores from the current token to prefix tokens, sorted from high to low.

A straightforward solution to mitigate these memory issues is to reduce the size of the KV caches, thereby decreasing memory usage (Xiao et al., 2023; Liu et al., 2024; Li et al., 2024; Chen et al., 2024; Ren and Zhu, 2024; Zhang et al., 2024b; Yang et al., 2024). The key to these methods lies in evicting nonessential KV caches while minimizing information loss caused by the eviction process. As depicted in Figure 1(a), the majority of these approaches uniformly treat each layer and preserve the top-B most important tokens at each respective layer. However, it remains unclear whether the strategy of equally preserving the top-B important tokens across all layers is an effective way to optimize information retention during the compression process.

To answer this question, we investigate the

[*]Work during Xiaohongshu internship.

[†]Corresponding authors

Proceedings of the 31st International Conference on Computational Linguistics, pages 8897–8907
January 19–24, 2025.

impact of token removal on information loss from the perspectives of attention mechanisms and hidden state outputs across different layers. Our empirical results reveal that the minimum required budget size varies across layers and models to maintain the same level of attention or hidden state output loss as in fullKV inference.

Building on these findings, we propose a novel KV caches compression method, called ZigZagKV, which minimizes information loss by dynamically allocating budget size for each layer. As illustrated in Figure 1(b), the key idea of ZigZagKV is to adjust the budget size based on the uncertainty of each layer. For instance, Layer 0, which exhibits more diffuse attention (i.e., higher uncertainty), is allocated a larger budget to retain its KV caches and reduce information loss. In contrast, layers with more concentrated attention (i.e., lower uncertainty) receive smaller budgets. In practice, the proposed method first assigns an initial budget to each layer and then dynamically adjusts the remaining cache based on layer uncertainty. Experiments across various benchmarks demonstrate that ZigZagKV outperforms existing partialKV inference methods.

Our key contributions can be summarized as follows:

- We analyze the differences in minimum budget sizes required to maintain information across layers and models, considering both attention mechanisms and hidden state outputs.
- Based on these observations, we propose ZigZagKV, a simple yet effective KV caches compression method that dynamically allocates budget size to each layer based on its uncertainty.
- Experimental results show that ZigZagKV outperforms existing KV cache compression methods on two widely-used benchmarks: Needle-in-a-Haystack and LongBench.

2 Problem Formulation

2.1 FullKV Inference

Large Language Model (LLM) inference operates in an autoregressive manner. During training, the upper triangular part of the attention matrix is masked to ensure that each token only attends to itself and the preceding tokens. At inference time, a common approach is to cache the key-value vectors computed up to the current step and append the newly computed vectors to this cache. Specifically, at each time step, the computed key states and value states are stored as a Key-Value (KV) Cache, which can be formalized as follows, where h denotes the number of attention heads and $i \in [1, h]$ indexes these heads, X represents the input embeddings, W_i^K is the key projection matrix for head i, and W_i^V is the value projection matrix for head i:

$$K_i = XW_i^K, V_i = XW_i^V.$$

Then, for the computation of the next token, x is mapped through the query projection W_i^Q, key projection W_i^K, and value projection W_i^V as follows:

$$q_i = xW_i^Q, k_i = xW_i^K, v_i = xW_i^V.$$

The key and value states are then updated based on the previous key-value (KV) Cache:

$$K_i = Cat[K_i : k_i], V_i = Cat[V_i : v_i].$$

Finally, the updated query, key, and value states are used to compute the attention weights.

$$A_i = \text{Softmax}(\frac{q_i K_i^T}{\sqrt{d_h}}).$$

$$y = \sum^{i\in[1,h]} A_i V_i W_i^O. \quad (1)$$

2.2 PartialKV Inference

In fullKV inference, the size of the key-value cache grows linearly with the total sequence length, which can lead to out-of-memory issues. Recent studies have shifted towards partialKV inference to address this. Given a budget size of B for each attention head, partialKV inference maintains the key-value cache by applying a cache eviction policy, as defined below:

$$\hat{K}_i, \hat{V}_i = \text{Eviction}(K_i, V_i).$$

Using the evicted key-value cache, attention weights are then calculated as follows:

$$\hat{y} = \sum^{i\in[1,h]} \hat{A}_i \hat{V}_i W_i^O, \quad (2)$$

where $\hat{A}_i = \text{Softmax}(\frac{q_i \hat{K}_i^T}{\sqrt{d_h}})$.

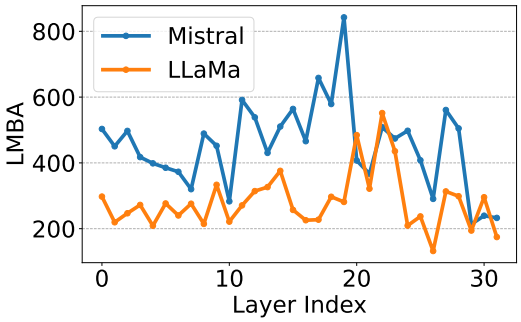


Figure 2: LMBA across various Layers of Mistral and LLaMa on 2WikiMQA dataset.

