UNCOMP: Can Information Compression Uncover Sparsity? — A Compressor Design from an Uncertainty-Aware Perspective

Anonymous ACL submission

Abstract

Deploying large language models (LLMs) for long-context inference remains challenging due to their substantial memory and computational demands. While techniques such as Key-Value (KV) cache compression are designed to reduce memory usage, they often neglect the structured sparsity inherent in the relationship between hidden states and their corresponding KV cache. In this work, we explore the role of uncertainty as a potential indicator of sparsity within LLMs. We propose UNCOMP, an uncertainty-aware framework that leverages *truncated matrix entropy* to identify areas of low information content, thereby revealing sparsity patterns that can be used for adaptive compression. Unlike traditional methods that apply uniform compression, UNCOMP dynamically adjusts its approach to compression, guided by uncertainty measures that reflect the importance of various model components. Our analysis shows that sparsity patterns, when derived from uncertainty estimates, can be exploited to reveal special long-range dependencies, such as retrieval heads and retrieval layers. This perspective not only enhances our understanding of how compression can be optimized but also provides new insights into the inherent sparsity of LLMs during long-context inference. By focusing on uncertainty to analyze the sparsity pattern in detail, UNCOMP reduces the KV cache size to 4.74% of the original, achieves a 6% prefill speedup, and improves throughput by 6.4× — not only delivering strong lossless compression performance, but also validating the effectiveness of the underlying theoretical tool. Our codes are submitted with the paper.

1 Introduction

The development of large language models (LLMs) drives remarkable progress in natural language processing (Achiam et al., 2023; Kaplan et al., 2020), enabling tasks ranging from text generation to complex reasoning. However, deploying LLMs for long-context inference remains resource-intensive, as they demand substantial memory and computation (Shazeer et al., 2017). A commonly adopted optimization, the *KV cache* (Pope et al., 2023; Liu et al., 2024b), stores keys and values from previous tokens to avoid redundant computations. Nonetheless, maintaining a full KV cache for long sequences imposes considerable memory overhead, which poses a bottleneck (Liu et al., 2024b).

To address this, several compression strategies emerge: *i*) **Eviction** (Ge et al., 2023; Zhang et al., 2024d; Li et al., 2024; Zhang et al., 2024c), *ii*) **Merging** (Liu et al., 2024b; Wan et al., 2024; Wang et al., 2024; Zhang et al.), *iii*) **Quantization** (Hooper et al., 2024; Zhang et al., 2024a; Liu et al., 2024e), and *iv*) **Head Pruning** (Ainslie et al., 2023; Shazeer, 2019; Liu et al., 2024a; Yu et al., 2024; Brandon et al., 2024). However, these methods typically operate on sparsity along a single axis—such as pruning attention heads or compressing individual layers—without fully capturing how sparsity emerges across the model's hierarchical structure, overlooking its complexity from different layers and attention heads. To bridge this gap, we introduce Matrix Information Theory (Zhang et al., 2023) and propose *truncated matrix entropy* as a unified framework that connects uncertainty and sparsity in a principled manner.

Specifically, existing methods (Zhang et al., 2024d; Xiao et al., 2023) often overlook the sparsity pattern shared between hidden states—typically the outputs of the multi-layer perceptron (MLP)—and the KV cache. Rather than viewing sparsity as a localized artifact of attention mechanisms, we interpret it as an indicator of low-information regions distributed throughout the hidden states and KV cache. This shift—from the empirical observation layer and head sparsity (Jiang et al., 2024; Wu et al., 2024; Xiao et al., 2024; Cai et al., 2024) to uncertainty-aware compression—reveals latent redundancies during long-

context inference and guides adaptive hidden state compression for faster prefill and KV cache eviction reducing. Our key contributions are as follows:

1. We propose a novel uncertainty-aware compression framework grounded in *truncated matrix entropy*, uncovering the relationship between information compression patterns and sparsity patterns in LLMs. Furthermore, we provide a detailed empirical study of the connection between information compression patterns and compression algorithms.

2. We design a two-stage compression framework that jointly compresses hidden states and the KV cache. We are the first to indirectly compress the KV cache and accelerate the prefill stage by compressing the hidden states, achieving a 60% speedup in the prefill stage, a 4.74% compression ratio, and a 6.4× improvement in throughput with negligible performance degradation.

3. We demonstrate that our method maintains performance even under aggressive compression regimes, including settings where heads are entirely removed. In the needle-in-a-haystack task, UNCOMP surpasses the full-size KV cache baseline at a 9.38% compression ratio.

2 Related Work

2.1 Attention-Based Token Eviction

Early works identify distinctive attention patterns in the KV cache, such as the *attention sink* (Liu et al., 2024c; Xiao et al., 2023). In addition, prior studies (Cai et al., 2024; Yang et al., 2024a) empirically observe that sparsity increases with model depth. Distinct sparsity patterns—such as retrieval heads (Xiao et al., 2024; Wu et al., 2024)—are also observed across attention heads. However, the underlying mechanisms driving these sparsity patterns remain to be fully understood.

Recent methods employ cumulative attention scores for selecting subsets of tokens (Zhang et al., 2024c; Li et al., 2023; Jiang et al., 2024; Zhang et al., 2024d; Ge et al., 2023; Sheng et al., 2023; Liu et al., 2024d; Li et al., 2024). These methods apply a fixed compression ratio to sparsity patterns observed in the layers of the KV cache. However, they overlook the compression of hidden states, as well as differences in retrieval behaviors across layers. This not only degrades the performance of *retrieval layers*, but also misses the opportunity to accelerate computation in the prefill stage by failing to compress the hidden states.

2.2 Information Compression Behavior

The sparsity patterns discussed in Section 2.1 are related to the model's internal information compression behavior (Feng et al., 2022). In this paper, we provide a detailed categorization and analysis of the compression patterns across different parameter matrices. Recent work (Delétang et al., 2023) reveals that models exhibit spontaneous compression behavior during training. Similar phenomena are reported in Tao et al. (2024); Huang et al. (2024). These observations, however, rarely connect information compression patterns with sparsity patterns. This motivates us to explore the model's information compression behavior—through entropy increases or decreases—as a means to analyze its sparsity patterns. To achieve this, we introduce Matrix Information Theory (Zhang et al., 2023), a theoretical tool for understanding the information compression behavior.

3 Method

In this section, to explore the information compression patterns in the model, we derive the definition of *truncated matrix entropy* and reveal the connection between information compression patterns and sparsity patterns, leading to compression strategy.

3.1 Preliminary

To define *truncated matrix entropy*, we first introduce the matrix entropy of the token matrix (either from a layer or a head). Let the token matrix be $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$, where $\mathbf{x}_i \in \mathbb{R}^D$ is the i-th token vector, and N is the sequence length. The covariance matrix $\mathbf{\Sigma_X} \in \mathbb{R}^{D\times D}$ is computed as:

$$\mathbf{\Sigma_X} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T, \quad (1)$$

where $\bar{\mathbf{x}}$ is the mean vector of the sequence $\mathbf{X}$. Based on this covariance matrix, we derive the definition of matrix entropy. Specifically, following Giraldo et al. (2014), the matrix entropy based on $\mathbf{\Sigma_X}$ is defined as:

Lemma 1 *As $\alpha \to 1$ (the entropy measure depending on α), we obtain the definition of the von Neumann (matrix) entropy (Von Neumann, 2013):*

$$H(\mathbf{\Sigma_X}) = -Tr\left(\mathbf{\Sigma_X}\log\left(\mathbf{\Sigma_X}\right)\right). \quad (2)$$

Lemma 2 *Let $\mathbf{\Sigma_X}$ be a symmetric positive definite matrix with eigenvalues $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_D)^T$. The matrix entropy of $\mathbf{\Sigma_X}$ can be expressed as:*

$$H(\mathbf{\Sigma_X}) = -\sum_{i=1}^{D} \sigma_i \log \sigma_i, \tag{3}$$

where D is the dimension of the covariance matrix. We define matrix entropy on the token matrix $\mathbf{X}$ and provide the proof in Appendix G.

To provide an intuition of its role in the token matrix, we introduce effective rank (Roy and Vetterli, 2007), which links matrix entropy to dimensionality. For the specific definition, please refer to Appendix G. Recent works (Zhang et al., 2023; Zhuo et al., 2023) investigate the relationship between matrix entropy and effective rank. Inspired by these studies, we adopt effective rank to quantify the effective information across heads and layers to explore the model's information compression patterns. Based on these patterns, we set different compression ratios for different heads and layers. We define the compression ratio ρ for each $\mathbf{X}$ as the ratio between the compressed length $\hat{N}$ and the original sequence length N, i.e.,

$$\rho = \frac{\hat{N}}{N}. \tag{4}$$

3.2 Truncated Matrix Entropy

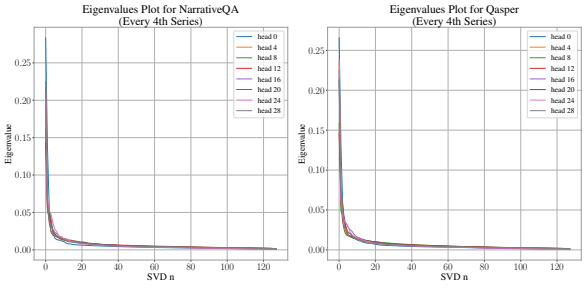

Figure 1: The spectrum of $\mathbf{\Sigma}_{Q_m}$ across two datasets in LongBench and various heads.

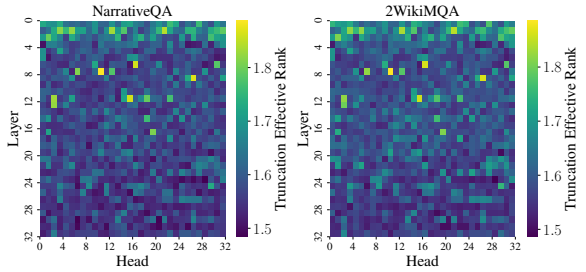

Figure 2: The heatmap of $\text{erank}_k(\mathbf{\Sigma}_{Q_m})$ across different layers and heads.

To uncover *information compression patterns* in the token matrix, how do we determine which matrix—key (K_m), query (Q_m), value (V_m), or hidden state (H_m)—best captures the compression patterns in the token matrices? To answer this question, we propose *truncated matrix entropy*.

Observation To gain insight, we first focus on the *spectrum* of $\mathbf{\Sigma}_{Q_m}$ (Tang et al., 2024b) across different heads in the model's final layer. Figure 1 reveals: *i)* The initial part of the eigenvalue distribution varies significantly, suggesting that only a small portion of the feature components are active. *ii*) Eigenvalue distributions across heads differ significantly in the leading part, prompting us to use this part to calculate the model's effective rank and distinguish head types.

A key observation from Figure 2 is that, in different layers, there are heads with abnormally high entropy, with most of them found in the middle layers (9-15). Later empirical experiments show that these high-entropy heads and their corresponding layers are associated with special sparsity patterns that have long-range dependency (retrieval layers).

Discussion *By Lemma 2, the covariance matrix used for computing the matrix entropy must be positive definite.* Therefore, we calculate the effective rank of $\mathbf{\Sigma}_{Q_m}$ using a positive definite submatrix. Furthermore, as shown in Figure 1, we observe a distinct elbow point (Thorndike, 1953), where the eigencomponents preceding the elbow point represent the principal components of the matrix.

Based on the above observations, we select the top-k eigenvalues before the elbow point (Kaiser, 1960) and compute the effective rank. This leads to the definition of *truncated effective rank*, which quantify uncertainty through the effective rank of a submatrix:

$$H_k(\mathbf{\Sigma_X}) = -\sum_{i=1}^{k} \sigma_i \log \sigma_i, \tag{5}$$

$$\text{erank}_k(\mathbf{\Sigma_X}) = \exp\left(H_k\left(\mathbf{\Sigma_X}\right)\right). \tag{6}$$

With $\text{erank}_k(\mathbf{\Sigma_X})$ defined, we begin classifying the compression patterns and exploring the estimation of the compression ratio of H_m across layers and K_m, V_m across heads.

Observation We visualize the entropy variation trends of Q_m, K_m, and V_m across different layers and datasets in Figure 3. The figure reveals the following: *i*) Q_m and K_m exhibit more pronounced trends of entropy increase or decrease compared to V_m (notably, V_m and H_m demonstrate similar compression patterns; see Appendix A), suggesting that Q_m and K_m serve as stronger indicators of information compression than V_m and

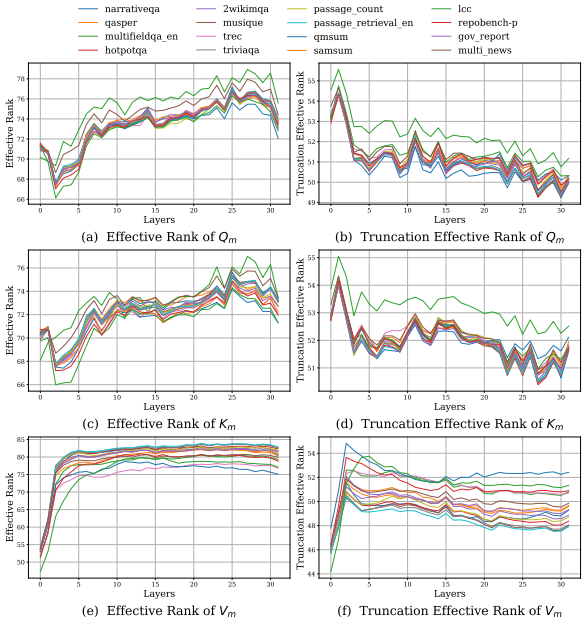

Figure 3: Effective rank and truncated effective rank for the Q_m, K_m, and V_m across different layers.

H_m. *ii*) *Truncated effective rank* and full effective rank display markedly different variation patterns, particularly in Q_m and K_m, *highlighting fundamentally different behaviors in information compression*. *iii*) As model depth increases, the inter-layer $\text{erank}_k(\mathbf{\Sigma}_{Q_m})$ and $\text{erank}_k(\mathbf{\Sigma}_{K_m})$ show a decreasing trend, indicating increasingly sparse structures (Wang et al., 2023). *iv*) From the initial to the final layer, the *truncated matrix entropy* of Q_m decreases more significantly than that of K_m, and its average entropy is also lower, reinforcing its role as an effective indicator.

Discussion Based on the above observations, to better connect information compression patterns with sparsity patterns, we select Q_m as a proxy to estimate the sparsity characteristics of K_m, V_m, and H_m and design the two-stage compression strategy presented in the next section.

3.3 Uncertainty-Aware Compression Strategy

In this section, we provide a detailed explanation of how *truncated matrix entropy* is used to guide compression. We first introduce the preparation stage, followed by a two-stage compression strategy, along with the mapping between information compression and sparsity patterns. The workflow of our method is illustrated in Figure 4.

3.3.1 Preparation Stage

Observation We first observe in Figure 2 that attention head information compression patterns remain consistent across datasets, suggesting that the sparsity pattern of heads is not data-dependent.

Design Since the head information compression pattern is not data-dependent, we first sample 500 data points from Wikitext2 (Merity, 2016) before inference to pre-group the heads. These samples are used to group the model heads and set compression ratios for each group, which helps identify the retrieval heads in the KV cache. Specifically, after inputting $d = 500$ data points into the model, we save the KV cache of all data and calculate the $\text{erank}_k(\mathbf{\Sigma}_{Q_m}^{(i,h)})$ for each head of each data point. Then, we average it to obtain $\hat{\text{erank}}_k(\mathbf{\Sigma}_{Q_m}^{h})$ as the grouping metric for the inference stage:

$$\hat{\text{erank}}_k(\mathbf{\Sigma}_{Q_m}^{h}) = \frac{1}{d}\sum_{i=1}^{d} \text{erank}_k(\mathbf{\Sigma}_{Q_m}^{(i,h)}), \tag{7}$$

where i is the i-th data point, and h is the h-th head.

3.3.2 Layer-Group Compression

For the first-stage compression, we focus on compressing the hidden states H_m and attempt to identify retrieval-layer.

Observation Previous KV cache compression methods (Zhang et al., 2024c,d) do not reduce computation, as they evict the KV cache only after prefill. In contrast, we compress hidden states before generating the KV cache, saving both computation and memory. From Figure 3(b), we observe that some layers show an increase in entropy, indicating that the token matrix is unsuitable for compression as it aggregates information.

Design Specifically, we divide the L layers into C groups, ensuring that the token length within each group remains consistent across layers. By applying the compression patterns in Figure 3(a), we compress layers where inter-layer entropy reduction $\Delta\, c_i$ falls below a threshold ϵ, determining the total compression stages C:

$$\Delta c_i = \text{erank}_k(\mathbf{\Sigma}_{Q_m}^{i}) - \text{erank}_k(\mathbf{\Sigma}_{Q_m}^{i+1}), \tag{8}$$

$$C = \sum_{i=1}^{L-1} \mathbf{1}(\Delta c_i > \epsilon > 0), \tag{9}$$

where $\mathbf{1}$ is the indicator function that evaluates to 1 if the entropy reduction between layer i and $i+1$ exceeds the threshold ϵ, and 0 otherwise. Eq. (8) is a partition function determining the division of the model's layers into C groups. The context size at each subsequent group is calculated by

$$N_{i+1} = N_i + \Delta n, \quad i = 1, 2, \ldots, C-1, \tag{10}$$

where Δn (hyperparameter) represents the increment between consecutive groups. With the token

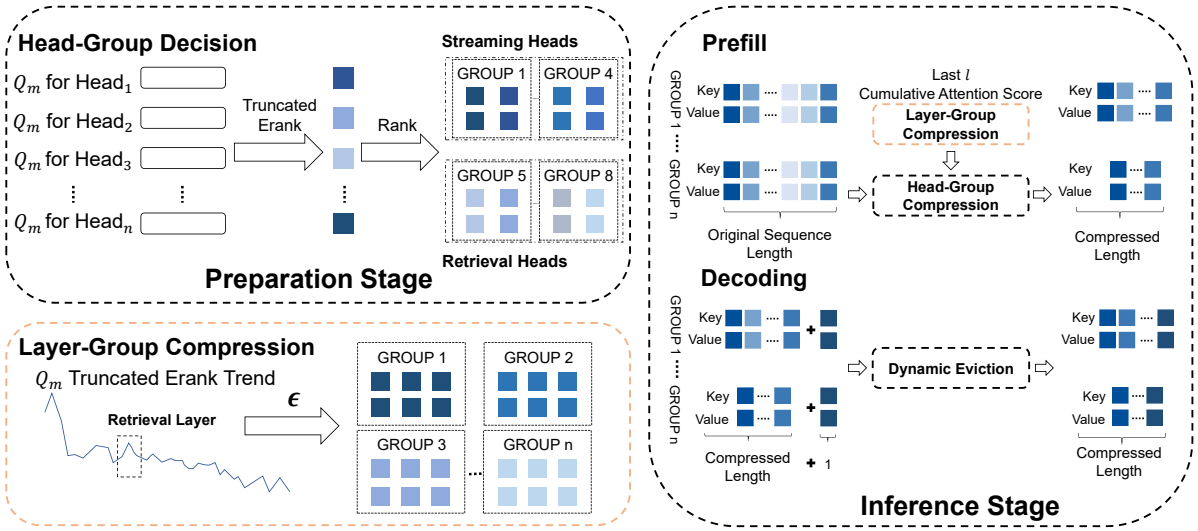

Figure 4: Overview of UNCOMP. Darker colors indicate more retained hidden states and a weaker compression (i.e., a higher ρ) corresponding to higher attention scores.

budget N_i for each group i determined, hidden state eviction starts from the second group, while the first group's hidden states remain full-size to preserve information in early layers.

Hidden State Eviction We observe that $H_m^{(i)}$ and $V_m^{(i)}$ share the same information compression pattern. Based on this observation, we decide to use the attention scores to determine the eviction strategy for $H_m^{(i)}$. From group 2 to group C, we predict token eviction in $H_m^{(i+1)}$ using the attention scores of the last token in the ith layer, retaining the tokens with the highest N_{i+1} attention scores. After generating $H_m^{(i+1)}$, we map $H_m^{(i+1)}$ to the three matrices $Q_m^{(i+1)}$, $K_m^{(i+1)}$, and $V_m^{(i+1)}$. Please refer to Appendix H for more details.