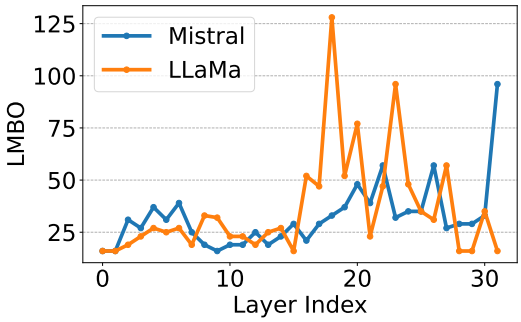


Figure 3: LMBO across various Layers of Mistral and LLaMa on 2WikiMQA dataset.

3 Rethinking PartialKV Inference

Compared to fullKV inference, partialKV inference inevitably incurs some degree of information loss due to the reduction of the key-value (KV) cache. To mitigate this information loss, many partialKV inference strategies seek to minimize the discrepancy from fullKV inference with a fixed memory budget. For example, some approaches (Zhang et al., 2023b; Liu et al., 2024; Li et al., 2024) focus on retaining tokens with the highest attention scores, aiming to preserve the most crucial information from fullKV and thus reduce attention loss, which in turn helps minimize overall information loss. Typically, these methods heuristically assign the same budget size B to each attention head across different layers, retaining the Top-B most important tokens.

However, it is uncertain whether preserving the top-B tokens with the highest scores equally across all heads in each layer effectively optimizes information retention. In the following section, we will examine the impact of token removal on information loss, considering the attention mechanisms and hidden state outputs across different layers.

Layer-Specific Budget Setting for Attention Retention. Firstly, we investigate the relationship between the budget size to retain KV caches and information loss across different layers from the perspective of attention mechanisms. Specifically, we calculate the **M**inimum **B**udget size required to maintain 90% of the total **A**ttention score (MBA) for each head, which corresponds to the attention loss as 0.1. This is formally defined as:

$$\text{MBA} = \arg\min_{I \subseteq [n]} \left\{ |I| \,\middle|\, 1.0 - \sum_{i \in I} a_i < 0.1 \right\}.$$

Next, we compute the average MBA across all heads within a layer to determine the required budget size for that layer, termed **L**ayer **M**inimum **B**udget size to maintain **A**ttention score (LMBA). The LMBA is formally defined as:

$$\text{LMBA} = \frac{1}{h} \sum_{i=1}^{h} \text{MBA}_i. \tag{3}$$

A higher LMBA indicates that more tokens are required to maintain an attention loss of 0.1 in that layer, suggesting a larger budget allocation. Empirically, we analyze the LMBA on two widely-used large language models, Mistral-7B-Instruct-v0.3 (Jiang et al., 2023)(Mistral) and LLaMA-3.1-8B-Instruct (Dubey et al., 2024)(LLaMA), using 200 samples from the 2WikiQA dataset (Ho et al., 2020). As illustrated in Figure 2, the LMBA varies across different layers: it initially requires a relatively larger budget in the lower layers to maintain an attention loss of 0.1, then decreases in the middle layers, increases again in the higher layers, and decreases. This phenomenon indicates that the LMBA varies across different layers to maintain the same level of attention loss. ***This suggest that applying a uniform budget size B across all layers to retain the top-B tokens may not be optimal for preserving attention information.***

Layer-Specific Budget Setting for Hidden State Output Retention. Next, we investigate the impact of budget size on information loss across different layers, focusing on the output of hidden states. Similarly, we examine each layer to determine the minimal budget size required for achieving a similarity of at least 90% between partialKV and fullKV inference outputs. This threshold is denoted as the Layer-wise Minimum

Budget for Output (LMBO). The formal definition of LMBO is as follows:

$$\text{LMBO} = \arg\min_{B} \left\{ B \,\middle|\, 1.0 - \text{similarity}(y, \hat{y}) < 0.1 \right\}$$

Where y represents the hidden state output in fullKV inference, as shown in Equation 1, and $\hat{y}$ is the hidden state output inf partialKV inference, as illustrated in Equation 2. The experimental results, depicted in Figure 3, indicate that similar to LMBA, the LMBO varies across different layers. In the case of LLaMa, the cache size required to maintain information stability is minimal initially but increases with layer depth. Conversely, for Mistral, the trend of required cache size to preserve stability is consistently upward as layer depth increases. This phenomenon suggests that LMBO varies across different layers and model types to maintain a consistent level of hidden state output information loss. ***Therefore, it may not be optimal to apply a uniform budget size to retain the top-B tokens across all layers for preserving hidden state output information.***

4 ZigZagKV

4.1 Dynamic Budget Allocation Based on Layer Uncertainty

The analysis in Section 3 demonstrates that using a uniform budget size B across all layers to select the top-B tokens is suboptimal for retaining information, both in terms of attention and hidden state outputs. Most current partialKV methods use this uniform approach, which may lead to unnecessary information loss. For instance, Figure 1(a) illustrates that certain layers, particularly the first one, may risk discarding important information. In contrast, layers where information is concentrated on specific tokens do not require the same budget size allocation.

To mitigate this issue, we introduce ZigZagKV, a dynamic method for allocating the budget size more effectively across layers to enhance information retention. Given an average budget size B, we determine the uncertainty in each layer l using the Layer Minimum Budget size to maintain Attention (LMBA) as defined in Equation 3. The uncertainty is then used to adjust the budget size for each specific layer as described below:

$$\text{uncertainty}_l = \frac{\text{LMBA}_l}{\sum i \in [1, L] \text{LMBA}_i}. \quad (4)$$

$$\hat{B}_l = B \cdot L \cdot \text{uncertainty}_l. \quad (5)$$

Where L represents the total number of layers. As illustrated in Figure 1(b), layers with higher uncertainty are allocated a larger portion of the budget, while those with less uncertainty receive a smaller share.

Allocating the budget solely based on layer uncertainty can result in shallow budget sizes for layers with lower uncertainty, potentially leading to inadequate information retention. For example, if the LMBA value of one layer is significantly higher compared to others, Equation 5 could allocate an excessively large budget to this layer, leaving the remaining layers with limited resources and potentially leading to information loss. To resolve this, we propose a mechanism where a fixed minimum budget, B_{bound}, is allotted to each layer to protect against information degradation. The leftover budget is then distributed dynamically, informed by the layer uncertainty. The allocation strategy is formulated as follows:

$$\hat{B}_l = B_{\text{bound}} + (B - B_{\text{bound}}) \cdot L \cdot \text{uncertainty}_l, \quad (6)$$

where uncertainty_l is calculated as shown in Equation 4. By incorporating B_{bound}, the method ensures that each layer receives a guaranteed minimum budget to preserve information while allowing dynamic adjustments to optimize information retention based on uncertainty.

4.2 KV Cache Selection

After determining the budget size for each layer, the next is to select the crucial tokens for each head of specific layers. The core concept of KV cache selection involves dynamically updating the KV cache by leveraging cumulative attention scores (Zhang et al., 2023b; Li et al., 2024; Zhang et al., 2024b; Wan et al., 2024). Following Li et al. (2024) using cumulative attention scores of instruction tokens as the importance scores, we adopt a similar approach by using the cumulative attention scores of the last w tokens to assign importance scores to the prefix tokens. Specifically, given the budget size calculated by Equation 6, for each attention head h, the importance score for retaining the i-th token in the KV cache, denoted as s_i^h, is computed as:

$$s_i^h = \sum_{j \in [n-w, n]} A_{ij}^h \quad (7)$$

Where n represents the sequence length of the prompt, and $[n - w, n]$ represents the range of the last segment (instruction tokens) in the prompt.

5 Experiments

5.1 Backbone LLMs

We compare the proposed method against several baselines using two open-source LLMs, specifically LLaMa-3.1-8B-Instruct (Dubey et al., 2024) and Mistral-7B-Instruct-v0.3 (Jiang et al., 2023).

5.2 Benchmarks

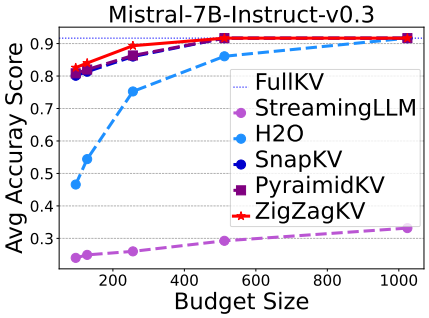
The proposed approach is evaluated on two widely used benchmarks: Needle-in-a-Haystack and LongBench.

Needle-in-a-Haystack The Needle-in-a-Haystack testing (Kamradt, 2023; Fu et al., 2024) challenges the model to accurately retrieve a specific sentence (the "needle") hidden within a lengthy document (the "haystack"), with the sentence placed at a random location. This test evaluates whether LLMs can extract key information from extensive texts and specifically examines the impact of the proposed adaptive allocation on the models' long-context retrieval abilities. We evaluate all partial KV inference methods for this test using mean cache budgets $B \in \{128, 256, 512, 1024\}$.

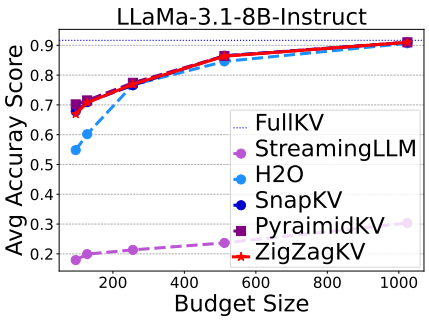
LongBench LongBench (Bai et al., 2023) is a multi-task benchmark designed to rigorously evaluate long-context understanding across various datasets, including single- and multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. For LongBench, we evaluate all partial KV inference methods using mean cache budgets $B \in \{128, 256, 512, 1024, 2048\}$.