Retrieval Layer We select our retrieval layers based on the maximum entropy increase layer and use average interpolation to merge it with the parameters of the final layer to improve performance.

3.3.3 Head-Group Compression

After the prefill stage ends, we compress the KV cache size of each head to a fixed size $N_{i,1}$. The initial number of heads in each group is consistent.

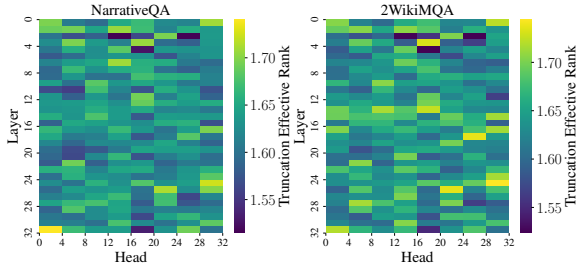

Figure 5: The heatmap of $\text{erank}_k(\mathbf{\Sigma}_{Q_m})$ across different layers and heads in LLaMa3.

Head Information Compression Pattern To further reveal the similar information compression patterns within the group of heads, we visualize the *truncated matrix entropy* distribution across different heads of LLaMa3 in Figure 5. We observe that every four groups of heads exhibit a clearly similar information compression pattern, suggesting they share the same sparsity pattern. This is due to the use of the Group Query Attention (GQA) (Ainslie et al., 2023) technique during training, highlighting the significant influence of the training method on the inference approach.

Observation To further explore the mapping between information compression patterns and sparsity patterns, and for the convenience of ablation, we divide the attention heads into two groups: one with a cache size of 96 and the other with a cache size of 32. We set up three configurations: one where the group with a high truncated matrix entropy cache size is set to 96 and the other to 32, one where the group with a low truncated matrix entropy cache size is set to 96 and the other to 32, and one where all attention heads are randomly divided into two groups. These are referred to as High Entropy, Low Entropy, and Random Group, respectively. *As shown in Table 2, we find that heads with a higher truncated matrix entropy require a higher compression ratio (i.e., weaker compression), which often leads to better performance.*

Design To leverage this sparsity, heads are ranked by $\hat{\text{erank}}_k(\mathbf{\Sigma}_{Q_m}^{(i,h)})$, grouped into M (hyperparameter), and then ordered based on the average $\hat{\text{erank}}_k(\mathbf{\Sigma}_{Q_m}^{(i,h)})$ within each group. To calculate the

Methods	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.	Time (s / sample)
	NtrvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P		
Llama2-7B-chat-hf, KV Size = FULL																		
FullKV	19.34	18.61	35.19	30.66	28.42	10.05	25.19	20.18	25.73	63.00	83.62	41.60	5.00	10.00	61.40	55.45	33.34	0.96
Llama-2-7B-chat-hf, KV Size = 384 , Compressibility is 9.38% (Except CHAI method)																		
H2O	14.96	14.60	17.40	26.72	27.97	6.11	17.83	18.76	20.17	47.00	77.56	39.39	4.50	5.00	57.08	50.31	27.84	0.94
StreamingLLM	13.71	13.68	19.40	26.97	28.03	6.78	15.13	18.87	18.27	46.50	80.02	**40.85**	4.50	5.00	56.84	51.56	27.88	0.88
SnapKV	16.27	17.34	30.37	**33.04**	27.82	9.92	19.34	20.33	22.63	59.50	83.50	38.45	**5.50**	**12.50**	59.18	**55.28**	31.94	0.82
PyramidKV	16.86	18.26	31.01	31.59	27.93	8.69	19.88	20.15	22.43	62.00	83.86	38.98	**5.50**	10.00	58.94	52.80	31.81	0.84
CHAI	16.75	16.91	**34.69**	26.09	20.80	9.20	20.79	20.23	23.33	57.00	75.52	35.67	4.00	6.33	50.10	46.55	29.00	1.51
Quest	17.31	19.55	32.18	30.25	27.20	9.48	22.82	19.25	**25.99**	62.50	83.26	40.37	5.00	5.25	58.81	53.24	32.03	0.96
Double-Sparse	17.27	19.85	32.30	29.45	**28.54**	**9.90**	20.88	19.84	25.38	61.50	83.48	40.56	5.25	8.00	52.27	51.97	31.65	1.02
RazorAttention	16.88	19.33	32.44	31.28	27.81	9.46	**24.29**	20.02	23.19	61.00	83.85	40.28	5.00	12.00	54.71	52.29	32.11	1.21
Ours-group-stage	**17.70**	**20.28**	30.98	30.35	27.37	9.29	22.25	**20.60**	24.04	**63.00**	83.68	38.27	4.50	6.50	58.02	53.79	31.91	**0.57**
Ours-group	17.12	19.20	32.09	30.99	27.88	9.27	23.20	20.10	24.72	62.50	**84.72**	39.33	**5.50**	11.10	**59.71**	54.15	**32.60**	0.83
Llama2-13B-chat-hf, KV Size = FULL																		
FullKV	18.20	26.07	37.06	36.20	32.44	14.19	25.82	20.20	26.00	66.50	87.49	35.93	3.12	11.50	53.29	52.73	34.17	2.21
Llama-2-13B-chat-hf, KV Size = 384 , Compressibility is 9.38% (Except CHAI method)																		
H2O	14.11	18.36	22.78	33.03	27.58	12.94	18.97	18.69	20.37	53.50	85.75	34.15	3.55	6.00	50.97	47.56	29.27	2.57
StreamingLLM	13.23	18.47	23.76	34.50	29.62	11.09	18.67	18.47	17.89	52.50	84.93	32.54	3.55	7.00	40.60	42.86	28.11	1.96
SnapKV	17.09	22.77	34.37	36.73	31.04	13.02	19.70	20.00	22.91	62.00	87.48	**37.44**	**4.05**	11.50	51.76	**51.27**	32.70	1.93
PyramidKV	16.33	22.81	34.19	**37.54**	30.25	13.82	19.79	20.11	23.14	64.50	86.45	36.62	**4.05**	**12.00**	52.06	50.58	32.77	3.50
CHAI	17.06	23.51	31.01	33.70	27.78	11.73	23.03	19.59	24.66	65.00	86.18	15.93	4.00	8.50	45.57	48.74	30.37	2.50
Quest	17.07	**26.36**	34.56	34.50	29.62	13.19	23.79	19.71	25.32	64.00	84.93	36.46	2.62	9.50	**52.12**	49.85	32.73	1.86
Double-Sparse	17.42	25.10	31.85	33.27	29.89	12.61	24.00	19.88	**25.68**	67.00	86.48	35.97	2.85	14.31	51.95	**51.27**	33.10	1.98
RazorAttention	17.28	25.43	36.88	35.27	**30.11**	12.89	**25.02**	20.17	24.98	65.00	84.29	35.28	3.81	11.00	51.89	49.29	33.04	2.44
Ours-group-stage	16.02	23.90	35.19	36.66	30.50	**14.19**	19.71	19.69	23.78	62.00	86.04	35.91	3.55	8.50	49.37	46.73	31.98	**1.32**
Ours-group	**18.27**	24.33	**37.12**	36.46	29.83	13.57	21.15	**20.84**	24.07	**67.50**	**87.96**	36.42	3.55	**12.00**	51.50	50.72	**33.46**	1.95
Llama3-8B-Instruct, KV Size = FULL																		
FullKV	23.31	31.18	38.09	43.67	35.26	21.43	28.42	22.90	26.64	73.50	89.76	42.20	4.78	67.88	60.12	56.76	41.62	2.98
Llama-3-8B-Instruct, KV Size = 384 , Compressibility is 4.74% (Except CHAI method)																		
H2O	18.80	13.76	21.20	38.90	31.38	14.81	20.38	20.70	22.03	61.00	82.07	39.49	5.12	66.92	58.59	54.98	35.63	3.98
StreamingLLM	19.39	10.44	21.98	39.39	30.03	14.29	20.37	20.82	22.62	57.50	82.89	39.73	5.25	68.00	58.68	55.67	35.44	2.78
SnapKV	21.47	19.77	33.97	43.10	32.79	**21.48**	21.69	22.01	22.92	63.00	89.69	39.78	5.06	67.83	60.19	56.82	38.85	2.68
PyramidKV	22.08	19.43	32.99	42.51	32.01	19.62	21.73	22.24	22.74	71.00	89.59	40.51	4.23	**68.50**	58.92	53.92	38.88	2.78
CHAI	18.99	23.44	31.82	33.37	22.63	19.07	24.46	21.74	23.78	69.00	89.28	37.15	4.92	67.75	44.44	36.12	35.50	4.20
Quest	21.35	23.40	34.56	32.94	31.34	14.93	19.68	22.18	26.07	71.00	88.30	**41.32**	5.37	67.06	54.26	46.25	37.50	2.84
Double-Sparse	21.92	**29.86**	33.09	27.85	29.49	11.11	**27.89**	21.41	**26.66**	71.00	88.52	40.80	**5.68**	65.55	57.18	54.77	38.30	2.98
RazorAttention	**22.36**	26.85	34.28	42.87	34.28	20.85	24.28	21.28	24.98	69.00	89.28	40.28	4.28	67.00	58.28	52.87	39.56	3.02
Ours-group-stage	21.32	26.42	33.98	**43.18**	**35.38**	19.76	22.47	22.13	22.89	**72.00**	90.26	39.29	4.81	60.50	**61.97**	57.61	39.62	**1.79**
Ours-group	22.27	26.57	**36.04**	**43.18**	35.25	20.57	22.71	**22.31**	22.80	**72.00**	**90.71**	40.59	5.21	68.00	61.85	**58.31**	**40.52**	2.70
Qwen2.5-7B-Instruct, KV Size = 8k																		
FullKV	23.64	40.50	50.03	43.44	46.28	26.67	32.94	22.50	25.32	69.00	88.22	39.87	3.72	63.00	6.70	3.93	36.61	2.45
Qwen2.5-7B-Instruct, KV Size = 384 , Compressibility is 9.38% (Except CHAI method)																		
H2O	20.62	23.90	27.84	36.66	34.71	19.84	25.49	19.64	21.67	39.50	82.55	**37.89**	5.25	16.50	8.33	**10.96**	26.96	3.14
StreamingLLM	17.20	21.26	32.74	37.54	36.97	18.64	25.14	18.81	23.54	45.00	73.39	32.10	**6.58**	16.50	**9.10**	7.29	26.36	2.64
SnapKV	24.16	36.32	47.64	42.64	42.60	23.42	25.64	20.87	21.64	65.00	87.45	37.65	5.37	69.50	5.78	6.13	35.11	2.63
PyramidKV	21.72	34.32	45.25	**44.43**	41.43	**24.93**	23.72	20.51	20.18	59.50	87.09	36.49	4.68	69.00	5.68	5.84	34.05	2.61
CHAI	22.98	38.42	42.68	43.69	39.65	22.58	24.58	20.64	21.87	**66.00**	86.98	36.97	4.89	69.00	3.64	5.64	34.39	3.48
Quest	24.36	36.84	48.25	42.65	40.80	24.68	23.54	20.23	22.58	65.00	86.54	37.61	4.89	67.00	8.12	8.26	35.08	2.89
Double-Sparse	23.56	37.24	**48.64**	43.54	41.26	24.54	**30.29**	20.41	**23.43**	**66.00**	87.65	36.54	4.36	68.00	6.34	5.64	35.47	2.76
RazorAttention	21.45	35.68	47.29	42.76	39.86	24.86	24.98	20.18	21.28	66.00	88.24	37.51	5.89	70.00	6.12	8.24	35.02	2.88
Ours-group-stage	**25.13**	40.22	48.49	44.21	40.67	24.75	26.05	**21.37**	22.04	65.00	88.47	37.76	5.05	**70.50**	8.44	6.37	**35.91**	**1.64**
Ours-group	23.91	**40.31**	48.41	43.97	**41.89**	23.81	25.89	21.10	21.91	**66.00**	**89.26**	37.59	4.69	70.00	8.38	6.20	35.83	2.71