5.3 Baselines

We conduct experiments comparing the following methods: **FullKV** (`FullKV`) caches all keys and values for every input token in each layer. **StreamingLLM** (`StreamLM`) (Xiao et al., 2023) retains the KV cache of the last α tokens and the first $k - \alpha$ tokens. **Heavy Hitter Oracle** (`H2O`) (Zhang et al., 2023b) is a KV cache compression policy that dynamically balances recent and "Heavy Hitter" (H2) tokens. H2O maintains a fixed cache size across all layers. **SnapKV** (`SnapKV`) (Li et al., 2024) compresses KV caches by selecting and clustering important tokens for each attention head. Unlike H2O, SnapKV captures attention signals using patterns from a localized window and applies a more nuanced clustering algorithm, including a pooling layer. **PyramidKV** (`PyramidKV`) (Zhang et al., 2024b) proposes a layer-wise retention strategy that reduces cache size per layer based on depth. For KV cache selection, PyramidKV employs the same method as SnapKV. **ZigZagKV** (proposed method) is detailed in Section 4.



(a) Mistral-7B-Instruct-v0.3

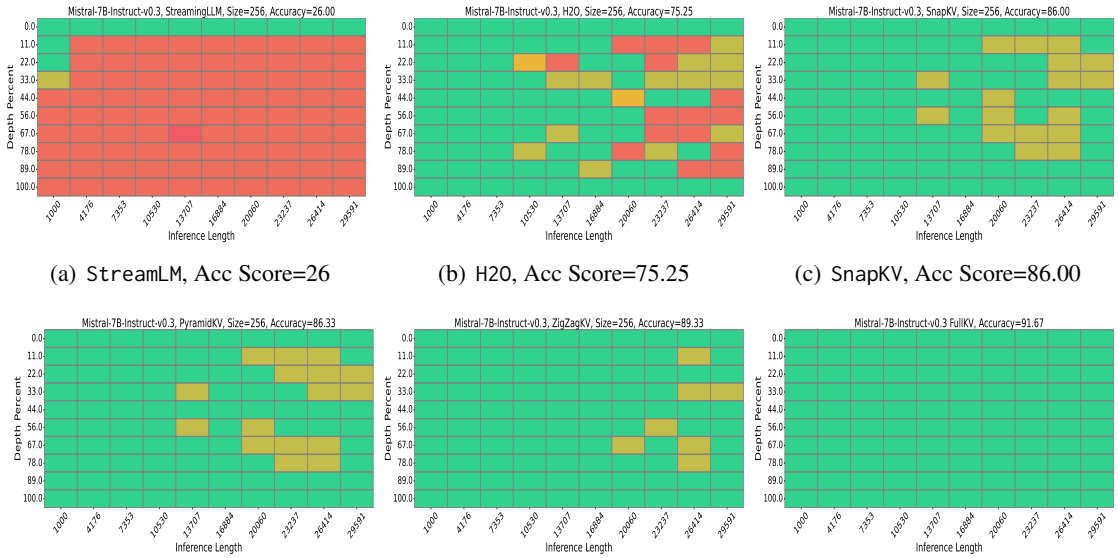


(b) LLaMa-3.1-8B-Instruct

Figure 4: The evaluation results from Needle-in-a-HayStack testing across 96, 128, 256, 512 and 1024 budget sizes on Mistral-7B-Instruct-v0.3 and LLaMa-3.1-8B-Instruct. Proposed method `ZigZagKV` outperforms `H2O`, `SnapKV`, `PyramidKV` and `StreamLM`, especially in limited budget sizes.

5.4 Main Results

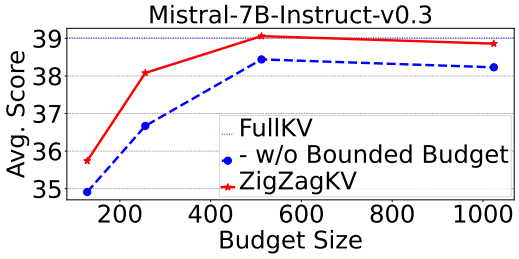
Results on Needle-in-a-Haystack Testing We first compare the proposed method with previous approaches on the Needle-in-a-Haystack test, as shown in Figure 4 and Figure 5. `ZigZagKV` consistently outperforms previous methods under almost all constrained cache budget settings, particularly when the average budget is limited. When the mean budget size is 256, `ZigZagKV` achieves an accuracy of 89.33%, closely matching the retrieval accuracy of `FullKV` . In contrast, `StreamLM` and `H2O` show a lower performance. Notably, `ZigZagKV` only requires an average budget of 256, even for 30K context, while `FullKV` requires retaining the entire KV cache to inference.



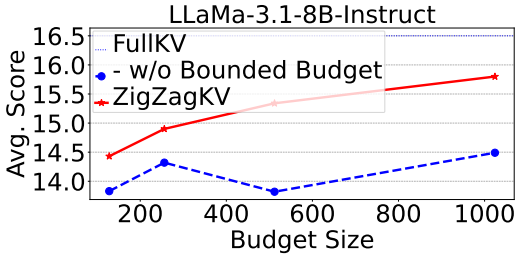
(a) StreamLM, Acc Score=26 (b) H2O, Acc Score=75.25 (c) SnapKV, Acc Score=86.00

(d) PyramidKV, Acc Score=86.33 (e) ZigZagKV, Acc Score=89.33 (f) FullKV, Acc Score=91.67

Figure 5: Performance comparison for Needle-in-a-Haystack testing with a budget size of 256 on Mistral-7B-Instruct-v0.3 and LLaMa-3.1-8B-Instruct. The x-axis represents the length of the document, while the y-axis indicates the position where the needle is located. A red cell indicates that the model fails to recall the information in the needle, whereas a green cell indicates success.