Table 1: Performance Comparison across Different Tasks: Ours-group-stage compresses both hidden states and KV cache, while Ours-group compresses only the KV cache. Ours-group-stage is 68.42% faster than FullKV, with only a 1.43% performance loss. All methods compress key and value caches at the same compression ratio.

Dataset	High Entropy	Random Group	Low Entropy
Qasper	15.40	11.70	10.60
Musique	6.96	4.47	4.50
GovReport	14.88	7.31	3.20
TREC	55.50	32.50	30.45
PCount	5.00	3.97	3.90
Lcc	50.00	32.50	30.55
Average	**24.62**	15.49	13.86

Table 2: Head Sparsity Patterns.

token budget for each head, we define a step size Δn_g, progressively reducing the KV cache context size from $N_{i,1}$ which corresponds to the largest *truncated effective rank*, down to $N_{i,M}$ as follows:

$$N_{i,g+1} = N_{i,g} - \Delta n_g, \quad g = 1, 2, \ldots, M. \tag{11}$$

where $N_{i,g}$ represents the context size for the g-th head group at layer i, and Δn_g is the fixed step size between consecutive groups. During decoding, each head maintains its context window based on the group's token budget $N_{i,g}$.

Dynamic KV Cache Eviction We maintain a fixed cumulative attention score window (Li et al., 2024), $w_{i,h}$, with a budget size of $N_{i,g}$, where h refers to the head. The window $w_{i,h}$ is computed by summing the attention scores of the last l tokens from the h-th head in the i-th layer, retaining the top $N_{i,g}$ tokens with the highest scores. As new tokens are generated, the token with the lowest cumulative attention score in $w_{i,h}$ is evicted.

Retrieval Head When $M = 2$, we can divide the heads into two groups: retrieval heads and streaming heads. Compared to previous methods (Xiao et al., 2024; Tang et al., 2024a) that required spe-

cially designed datasets and complex searches, we reduce the retrieval head identification from 1.2 hour to 1.6 minute, achieving a 45× speedup in retrieval head identification.

4 Experiment

4.1 Experimental Settings

We evaluate three LLMs: Llama2-7B/13B-chat-hf (Touvron et al., 2023), Llama-3-8B-Inst (AI, 2024), and Qwen2.5-7B-Instruct (Yang et al., 2024b). UNCOMP is compared to KV cache eviction techniques, including H2O (Zhang et al., 2024d), PyramidKV (Zhang et al., 2024c), SnapKV (Li et al., 2024), Double-Sparse (Yang et al., 2024c), Quest (Tang et al., 2024b), StreamLLM (Xiao et al., 2023), RazorAttention (Tang et al., 2024a) and CHAI (Agarwal et al., 2024). UNCOMP is assessed on LongBench (Bai et al., 2023) and 'Needle in a Haystack' task (Liu et al., 2024c). Results of InfiniteBench (Zhang et al., 2024b) and RULER (Hsieh et al., 2024) are provided in Appendix E.

4.2 Memory-Constrained Setting

Main Results We divide the heads in the KV cache into 8 groups, with $N_{i,1} = 640$ and $\Delta n_g = 74$. For fairness, the KV cache size of all heads in other models is set to 384. From Table 1, we draw the following conclusions: *i)* UNCOMP achieves the best performance, especially on LLaMA3, as we reveal in Section 3.3.3 that grouping settings with a compression ratio performs better due to their training with GQA. This leads to certain heads sharing the same sparsity patterns, resulting in a speedup up to 60% per instance with a 4.68% compression ratio. *ii)* UNComp exhibits near-lossless performance in certain models, especially compared to the full-size KV cache setting in Llama2-7B/13B-chat-hf, with only a 0.74% performance loss down to a 9.38% compression ratio. *iii)* The method most similar to our head grouping approach is **CHAI**. With a KV cache compression ratio lower than CHAI's 68.55%, our method achieved 5.4× faster inference speed.

Head Pruning To further explore the relationship between the information compression pattern and the sparsity pattern, we compare performance under extreme compression settings. For this investigation, we divide the heads into two groups. As shown in Table 3, when the compression ratio of the KV cache is set to 1.56%, our method shows a substantial enhancement over existing methods.

Llama2-7B-chat-hf, KV Size = 64							
Methods	**Qasper**	**Musique**	**GovReport**	**TREC**	**PCount**	**Lcc**	**Average**
FullKV	18.61	10.05	25.19	63.00	5.00	61.40	30.54
H2O	13.84	1.33	8.57	18.00	0.50	28.86	11.85
StreamingLLM	14.26	0.79	8.37	18.50	4.00	29.81	12.62
SnapKV	15.70	6.15	11.16	40.50	5.00	43.77	20.38
PyramidKV	16.10	6.58	12.07	46.00	**5.50**	46.09	22.06
Quest	16.50	6.01	10.42	53.50	5.00	46.09	22.92
Double-Sparse	16.30	5.92	**16.23**	51.00	5.00	42.00	22.74
Ours-group-stage	**17.57**	6.58	15.25	**59.00**	4.50	49.75	**25.44**
Ours-group	15.40	**6.96**	14.88	55.50	5.00	**50.00**	24.62
Llama2-7B-chat-hf, KV Cache Size of One Group With Extreme Compression							
Methods	**Qasper**	**Musique**	**GovReport**	**TREC**	**PCount**	**Lcc**	**Average**
FullKV	18.61	10.05	25.19	63.00	5.00	61.40	30.54
Remain-256	19.67	9.80	20.19	63.00	5.50	60.28	29.74
Remain-128	18.67	9.75	20.02	63.00	5.50	59.60	29.42
Remain-64	18.13	9.79	19.84	63.00	5.50	58.09	29.06
Remain-32	18.04	9.24	19.31	63.00	5.50	57.04	28.69
Remain-16	18.48	8.08	18.21	63.00	5.00	47.29	26.68
Remain-12	17.31	8.78	18.16	62.00	5.00	45.23	26.08
Delete-2-heads	18.30	8.58	18.98	63.00	5.50	59.38	28.96
Delete-4-heads	13.29	7.96	19.12	62.50	5.50	53.94	27.05
Delete-8-heads	12.64	6.60	9.87	63.50	3.21	37.87	22.28
CHAI Delete-2-heads	18.99	10.20	23.65	69.00	4.00	55.28	30.18
CHAI Delete-4-heads	16.75	9.20	20.79	57.00	4.00	50.10	26.30
CHAI Delete-8-heads	-	-	-	-	-	-	-
RA Delete-2-heads	18.27	10.84	24.87	64.00	4.00	59.87	30.31
RA Delete-4-heads	13.28	7.28	22.89	63.00	4.00	56.91	27.89
RA Delete-8-heads	8.91	4.29	7.19	54.00	3.00	32.98	18.40

Table 3: Extreme Compression Ratio.

We further explore the minimum achievable compression ratio for the group with the lower effective rank, as detailed in Table 3 where *Ours-remain-tokens-N* indicates the retention of N tokens per attention head, while the other group maintains a KV cache size of 512. Furthermore, *Delete-K-head* denotes the complete pruning of K heads per layer, contingent on the effective rank order. *The results show that our method maintains high accuracy with only 12 tokens retained in streaming heads or after pruning certain heads.* We primarily compare with CHAI, as it also uses head grouping for pruning and maintains strong performance after removing heads. The performance of CHAI crashes when the cache size is only 64 or when 8 heads are removed, so its performance is not provided. This finding further supports the validity of our way of identifying special sparse pattern, such as retrieval heads and streaming heads. We also compared our method with the RazorAttention (RA) approach, which prunes heads based on retrieval heads identification. Although RA shows some advantages under extreme compression rates, its performance still falls short of ours even when 8 heads are pruned.

Inference Time Latency and Performance We analyze the inference time latency and the specific time costs of each component. To achieve reliable time analysis, we synchronize the CPU and GPU clock frequencies to facilitate our measurements. We sample 150 data points from the MultifieldQA and use the Llama2 model to measure the overhead in a single batch on a NVIDIA A100 80G GPU. We analyze the duration of the prefill stage, the decoding duration, the total duration of the stage

Methods	NVIDIA A100 80G GPU			
	Inference	Prefill	Decoding	Max Memory
FullKV	129.13	77.34	51.79	25900
StreamingLLM	170.42	90.28	80.14	22908
H2O	140.93	90.56	50.37	22936
PyramidKV	126.03	78.98	47.05	22938
SnapKV	123.71	78.71	45.00	22920
Quest	170.59	110.33	60.26	22940
Ours-group	137.84	79.25	58.59	22900
Ours-group-stage	105.47	48.18	57.29	22988

Table 4: Time Consumption (s) and Memory Usage (MB) Analysis on an GPU. The open-source code of double-sparse does not implement a true token eviction strategy, so it is not compared here. The initial KV cache size is 384 per layer, with a batch size of 1.

inference, and the maximum memory usage.