(a) Mistral



(b) LLaMa

Figure 6: Ablation studies between ZigZagKV w/ Bounded Budget and w/o Bounded Budget on both Mistral and LLaMa on 2WikiMQA Dataset.

Results on LongBench To assess the performance of the proposed method across various tasks, we conduct experiments using LongBench. The results are depicted in Table 1 and Table 2. Similarly, ZigZagKV demonstrates improvements over the four baseline methods, achieving higher average scores across multiple tasks. In particular, ZigZagKV outperforms FullKV using only a mean KV cache size of 128 on the TriviaQA few-shot learning task. This demonstrates that the proposed method reduces memory overhead and captures more information from few-shot examples, highlighting its potential for further study in in-context learning tasks.

5.5 Analysis and Ablation Studies

ZigZagKV Preserves More Attention Information. To investigate whether the proposed method achieves more attention information, as described in Section 3, we calculated the average deviation from achieving 0.9 attention across various cache settings, which we refer to as attention loss. The results are presented in Table 3. Both SnapKV and PyramidKV exhibit higher attention loss. This is because they either apply uniform treatment for all layers or allocate smaller cache budgets to higher layers, resulting in a significant deviation in attention scores compared to FullKV. In contrast, ZigZagKV substantially

	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shotLearning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
FullKV	28.7	41.6	52.9	49.4	39.0	28.6	34.8	25.6	27.9	76.0	88.6	47.4	5.5	98.0	61.4	62.6	48.0
KV Size = 128																	
StreamLM	20.0	19.9	29.4	37.0	24.1	16.8	18.9	17.2	20.2	46.5	73.7	18.1	4.5	67.8	32.1	36.6	30.2
H2O	25.3	29.1	44.6	45.7	32.9	23.8	21.9	**23.0**	21.2	34.0	88.2	**44.2**	6.0	94.0	53.7	52.5	40.0
SnapKV	25.3	30.6	**48.2**	47.6	35.1	21.3	21.8	22.2	21.7	69.5	88.6	43.7	6.0	93.5	56.0	**55.2**	42.9
PyramidKV	25.6	**31.5**	47.9	**47.9**	35.2	**25.4**	21.9	22.3	21.8	**70.0**	88.2	43.8	4.0	**94.5**	55.7	55.0	43.2
ZigZagKV	**26.0**	30.4	48.1	47.7	**35.7**	24.7	**22.0**	22.6	**21.9**	69.5	**89.0**	44.0	**6.5**	93.5	**56.1**	55.1	**43.3**
KV Size =256																	
StreamLM	20.1	21.4	32.5	37.0	25.4	16.8	21.8	16.9	23.6	57.5	72.4	18.5	4.0	65.0	34.4	37.1	31.5
H2O	24.7	29.9	47.0	46.0	35.5	25.2	22.9	**24.1**	22.8	35.0	88.6	44.6	4.5	**96.5**	56.4	54.5	41.1
SnapKV	26.8	33.2	52.2	47.9	37.5	24.2	23.8	23.3	23.4	73.0	88.8	45.0	5.0	96.0	58.3	58.2	44.8
PyramidKV	26.7	**34.0**	51.6	47.5	38.1	**27.4**	**24.0**	23.6	23.5	**74.0**	88.7	**45.0**	5.0	96.0	**58.4**	58.1	45.1
ZigZagKV	**27.6**	33.4	**53.4**	**48.6**	**38.1**	27.3	24.0	23.6	**23.7**	73.0	**88.9**	45.0	**6.0**	96.0	58.3	**58.5**	**45.3**
KV Size = 512																	
StreamLM	20.8	22.2	34.4	37.9	26.1	16.1	25.2	18.5	**26.5**	65.5	71.5	18.1	3.3	67.5	36.9	37.5	33.0
H2O	25.3	33.4	50.5	48.4	**39.3**	**27.2**	24.3	24.0	24.4	39.5	88.7	46.1	5.5	**97.0**	59.0	57.2	43.1
SnapKV	27.4	36.4	54.0	49.7	38.7	26.7	25.8	24.5	25.2	**77.8**	89.3	46.7	5.0	94.5	60.2	**60.9**	46.4
PyramidKV	26.7	36.1	53.7	**50.1**	38.4	27.0	25.8	24.5	25.3	74.5	89.4	46.4	5.5	96.0	60.2	60.5	46.3
ZigZagKV	**27.8**	**36.9**	**54.2**	49.7	39.1	26.9	**25.9**	**24.9**	25.2	75.0	**89.4**	**46.7**	**5.5**	96.5	**60.3**	60.7	**46.5**
KV Size = 1024																	
StreamLM	22.0	28.1	41.2	37.9	27.0	17.2	27.9	19.9	**27.2**	71.5	70.3	19.0	5.4	69.0	41.0	38.3	35.2
H2O	**27.3**	34.8	51.2	49.2	37.0	26.8	26.1	24.9	26.3	48.0	**89.3**	46.5	5.0	97.5	60.5	58.6	44.3
SnapKV	26.8	37.8	52.7	49.2	38.9	28.1	28.2	**25.3**	26.8	76.0	89.0	46.2	5.5	97.5	61.3	62.2	47.0
PyramidKV	26.7	37.7	52.7	49.3	**38.9**	27.9	28.2	25.0	26.8	76.0	89.2	46.4	5.5	97.5	61.4	61.9	47.0
ZigZagKV	26.9	**37.8**	**53.4**	**49.6**	38.9	**28.1**	**28.6**	25.1	26.9	**76.0**	89.2	**46.5**	**5.5**	**98.0**	**61.5**	**62.2**	**47.1**
KV Size = 2048																	
StreamLM	23.3	31.8	47.1	38.0	29.5	18.9	30.1	20.2	27.3	73.0	70.0	19.0	5.3	72.2	45.6	39.8	36.9
H2O	27.7	38.8	52.7	49.3	38.4	27.4	29.1	25.1	27.3	63.5	89.1	47.0	5.5	98.0	61.3	61.1	46.3
SnapKV	28.2	40.5	**53.1**	49.7	38.6	28.3	30.8	**25.6**	27.5	75.5	88.9	**47.3**	5.5	98.0	**62.0**	62.0	47.6
PyramidKV	**28.2**	**40.8**	52.8	49.7	**38.8**	**28.5**	30.6	25.5	27.5	75.5	89.1	47.2	5.5	98.0	61.9	62.2	47.6
ZigZagKV	28.2	40.8	53.0	**49.9**	38.6	28.5	**30.8**	25.4	**27.6**	**75.5**	**89.1**	47.2	**5.5**	**98.0**	62.0	**62.3**	**47.7**

Table 1: Comparison Based on Mistral-7B-Instruct-v0.3 Among 16 Datasets

reduces attention loss in the Mistral and LLaMa models, minimizing the attention score gap between the proposed method and FullKV. This indicates that ZigZagKV preserves more attention information compared to the baseline methods.

ZigZagKV Maintains More Hidden State Information. Furthermore, to analyze whether the proposed method maintains a more stable hidden state output as described in Section 3, we compared the difference between the output of PartialKV Inference and FullKV Inference using the metric $1 - \text{similarity}(y, \hat{y})$, termed as output loss. We then computed the average output loss for each layer. The results are illustrated in Table 4. The mean output loss of ZigZagKV is the lowest among all methods and models, except when the mean budget is set to 128. This indicates that, compared to baseline methods, the proposed method effectively maintains more stable output by setting the budget size based on layer uncertainty.

Ablation Studies on Bounded Budget. To evaluate the effectiveness of the bounded budget operation in our proposed method, we compare the performance of our method with and without using this strategy. As shown in Figure 6, utilizing the bounded budget strategy enhances performance on both Mistral and LLaMa across various budget sizes.

Computational Overhead. To evaluate the computational cost differences between the ZigZagKV and the baseline method, we measure the latency of StreamLM, PyramidKV, and ZigZagKV, as presented in Table 5. The latency tests indicate that PyramidKV and ZigZagKV demonstrate similar performance. In contrast, StreamLM exhibits faster processing speeds, while StreamLM is faster but has a performance drop.