In Table 4, we compare our experimental results, where the prompt and generated token lengths are 3712 and 384, and present the following observations: *i)* The prefill stage takes up more time throughout the inference process. *ii)* UNComp primarily accelerates the prefill stage, achieving up to 60.52% acceleration over the full-size KV cache in a single batch, mainly benefiting from the compression of H_m in the first stage. *iii)* For throughput analysis, experiments with a prompt length of 2048 and generation lengths of 8096 for *ultra-long generation* show that FullKV supports a maximum batch size of 6 with a token generation time of 15.67ms. In comparison, UNComp supports a batch size of 32 with a token generation time of 2.45ms, delivering 6.4× the throughput of FullKV. Details of other prompt lengths and generation text lengths can be found in Appendix C.

Methods	Llama2-4k	Llama3-8k
FullKV	98.70	84.99
H2O	61.14	51.56
StreamingLLM	50.14	42.36
PyramidKV	93.24	79.08
SnapKV	94.50	81.27
Quest	95.50	81.02
Double-Sparse	96.30	82.12
CHAI	97.80	84.20
Ours-group	98.42	84.13
Ours-group-stage	**98.80**	83.73
Ours-group-retrieval-layer	98.56	**85.02**

Table 5: Needle-in-a-haystack results. The interpolation between the retrieval layer and the model's final layer is denoted as Ours-group-retrieval-layer.

Needle in a Haystack Task *The results show that UNComp outperforms FullKV at a 9.38% compression rate.* Table 5 compares Llama2-4k and Llama3-8k, both with a max KV cache size of 384. This indicates that our uncertainty measurement method can identify the special retrieval layers and effectively retrieve key information.

The Ratio of Recent Tokens to Historical Tokens *A peculiar phenomenon in KV cache compression: selecting an appropriate proportion of recent tokens can maintain the inter-layer compression trend, thereby enabling optimal performance.* We find that the model's performance is sensitive to the number of recent tokens (the hyperparameter l), for calculating cumulative attention scores. Our analysis shows that this is due to the special information compression ratio between recent and historical tokens. As shown in Figure 6(a), different proportions of historical and recent tokens exhibit distinct trends across layers. The matrix entropy trend of a compressed KV cache, when more similar to the full KV cache across layers, indicates better performance under the same compression ratio. This is validated through experiments using the *pearson correlation coefficient* to quantify the correlation between the trends of different information compression patterns (i.e., changing the proportion of historical tokens and recent tokens across all layers under a fixed token budget). As depicted in Figure 6 (b), when the inter-layer trend of the compressed KV cache exhibits a higher similarity to the trend of the full-size KV cache, we can achieve optimal performance.

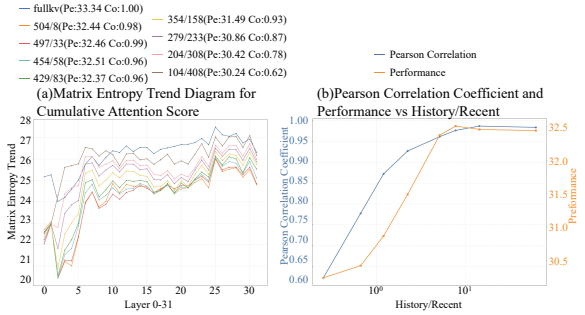

Figure 6: Comparison of inter-layer matrix entropy trends for different H/R, where H/R represents the ratio of the length of historical tokens (H) to the length of recent tokens (R), under the fixed cache size setting.

5 Conclusion

We propose UNCOMP, an uncertainty-aware method for compressing hidden states and KV cache in LLMs. Using *truncated matrix entropy* to measure uncertainty across layers and heads, UNCOMP adaptively adjusts compression ratios, balancing memory efficiency, computational efficiency, and performance. Experiments show that UNCOMP achieves up to 60% speedup in the prefill stage, 6.4× throughput improvement, and compresses the KV cache down to 4.74% of its original size, with only a mild performance loss. UNCOMP outperforms state of the art in many tasks, demonstrating its effectiveness for LLM inference.

Limitations

Our two-stage compression method UNCOMP demonstrates promising results in accelerating inference and reducing memory overhead. The method performs well on tasks such as the needle-in-a-haystack benchmark, further exploration is needed to assess its effectiveness on tasks involving dense, context-dependent dependencies, such as machine translation or dialogue systems.

Potential Risks

While our approach improves the efficiency of long-context inference in LLMs through uncertainty-aware compression, it also introduces new considerations. Dynamically adapting compression based on uncertainty may lead to unintended information loss if uncertainty estimates are miscalibrated, potentially affecting model fidelity in critical tasks. Moreover, the ability to uncover long-range dependencies such as retrieval heads could inadvertently expose internal model mechanisms in ways that may be exploitable. As such, careful validation, transparency in deployment, and alignment with responsible AI practices are essential to ensure these optimizations yield beneficial and trustworthy outcomes without compromising model integrity or user trust.

References

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

Saurabh Agarwal, Bilge Acun, Basil Homer, Mostafa Elhoushi, Yejin Lee, Shivaram Venkataraman, Dimitris Papailiopoulos, and Carole-Jean Wu. 2024. Chai: Clustered head attention for efficient llm inference. *arXiv preprint arXiv:2403.08058*.

Meta AI. 2024. Llama 3: A family of large language models. https://llama.meta.com. Instruction-tuned version.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

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 arXiv:2308.14508*.

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

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, et al. 2023. Language modeling is compression. *arXiv preprint arXiv:2309.10668*.

Ruili Feng, Kecheng Zheng, Yukun Huang, Deli Zhao, Michael Jordan, and Zheng-Jun Zha. 2022. Rank diminishing in deep neural networks. *Advances in Neural Information Processing Systems*, 35:33054–33065.

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

Luis Gonzalo Sanchez Giraldo, Murali Rao, and Jose C Principe. 2014. Measures of entropy from data using infinitely divisible kernels. *IEEE Transactions on Information Theory*, 61(1):535–548.

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

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

Yuzhen Huang, Jinghan Zhang, Zifei Shan, and Junxian He. 2024. Compression represents intelligence linearly. *arXiv preprint arXiv:2404.09937*.

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

Henry F Kaiser. 1960. The application of electronic computers to factor analysis. *Educational and psychological measurement*, 20(1):141–151.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

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

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

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

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

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024c. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

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

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

Stephen Merity. 2016. The wikitext long term dependency language modeling dataset. *Salesforce Metamind*, 9.

Alexander Peysakhovich and Adam Lerer. 2023. Attention sorting combats recency bias in long context language models. *arXiv preprint arXiv:2310.01427*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624.

Olivier Roy and Martin Vetterli. 2007. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*, pages 606–610. IEEE.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

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

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

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

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. 2024. Scaling laws with vocabulary: Larger models deserve larger vocabularies. *arXiv preprint arXiv:2407.13623*.

Robert L Thorndike. 1953. Who belongs in the family? *Psychometrika*, 18(4):267–276.

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 arXiv:2307.09288*.

John Von Neumann. 2013. *Mathematische grundlagen der quantenmechanik*, volume 38. Springer-Verlag.

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 arXiv:2406.13035*.

Lean Wang, Lei Li, Damai Dai, Deli Chen, Hao Zhou, Fandong Meng, Jie Zhou, and Xu Sun. 2023. Label words are anchors: An information flow perspective for understanding in-context learning. *arXiv preprint arXiv:2305.14160*.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*.

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

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

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

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

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. 2024b. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Shuo Yang, Ying Sheng, Joseph E Gonzalez, Ion Stoica, and Lianmin Zheng. 2024c. Post-training sparse attention with double sparsity. *arXiv preprint arXiv:2408.07092*.

Hao Yu, Zelan Yang, Shen Li, Yong Li, and Jianxin Wu. 2024. Effectively compress kv heads for llm. *arXiv preprint arXiv:2406.07056*.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. 2024a. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *arXiv preprint arXiv:2405.03917*.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, et al. 2024b. ∞ bench: Extending long context evaluation beyond 100k tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277.

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

Yifan Zhang, Zhiquan Tan, Jingqin Yang, Weiran Huang, and Yang Yuan. 2023. Matrix information theory for self-supervised learning. *arXiv preprint arXiv:2305.17326*.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient llms inference. In *Forty-first International Conference on Machine Learning*.

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

Zhijian Zhuo, Yifei Wang, Jinwen Ma, and Yisen Wang. 2023. Towards a unified theoretical understanding of non-contrastive learning via rank differential mechanism. *arXiv preprint arXiv:2303.02387*.

Appendix

A Implementation details

A.1 Machine Environment

Main of our experiments are conducted on eight AMD MI210 64G GPUs. Some experiments are conducted on NVIDIA A100 80GB GPUs. Based on our comparison, apart from significant differences in inference speed, the final performance shows slight differences.

A.2 Model Selection

In all of our experiments, the model weights are obtained from Hugging Face. Specifically, for the Llama architectures, we utilize the following versions: Llama2-7B employs 'meta-llama/Llama-2-7b-chat-hf', Llama2-13B utilizes 'meta-llama/Llama-2-13b-chat-hf', and Llama3-8B adopts 'meta-llama/Meta-Llama-3-8B-Instruct'. For the Qwen architecture, we use the 'Qwen/Qwen2.5-7B-Instruct' version.

A.3 Hyperparameter Setting

We conduct the experiment in a scenario with an average KV cache size of 384 per layer. For the baselines, all hyperparameters are taken from the open-source code repository.

For our method, the experiment is governed by five main hyperparameters, the selection of last l token's cumulative attention score, the threshold ϵ, maximum KV cache size of the head in each layer $N^{'}_{i,1}$, fixed step size Δn_g and Δn .

For various models, we configure the parameters as follows: l = 8, $N^{'}_{i,1}$ = 640, Δn_g = 74 and Δn = 512. Threshold ϵ is set to 0.3 for Llama2-7B, 0.5 for Llama2-13B, 0.4 for Llama3-8B and 0.3 for Qwen2.5-7B.