	Single-Doc. QA			Multi-Doc. QA			Summarization			Few-shotLearning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
FullKV	28.8	13.0	27.5	16.7	16.5	11.4	34.5	23.5	26.9	72.5	91.7	43.8	7.1	97.7	65.1	58.7	39.7
KV Size = 128																	
StreamLM	11.7	4.9	14.5	10.0	10.4	5.9	21.5	17.9	20.6	43.5	71.1	17.3	**9.5**	72.7	40.5	37.0	25.6
H2O	**23.1**	7.7	19.0	13.2	13.4	8.1	22.3	22.1	20.1	39.0	90.2	**40.7**	7.5	93.7	56.6	48.9	32.9
SnapKV	21.3	8.4	20.6	14.6	14.1	8.4	22.2	**22.6**	21.5	62.0	90.2	40.0	8.1	92.8	60.7	50.6	34.9
PyramidKV	21.3	**8.7**	21.0	13.9	13.8	**9.2**	22.2	22.5	21.8	**63.5**	90.3	40.1	8.0	93.9	58.7	50.0	34.9
ZigZagKV	21.4	8.6	**21.6**	**15.0**	**14.4**	8.7	**22.6**	22.5	**22.1**	62.0	**90.8**	40.5	8.7	**94.3**	**61.1**	**51.7**	**35.4**
KV Size =256																	
StreamLM	14.0	5.6	14.8	10.2	9.7	6.2	23.8	18.0	22.9	54.5	70.6	17.7	**8.9**	76.7	42.2	37.4	27.1
H2O	25.0	7.8	20.2	14.7	13.4	9.1	23.3	22.8	21.1	39.0	90.6	41.5	7.1	93.0	60.0	49.8	33.7
SnapKV	24.2	9.4	23.2	15.1	14.7	9.2	24.1	23.1	23.2	70.0	91.4	41.1	7.2	**95.9**	62.0	53.7	36.7
PyramidKV	24.4	9.2	**23.4**	14.7	14.8	9.3	**24.3**	23.2	23.3	70.0	91.4	**41.7**	7.1	95.8	61.6	53.1	36.7
ZigZagKV	**25.5**	**9.5**	23.3	**15.2**	**14.9**	**9.9**	24.2	**23.2**	**23.5**	**70.0**	**91.6**	41.6	7.5	94.5	**62.1**	**53.8**	**36.9**
KV Size = 512																	
StreamLM	12.8	6.4	19.4	10.6	10.1	6.3	25.9	19.0	24.7	60.5	71.9	18.7	**8.1**	79.6	43.9	38.9	28.5
H2O	23.9	8.5	21.5	14.3	13.6	9.6	24.3	22.6	23.4	41.0	91.6	41.5	7.6	94.3	61.5	51.7	34.4
SnapKV	25.2	**11.3**	25.1	15.1	**15.7**	**9.9**	26.1	23.1	**24.7**	70.5	91.7	41.4	7.7	96.2	63.8	**55.6**	37.7
PyramidKV	25.9	11.1	24.7	**15.5**	15.5	9.8	26.1	23.3	24.6	70.5	**91.9**	**41.7**	7.8	96.3	63.7	54.9	37.7
ZigZagKV	**26.1**	11.2	**25.2**	15.5	15.3	9.6	**26.3**	**23.5**	24.6	**70.5**	91.7	41.5	8.1	**96.8**	**64.2**	55.2	**37.8**
KV Size = 1024																	
StreamLM	13.0	7.4	20.9	12.1	10.6	7.1	27.8	19.3	**26.0**	67.5	74.0	19.5	8.1	79.9	45.8	39.0	29.9
H2O	24.7	10.0	24.1	14.8	14.9	9.9	26.1	23.2	25.5	45.0	91.7	42.4	**8.1**	95.0	63.3	54.6	35.8
SnapKV	**28.8**	11.8	**27.3**	15.8	15.6	10.8	28.3	23.6	25.8	70.0	91.7	43.0	7.4	97.6	63.9	56.6	38.6
PyramidKV	28.2	11.7	26.9	16.1	15.7	**10.9**	**28.5**	23.7	25.8	70.0	91.7	**43.1**	7.4	97.8	63.9	56.9	38.6
ZigZagKV	28.8	**12.0**	26.8	**16.2**	**15.8**	10.7	28.4	**23.8**	25.8	**70.0**	**91.7**	43.1	7.8	**97.8**	**64.0**	**56.9**	**38.7**
KV Size = 2048																	
StreamLM	13.6	10.1	23.4	11.7	12.5	7.2	29.9	19.8	**26.7**	68.5	79.5	21.1	**8.2**	64.3	54.1	40.4	30.7
H2O	28.0	11.3	25.5	16.0	15.3	10.4	28.7	23.3	26.6	56.5	**91.6**	**43.0**	7.9	96.7	64.7	57.2	37.7
SnapKV	29.2	12.4	**27.2**	16.6	16.3	11.3	30.4	23.5	26.6	71.0	91.5	42.8	7.7	**97.7**	64.9	58.2	39.2
PyramidKV	29.2	12.4	27.1	16.6	16.5	**11.6**	30.6	23.6	26.3	71.0	91.5	42.5	7.7	97.6	64.7	58.3	39.2
ZigZagKV	**29.4**	**12.7**	27.1	**16.6**	**16.5**	11.5	**30.8**	**23.7**	26.7	**71.0**	91.5	42.7	7.5	97.6	**65.0**	**58.4**	**39.3**

Table 2: Comparison Based on LLaMA-3.1-8B-Instruct Among 16 Datasets

Model	Mistral			LLaMa		
Budget	128	256	512	128	256	512
SnapKV	2.71	1.54	0.89	1.76	0.90	0.46
PyramidKV	2.96	1.59	0.89	1.91	0.88	0.42
ZigZagKV	**2.45**	**1.25**	**0.61**	**1.50**	**0.64**	**0.23**

Table 3: Attention loss of Mistral-7B-Instruct-v0.3 and LLaMa-3.1-8B-Instruct on 2WikiMQA Dataset.

Model	Mistral			LLaMa		
Budget	128	256	512	128	256	512
SnapKV	2.55	1.51	0.85	**2.91**	1.67	0.88
PyramidKV	2.98	1.60	0.87	3.26	1.76	0.90
ZigZagKV	**2.54**	**1.50**	**0.83**	2.92	**1.67**	**0.88**

Table 4: Hidden state loss of Mistral-7B-Instruct-v0.3 and LLaMa-3.1-8B-Instruct on 2WikiMQA Dataset.

Method	**Average Latency (s)**
StreamingLM	4.59
PyramidKV	6.56
ZigZagKV	6.50

Table 5: Average Latency on NarrativeQA of LLaMA-3.1-8B-Instruct

6 Related Work

Existing KV cache compression techniques can be broadly divided into two categories: fixed policies and adaptive policies.