B Details of Evaluation

Longbench is the first benchmark for assessing the long-context understanding capabilities of large language models in a bilingual and multitask framework. It evaluates multilingual capabilities in both Chinese and English, consisting of six major categories and twenty-one tasks. Key application scenarios include single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. We use Longbench to evaluate the performance of our method on contextual input tasks. The details of metrics at Table 6.

Dataset	Metric	Language	Data Length
NarrativeQA	F1	English	200
Qasper	F1	English	200
MultifieldQA	F1	English	150
HotpotQA	F1	English	200
2WikiMQA	F1	English	200
Musique	F1	English	200
GovReport	range-l	English	200
QMSum	range-l	English	200
MultiNews	range-l	English	200
trec	classification accuracy	English	200
TriviaQA	F1	English	200
SAMSum	range-l	English	200
PCount	exact match accuracy	English	200
PRe	exact match accuracy	English	200
Lcc	edit similarity	Python/C#/Java	500
RB-P	edit similarity	Python/Java	500

Table 6: The details of statistics in LongBench

Additionally, once the data sample is encoded into tokens, if its length exceeds the model's maximum input length, we truncate it by taking equal portions from the beginning and the end.

C Throughput Analysis

To thoroughly investigate the inference performance of the model, we conduct experiments on an NVIDIA A100 80G GPU. We randomly sample 96 data points from the Wikitext2 dataset, with strict control over the token lengths for both the prompt and generation phases. Detailed analyses of memory usage and throughput are provided in the Table 7 . We can observe that under the setting of a prompt length of 3712 and a generation length of 384 for this short generated text, our method achieves up to 2.96 times throughput.

D Ablation Study

D.1 Truncation Strategy

Llama-2-7B-chat-hf, KV Cache Size=384					
Top k	Qasper	QMSum	SAMSum	Lcc	Average
Top 16	19.28	20.38	39.45	59.72	34.71
Top 32	19.34	20.51	39.35	59.93	34.78
Top 64	18.75	20.43	39.36	59.86	34.60
Top all	18.14	20.14	38.52	59.51	34.08

Table 8: Truncation Strategy

In this section, we examine truncation strategies, with a focus on evaluating the effectiveness of elbow points. We conduct tests using various elbow points by selecting different top k eigenvalues and compared the results to cases where no elbow points are applied. As demonstrated in Table 8, the results demonstrate a 0.70% performance gap between the truncated and untruncated settings, highlighting the efficacy of our approach.

Llama2-7B-chat-hf, KV Cache Size = 384, Prompt+Generate is 3712+384						
Batch Size	Ours-group-stage		Ours-group		FullKV	
	ms/token	max memory used(MB)	ms/token	max memory used(MB)	ms/token	max memory used(MB)
1	28.055	23492	28.536	23080	25.691	24690
4	7.910	37526	8.504	37516	13.806	44220
8	4.822	59014	5.436	59036	11.823	72444
10	5.340	69802	5.887	69780	-	Out-of-Memory
12	3.994	80514	4.567	80522	-	Out-of-Memory
Llama2-7B-chat-hf, KV Cache Size = 384, Prompt+Generate is 4032+64						
Batch Size	Ours-group-stage		Ours-group		FullKV	
	ms/token	max memory used(MB)	ms/token	max memory used(MB)	ms/token	max memory used(MB)
1	34.782	24298	39.231	24312	36.596	24240
4	13.671	41180	18.458	41170	23.952	41146
8	10.186	66560	15.074	66580	21.944	66532
10	9.907	79198	14.603	79206	21.482	79168
12	9.464	79140	14.150	79174	-	Out-of-Memory

Table 7: Throughput Analysis

Llama2-7B-chat-hf, KV Cache Size=384							
Group Number	KV cache size in different groups	Qasper	HotpotQA	QMSum	SAMSum	Lcc	Average
2 groups	32/736	18.23	30.96	19.82	40.05	57.13	33.24
3 groups	32/384/736	18.90	30.53	19.95	40.04	58.29	33.54
4 groups	32/266/502/736	19.29	30.48	20.10	41.03	59.37	34.05
5 groups	32/208/384/560/736	19.58	31.17	20.72	40.61	58.93	34.20
8 groups	32/132/232/332/436/536/636/736	19.34	31.04	20.16	40.92	59.48	34.19
2 groups	256/512	19.67	30.98	20.20	39.36	60.28	34.10
3 groups	256/384/512	19.45	31.29	20.24	39.63	59.63	34.05
4 groups	256/342/427/512	19.20	30.99	20.10	39.33	59.71	33.87
5 groups	256/320/384/448/512	19.71	30.95	20.22	39.60	59.99	34.09
8 groups	256/296/332/368/404/440/476/512	19.55	31.02	20.59	39.10	59.39	33.93

Table 9: Group Number Analysis

D.2 Number of Head Groups

In this section, we analyze the impact of the number of groups on performance. As illustrated in Table 9, when the KV cache size between groups changes minimally, the maximum performance difference with changes in the number of groups is only 0.23%. However, when there is a significant disparity between the maximum and minimum KV cache sizes of the groups, increasing the number of groups tends to enhance performance, with a performance improvement of 0.96%. It indicates that the number of groups is highly correlated with the distribution of KV cache sizes within groups, and the greater the disparity in sparse patterns, the better the performance.

D.3 Matrix Entropy and Attention Variance

In this section we discuss the grouping policy. We provide the compression ratio estimates based on the variance of attention scores in Table 10, evaluated under two KV cache sizes, 384 and 64. The results clearly highlight our advantages, especially under the budget of 64. This suggests that solely relying on compression ratio estimation based on attention is unreasonable, as attention itself is subject to biases such as the attention sink(Xiao et al., 2023) and recency bias(Peysakhovich and Lerer, 2023). It is necessary to introduce additional unbiased compression estimation methods.

E Supplementary Dataset Comparison

E.1 RULER

RULER (Hsieh et al., 2024) is a novel synthetic benchmark designed to comprehensively evaluate the capabilities of long-context language models (LMs). Unlike the traditional Needle-in-a-Haystack (NIAH) test, which focuses solely on retrieval tasks, RULER provides flexible configurations to support customized sequence lengths and task complexities. It extends the vanilla NIAH test by introducing diverse variations in the types and quantities of “needles" and adding new task categories, such as multi-hop tracing and aggregation, to assess capabilities beyond simple context search. The results are shown in Table 11, where the Llama-3-8B-Instruct model is used, and other settings are consistent with those in the previous section A.3. The experiments are conducted on a single A100 80G GPU. Our method demon-

Methods	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Variance KV (384)	16.75	18.15	32.09	**32.42**	27.29	8.50	19.46	20.42	22.94	62.50	**84.65**	38.64	**5.50**	**12.00**	58.59	52.98	32.06
Uncomp (384)	**17.33**	**19.34**	**34.16**	31.54	**28.23**	**10.04**	**20.38**	**20.51**	**23.33**	**63.00**	84.11	**39.35**	**5.50**	9.50	**59.93**	**54.87**	**32.57**
Variance KV (64)	8.75	13.58	12.24	20.27	13.38	3.89	8.76	15.73	13.98	29.50	56.22	30.35	**5.00**	5.45	37.10	30.47	19.04
Uncomp (64)	**14.05**	**15.40**	**25.56**	**26.28**	**21.96**	**6.96**	**14.88**	**18.83**	**17.58**	**55.50**	**81.61**	**34.74**	**5.00**	**5.00**	**50.00**	**45.55**	**27.43**

Table 10: Comparison of entropy and variance of truncated matrices

RULER(8k)	niah single 1	niah single 2	niah single 3	niah multikey 1	niah multikey 2	niah multikey 3	niah multivalue	niah multiquery	vt	cwe	fwe	qa 1	qa 2	average
FullKV 8k	100.00	100.00	100.00	98.80	88.20	97.60	95.40	99.40	98.60	97.74	83.93	67.40	50.80	90.61
UNComp	**100.00**	**99.80**	3.80	**99.40**	**72.80**	0.00	**81.55**	**74.75**	93.88	20.78	53.93	64.40	49.60	**62.67**
SnapKV	**100.00**	**99.80**	1.60	98.80	72.60	0.00	78.00	71.05	**94.36**	21.16	49.60	**64.80**	**50.00**	61.67
PyramidKV	**100.00**	98.40	0.00	98.40	66.00	0.00	63.60	42.55	81.96	8.16	41.00	65.00	48.60	54.90
CHAI	35.00	22.80	**23.40**	22.00	3.80	0.60	23.40	23.80	11.24	0.66	7.00	25.80	21.80	17.02
H2O	2.80	3.80	5.80	5.40	4.00	**3.00**	4.60	5.20	4.60	**34.60**	**85.87**	42.00	39.60	18.56
RULER(4k)	**niah single 1**	**niah single 2**	**niah single 3**	**niah multikey 1**	**niah multikey 2**	**niah multikey 3**	**niah multivalue**	**niah multiquery**	**vt**	**cwe**	**fwe**	**qa 1**	**qa 2**	**average**
FullKV 4k	100.00	100.00	100.00	99.40	100.00	98.80	99.15	99.85	99.72	99.80	94.20	81.40	58.00	94.64
UNComp	**100.00**	**99.80**	18.80	95.60	**98.80**	0.00	**93.00**	**93.00**	**95.84**	**56.06**	**78.07**	**81.40**	**57.20**	**74.43**
SnapKV	**100.00**	99.60	8.00	**99.40**	97.40	0.00	88.30	87.70	95.80	52.86	76.33	**81.40**	56.60	72.57
PyramidKV	**100.00**	99.40	0.60	98.60	91.80	0.00	65.85	49.40	78.84	10.50	66.20	81.00	55.40	61.35
CHAI	44.40	54.00	**46.60**	36.60	14.00	**7.20**	53.40	52.60	17.16	13.00	25.60	59.40	30.20	34.94
H2O	10.40	12.60	13.00	14.60	9.20	7.00	12.25	13.15	8.64	82.94	93.00	81.80	40.00	30.66

Table 11: Performance comparison of methods on RULER benchmark across different context lengths. The first section shows results for an 8k context, while the second section highlights 4k context performance.

strates superior performance, while PyramidKV's relatively poor performance suggests that setting a separate compression rate for each layer may hinder the effective context length of the model. Therefore, an appropriate grouping strategy for the layers is essential.