For fixed policies, Xiao et al. (2023) and Han et al. (2024) suggest that the initial tokens often receive consistently high attention weights across layers and heads. Therefore, they propose reducing the memory required for the KV cache by retaining only the first few tokens and local tokens. For adaptive policies , most approaches

select important tokens based on attention weights. Liu et al. (2024) introduce the "persistence of importance" hypothesis, suggesting that tokens with significant influence at one step will continue to impact future generations. Zhang et al. (2023b); Oren et al. (2024) employ cumulative normalized attention scores to determine which tokens to retain while preserving recent tokens due to their strong correlation with the current generation. Li et al. (2024) compress the KV cache by selecting and clustering necessary tokens based on the attention scores from the last segment of tokens.

While these methods differ in selecting tokens for KV cache retention, they generally apply a uniform budget size across layers, even though the optimal budget size may vary. Recently, some studies have explored budget size allocation across different layers (Zhang et al., 2024b; Yang et al., 2024; Wan et al., 2024), but these approaches overlook the need for a minimum budget size to preserve essential information.

Unlike PyramidKV, which heuristically allocates more cache in the lower layers and less in the higher ones, ZigZagKV leverages layer uncertainty to allocate the cache budget. As illustrated in Figure 2 of the submission, the largest LMBA might not occur in the highest layers, which can lead to more cache being allocated to the middle layers. In PyramidKV, the cache sizes for all intermediate layers are set according to an arithmetic sequence. In contrast, with ZigZagKV , the cache sizes for all layers vary depending on the context and models.

7 Conclusion

In this paper, we pay attention to the variation in minimum budget sizes required to retain information across different layers. A comprehensive analysis reveals that the necessary budget size differs across layers from the perspectives of attention mechanisms and hidden state outputs. Building on these findings, we propose a training-free method that dynamically allocates budget sizes based on layer uncertainty, effectively reducing information loss during PartialKV inference. Experiments conducted on two benchmarks and several models demonstrate the effectiveness of our proposed approach.

Limitations

This paper primarily analyzes two widely-used decoder-only LMs, LLaMa (Dubey et al., 2024) and Mistral (Jiang et al., 2023). It does not include a validation study of encoder-decoder and encoder-only architectures.

Acknowledgements

We would like to thank the anonymous reviewers and meta-reviewer for their insightful suggestions. This work was supported by the National Natural Science Foundation of China under Grant U23B2055 and 62276077, and Shenzhen Science and Technology Program under Grant ZDSYS20230626091203008.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *ArXiv preprint*, abs/2308.14508.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. 2024. Nacl: A general and effective kv cache eviction framework for llms at inference time. *ArXiv preprint*, abs/2408.03675.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *ArXiv preprint*, abs/2407.21783.

Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. 2024. Data engineering for scaling language models to 128k context. *ArXiv preprint*, abs/2402.10171.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. Lm-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human*

Language Technologies (Volume 1: Long Papers), pages 3991–4008.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Hui Huang, Bing Xu, Xinnian Liang, Kehai Chen, Muyun Yang, Tiejun Zhao, and Conghui Zhu. 2024. Multi-view fusion for instruction mining of large language model. *Information Fusion*, 110:102480.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *ArXiv preprint*, abs/2310.06825.

Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. 2021. How can we know when language models know? on the calibration of language models for question answering. *Transactions of the Association for Computational Linguistics*, 9:962–977.

Ehsan Kamalloo, Nouha Dziri, Charles LA Clarke, and Davood Rafiei. 2023. Evaluating open-domain question answering in the era of large language models. *ArXiv preprint*, abs/2305.06984.

Greg Kamradt. 2023. Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack.

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

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

Matanel Oren, Michael Hassid, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state rnns. *ArXiv preprint*, abs/2401.06104.

Bowen Peng and Jeffrey Quesnelle. 2023. Ntk-aware scaled rope allows llama models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation. https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. Yarn: Efficient context window extension of large language models. *ArXiv preprint*, abs/2309.00071.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Siyu Ren and Kenny Q Zhu. 2024. On the efficacy of eviction policy for key-value constrained generative language model inference. *ArXiv preprint*, abs/2402.06262.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open foundation models for code. *Preprint*, arXiv:2308.12950. ArXiv: 2308.12950.

Dan Su, Yan Xu, Genta Indra Winata, Peng Xu, Hyeondey Kim, Zihan Liu, and Pascale Fung. 2019. Generalizing question answering system with pre-trained language model fine-tuning. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 203–211, Hong Kong, China. Association for Computational Linguistics.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv preprint*, abs/2307.09288.

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

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

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

Chen Zhang, Dawei Song, Zheyu Ye, and Yan Gao. 2023a. Towards the law of capacity gap in distilling language models. *ArXiv preprint*, abs/2311.07052.

Chen Zhang, Meizhi Zhong, Qimeng Wang, Xuantao Lu, Zheyu Ye, Chengqiang Lu, Yan Gao, Yao Hu, Kehai Chen, Min Zhang, et al. 2024a. Modification: Mixture of depths made easy. *arXiv preprint arXiv:2410.14268*.

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

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2023b. H 2 o: Heavy-hitter oracle for efficient generative inference of large language models.(2023). *arXiv preprint cs.LG/2306.14048*.

Meizhi Zhong, Chen Zhang, Yikun Lei, Xikai Liu, Yan Gao, Yao Hu, Kehai Chen, and Min Zhang. 2024. Understanding the rope extensions of long-context llms: An attention perspective. *ArXiv preprint*, abs/2406.13282.