E.2 InfiniteBench

InfiniteBench(Zhang et al., 2024b) is a state-of-the-art benchmark designed to evaluate language models' ability to process, understand, and reason over extremely long contexts exceeding 100k tokens. By pushing context lengths 10 times beyond traditional datasets, InfiniteBench aims to advance applications of LLMs and enable high-level interactions in scenarios requiring extensive context comprehension. Results are showed at Table 12, where the Llama-3-8B-Instruct model is used, and other settings are consistent with the previous section A.3. Our method demonstrates exceptional robustness and is the only approach that surpasses the performance of FullKV.

E.3 Evaluating Generalization on Reasoning Task

To verify the generalization of our method, we conducted a comparison of experiments on the GSM8K (Cobbe et al., 2021) dataset, and the results are shown in the table 13. The experiment demonstrates the superiority of our method in few-shot reasoning performance. We also found that our method significantly outperforms other methods in a zero-shot setting, which suggests that our approach may be able to identify a sparse pattern in the absence of samples, thereby avoiding the loss of reasoning capability.

Table 13: Based on the input question, half of the tokens are kept at a time, while for few-shot prompts, the 384 KV cache size is consistently maintained. We compare the experimental results under 6-shot and 12-shot conditions. The experiments are performed using Llama2-7B-chat-hf.

Method(Number of tokens)	Zero-shot(112))	6-shot(1251)	12-shot(2321)
FullKV	24.63	27.14	26.91
StreamingLLM	5.68	26.46	24.68
H2O	5.31	27.82	27.14
CHAI	5.69	3.87	4.06
SnapKV	5.53	26.61	24.48
PyramidKV	2.35	24.49	24.68
Ours-group	12.05	27.89	27.68
Ours-group-stage	12.86	26.23	26.84

F Analysis about Truncated Effective Rank of Hidden States

Figure 7 shows the entropy trend of sample of the Wikitext2. It can be seen that the matrix entropy of hidden states increases layer by layer, which indicates that the information compression pattern is completely consistent with V_m. This is an interesting phenomenon because the Keys and queries in the KV cache share the same pattern, while the values share the same pattern as the hidden states. We believe this may be due to the positional encodings

Method	En.Sum	En.QA	En.MC	En.Dia	Zh.QA	Code.Debug	Code.Run	Math.Calc	Math.Find	Retrieve.PassKey	Retrieve.Number	Retrieve.KV	Average
FullKV	12.55	0.27	42.79	1.00	4.04	22.34	0.00	0.00	38.57	6.27	6.44	4.80	14.38
uncomp	**11.74**	0.23	**44.98**	3.80	3.00	21.57	0.00	0.00	**38.57**	**6.27**	6.44	0.00	**14.77**
snapkv	11.59	0.28	42.36	1.00	4.01	21.83	0.00	0.00	38.29	**6.27**	6.61	0.00	14.22
pyramidkv	11.34	0.23	40.61	2.50	**4.03**	22.08	0.00	0.00	**38.57**	**6.27**	**6.78**	0.00	14.26
chai	9.69	**0.37**	34.06	**8.00**	3.26	**24.97**	0.00	0.00	27.43	4.58	5.93	1.20	12.79
h2o	10.99	0.18	**44.98**	3.50	3.98	22.08	0.00	0.00	37.71	1.69	1.69	0.00	14.24

Table 12: Performance comparison of various methods on InfiniteBench across different tasks, including summarization, QA, mathematical reasoning, and code-related benchmarks. The “Average" column represents the overall average performance.

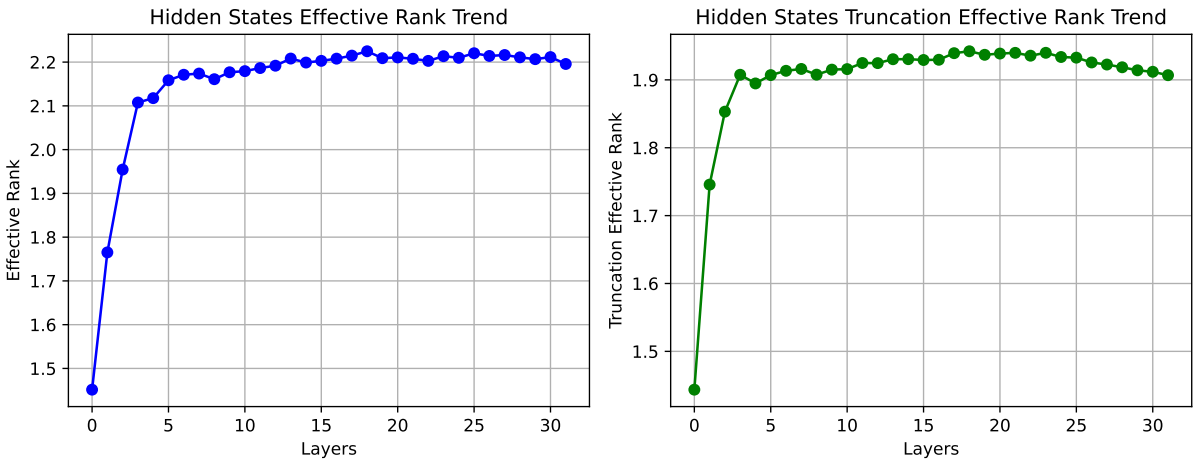

Figure 7: Effective rank and truncated effective rank of hidden states across different layers.

assigned to both the Queries and Keys. This also reveals a widespread connection between different types of parameters in the model, suggesting that predicting the sparse pattern of one set of parameters using another set is feasible.

G Appendix for Proofs

The effective rank of $\mathbf{\Sigma_X}$, denoted $\text{erank}(\mathbf{\Sigma_X})$, is defined as (Roy and Vetterli, 2007):

$$\text{erank}(\mathbf{\Sigma_X}) = \exp(H(\mathbf{\Sigma_X})). \tag{12}$$

Lemma 3 *The rank of the covariance matrix $\mathbf{\Sigma_X}$ is upper bounded by the rank of the input matrix $\mathbf{X}$:*

$$\textit{rank}(\mathbf{\Sigma_X}) \leq \textit{rank}(\mathbf{X}). \tag{13}$$

Lemma 4 *Eq. (12) can be interpreted as the dimension of the affine subspace spanned, i.e., the effective dimensionality of the token matrix in the head and layer dimensions. The bounds are:*

$$1 \leq \textit{erank}(\mathbf{\Sigma_X}) \leq \textit{rank}(\mathbf{\Sigma_X}) \leq D. \tag{14}$$

We use $\text{erank}(\mathbf{\Sigma_X})$ to quantify the information of token matrix representations, providing a unified uncertainty measure for both the KV cache and hidden states.

Proof Lemma 1

Proof. To derive the von Neumann entropy from the Rényi entropy, we first need to clarify the relationship between the two. The von Neumann entropy can be seen as a special case of the Rényi entropy in the limit where the Rényi parameter $\alpha \to 1$. The Rényi entropy is defined as:

$$S_\alpha(\mathbf{\Sigma_X}) = \frac{1}{1-\alpha} \log \left(\text{Tr}((\mathbf{\Sigma_X})^\alpha) \right), \tag{15}$$

where α is the order of the Rényi entropy, $\mathbf{\Sigma_X}$ is the density matrix, and $\text{Tr}(\rho^\alpha)$ is the trace of the density matrix raised to the power of α. To derive the von Neumann entropy, we need to examine the limit of the Rényi entropy as $\alpha \to 1$. Let's consider the form of the Rényi entropy:

$$S_\alpha(\mathbf{\Sigma_X}) = \frac{1}{1-\alpha} \log \left(\sum_i \sigma_i^\alpha \right), \tag{16}$$

where σ_i are the eigenvalues of the density matrix $\mathbf{\Sigma_X}$. As $\alpha \to 1$, we can apply L'Hôpital's rule to compute this limit:

$$S(\mathbf{\Sigma_X}) = \lim_{\alpha \to 1} S_\alpha(\rho) = \lim_{\alpha \to 1} \frac{1}{1-\alpha} \log \left(\sum_i \sigma_i^\alpha \right) \tag{17}$$

To proceed, consider the Taylor expansion of $\sum_i \sigma_i^\alpha$:

$$\begin{aligned}\sum_i \sigma_i^\alpha &= \sum_i \sigma_i \cdot e^{(\alpha-1)\log\sigma_i} \\ &\approx \sum_i \sigma_i \left(1 + (\alpha-1)\log\sigma_i\right) \\ &= 1 + (\alpha-1)\sum_i \sigma_i \log\sigma_i\end{aligned} \tag{18}$$

Thus,

$$S_\alpha(\mathbf{\Sigma_X}) \approx \frac{1}{1-\alpha}\log\left(1 + (\alpha-1)\sum_i \sigma_i \log\sigma_i\right) \tag{19}$$

As $\alpha \to 1$, we can use the approximation $\log(1+x) \approx x$ for small x. Therefore, we get:

$$S_\alpha(\mathbf{\Sigma_X}) \approx -\sum_i \sigma_i \log\sigma_i \tag{20}$$

which is exactly the expression for the von Neumann entropy:

$$H(\mathbf{\Sigma_X}) = -\text{Tr}(\mathbf{\Sigma_X}\log(\mathbf{\Sigma_X})) \tag{21}$$

Proof Lemma 2

Proof. In this section, we present a continuous proof of the transformation from the matrix entropy formula to the eigenvalue form.

$$H(\mathbf{\Sigma_X}) = -\text{Tr}\left(\mathbf{\Sigma_X}\log\left(\mathbf{\Sigma_X}\right)\right) \tag{22}$$

Given that $\mathbf{\Sigma_X}$ is a symmetric positive definite matrix, we can perform an eigenvalue decomposition:

$$\mathbf{\Sigma_X} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \tag{23}$$

where $\mathbf{U}$ is an orthogonal matrix composed of eigenvectors, and $\mathbf{\Lambda}$ is a diagonal matrix whose entries are the eigenvalues $\sigma_1, \sigma_2, \ldots, \sigma_D$. The logarithm of $\mathbf{\Sigma_X}$ can then be written as:

$$\log(\mathbf{\Sigma_X}) = \mathbf{U}\log(\mathbf{\Lambda})\mathbf{U}^\top \tag{24}$$

where $\log(\mathbf{\Lambda})$ is a diagonal matrix whose elements are $\log(\sigma_1), \log(\sigma_2), \ldots, \log(\sigma_D)$. Substituting these into the entropy expression:

$$H(\mathbf{\Sigma_X}) = -\text{Tr}\left(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top\mathbf{U}\log(\mathbf{\Lambda})\mathbf{U}^\top\right) \tag{25}$$

Since $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$, this simplifies to:

$$H(\mathbf{\Sigma_X}) = -\text{Tr}\left(\mathbf{\Lambda}\log(\mathbf{\Lambda})\right) \tag{26}$$

For a diagonal matrix, the trace is the sum of its diagonal elements. Therefore, we have:

$$H(\mathbf{\Sigma_X}) = -\sum_{i=1}^{D} \sigma_i \log(\sigma_i) \tag{27}$$

This concludes the proof that the matrix entropy formula can be written as the sum of the eigenvalues of $\mathbf{\Sigma_X}$.

Proof Lemma 3

Proof. Let $\mathbf{X} \in \mathbb{R}^{n\times p}$ be a matrix representing n observations and p variables. The covariance matrix $\mathbf{\Sigma_X}$ of $\mathbf{X}$ is defined as:

$$\mathbf{\Sigma_X} = \frac{1}{n-1}\mathbf{X}^\top\mathbf{X} \tag{28}$$

The goal is to determine the relationship between the rank of the matrix $\mathbf{X}$ and the rank of its covariance matrix $\mathbf{\Sigma_X}$.

The rank of the matrix $\mathbf{X}$, denoted as $\text{rank}(\mathbf{X})$, is the number of linearly independent columns in $\mathbf{X}$, and it satisfies the inequality:

$$\text{rank}(\mathbf{X}) \leq \min(n, p) \tag{29}$$

Since the covariance matrix $\mathbf{\Sigma_X}$ is given by $\mathbf{\Sigma_X} = \frac{1}{n-1}\mathbf{X}^\top\mathbf{X}$, it is a $p \times p$ symmetric matrix. The rank of $\mathbf{\Sigma_X}$, denoted $\text{rank}(\mathbf{\Sigma_X})$, is determined by the product $\mathbf{X}^\top\mathbf{X}$. The rank of this product is bounded by the rank of $\mathbf{X}$, so we have the following inequality:

$$\text{rank}(\mathbf{\Sigma_X}) \leq \text{rank}(\mathbf{X}) \tag{30}$$

This shows that the rank of the covariance matrix $\mathbf{\Sigma_X}$ cannot exceed the rank of the original matrix $\mathbf{X}$. In the case where the number of observations n is greater than or equal to the number of variables p (i.e., $n \geq p$), and the columns of $\mathbf{X}$ are linearly independent, the rank of $\mathbf{X}$ is equal to p, meaning $\text{rank}(\mathbf{X}) = p$. In this scenario, the matrix $\mathbf{X}^\top\mathbf{X}$ has full rank, which implies that the covariance matrix $\mathbf{\Sigma_X}$ will also have full rank. Therefore, we have $\text{rank}(\mathbf{\Sigma_X}) = p$, and the rank of the covariance matrix is equal to the rank of the original matrix, i.e., $\text{rank}(\mathbf{\Sigma_X}) = \text{rank}(\mathbf{X})$.

On the other hand, when the number of observations is less than the number of variables (i.e., $n < p$), the rank of $\mathbf{X}$ is constrained by the number of observations, such that $\text{rank}(\mathbf{X}) \leq n$. Consequently, the rank of the covariance matrix $\mathbf{\Sigma_X}$ is

also limited by n, meaning $\text{rank}(\mathbf{\Sigma_X}) \leq n$. Since $n < p$ in this case, the covariance matrix is rank-deficient, and we have $\text{rank}(\mathbf{\Sigma_X}) < p$.

In general, the rank of the covariance matrix $\mathbf{\Sigma_X}$ is less than or equal to the rank of the original matrix $\mathbf{X}$. Specifically, $\text{rank}(\mathbf{\Sigma_X}) = \text{rank}(\mathbf{X})$ when the number of observations $n \geq p$ and the columns of $\mathbf{X}$ are linearly independent. However, when $n < p$, the covariance matrix $\mathbf{\Sigma_X}$ will be rank-deficient, such that $\text{rank}(\mathbf{\Sigma_X}) < p$.

Proof Lemma 4

Proof. The entropy $H(\mathbf{\Sigma_X})$ of a set of singular values $\sigma_1, \sigma_2, \ldots, \sigma_D$ is given by the formula:

$$H(\sigma_1, \sigma_2, \ldots, \sigma_D) = -\sum_{i=1}^{D} \sigma_i \log \sigma_i. \quad (31)$$

The trace of $\mathbf{\Sigma_X}$, $\text{Tr}(\mathbf{\Sigma_X})$, is 1. Since entropy measures the uncertainty or disorder in a distribution, we can establish certain bounds for the entropy based on the structure of the singular values.

First, we note that if the distribution is concentrated entirely at a single value (i.e., all but one of the singular values are zero), then the entropy will be minimized at 0. Specifically:

$$H(1, 0, \ldots, 0) = 0. \quad (32)$$

On the other hand, the entropy is maximized when the singular values are uniformly distributed. In the case of a uniform distribution over D singular values, we have:

$$\sigma_1 = \sigma_2 = \cdots = \sigma_D = \frac{1}{D}, \quad (33)$$

and the entropy in this case is:

$$H\left(\frac{1}{D}, \frac{1}{D}, \ldots, \frac{1}{D}\right) = -D\left(\frac{1}{D} \log \frac{1}{D}\right) = \log D. \quad (34)$$

Thus, we have the inequality:

$$0 = H(1, 0, \ldots, 0) \leq H(\sigma_1, \sigma_2, \ldots, \sigma_D) \leq \log D. \quad (35)$$

The *effective rank* is defined as:

$$\text{erank}(\mathbf{\Sigma_X}) = \exp(H(\sigma_1, \sigma_2, \ldots, \sigma_D)), \quad (36)$$

which quantifies the "effective" number of singular values that are significantly contributing to the rank of the matrix. Since $H(\sigma_1, \sigma_2, \ldots, \sigma_D)$ is bounded by $\log D$, it follows that the effective rank is bounded by:

$$1 \leq \text{erank}(\mathbf{\Sigma_X}) \leq D. \quad (37)$$

Equality holds at the lower bound if and only if $(\sigma_1, \sigma_2, \ldots, \sigma_D) = (1, 0, \ldots, 0)$, that is, when all but one singular value is zero. In this case, the singular value vector is:

$$\sigma = (\|\sigma\|_1, 0, \ldots, 0)^T, \quad (38)$$

where $\|\sigma\|_1 = 1$. Hence, $\text{erank}(\mathbf{\Sigma_X}) = 1$.

Next, suppose that only k singular values of A are non-zero for some $k \in \{1, 2, \ldots, D\}$. In this case, the rank of A is given by $\text{rank}(A) = k$, and the entropy only depends on the non-zero singular values. Thus, we have:

$$H(\sigma_1, \sigma_2, \ldots, \sigma_D) = H(\sigma_1, \sigma_2, \ldots, \sigma_k), \quad (39)$$

where $\sigma_1, \sigma_2, \ldots, \sigma_k$ are the non-zero singular values. Since entropy is maximized when these non-zero singular values are uniformly distributed, we have:

$$H(\sigma_1, \sigma_2, \ldots, \sigma_k) \leq \log k. \quad (40)$$

Hence, the effective rank satisfies:

$$\text{erank}(\mathbf{\Sigma_X}) \leq \text{rank}(\mathbf{\Sigma_X}) \leq D, \quad (41)$$

with equality $\text{erank}(\mathbf{\Sigma_X}) = \text{rank}(\mathbf{\Sigma_X})$ if and only if the non-zero singular values are uniformly distributed, i.e.,

$$(\sigma_1, \ldots, \sigma_k, \sigma_{k+1}, \ldots, \sigma_D) = \left(\frac{1}{k}, \ldots, \frac{1}{k}, 0, \ldots, 0\right), \quad (42)$$

or equivalently:

$$\sigma = (\|\sigma\|_1/k, \ldots, \|\sigma\|_1/k, 0, \ldots, 0)^T. \quad (43)$$

In this case, the effective rank coincides with the actual rank of the matrix, since the singular values contribute equally to the rank.

H A Comprehensive Walk-Through

We present the detailed procedure of the algorithm during the preparation phase in Algorithm 1. This step is designed to identify consistent attention head behaviors across different layers and input samples by leveraging truncated matrix entropy. Specifically, for each sample in a batch of 500, we tokenize the input and perform a full forward pass through the transformer's self-attention layers. At each layer, we calculate the truncated entropy $E_{l,h}$ for every attention head h, which serves as a proxy for information compression.

Algorithm 1 Head Group Identification (Preparation Phase)

for each input sample x_i in the 500-sample batch **do**
    Tokenize x_i
    Forward x_i through all self-attention layers
    for each layer l **do**
        for each head h **do**
            Compute truncated entropy $E_{l,h}$ (Assign h to one of 8 clusters based on $E_{l,h}$)
        end for
    end for
    Save head cluster labels for all layers
end for
for each head h across all layers **do**
    Assign final group label by majority vote
end for

Algorithm 2 Inference Phase with Dynamic KV Cache Compression

Init: Load head group labels and set the size of kv cache per head

Prefill:
for each transformer layer l **do**
    if $l > 0$ and erank$_k(Q_l)$ - erank$_k(Q_{l-1})$ > ϵ **then**
        Compress hidden states
    end if
    for each head h **do**
        Compress the h-th head of kv cache based on group
    end for
end for
Forward MLP

Decoding:
for each new token t **do**
    for each transformer layer l **do**
        for each head h **do**
            Concatenate the new token with old the h-th head of kv cache and compute attention using concatenated the h-th head kv cache
        end for
    end for
    Emit one token and update kv cache
    Forward MLP
end for

Each head is then assigned to one of eight clusters according to its entropy value, providing a coarse-grained categorization of its behavior. These per-sample cluster assignments are aggregated across the dataset, and for each head, a final group label is determined by majority voting. This group label serves as a stable representation of the head's functional role and is used in subsequent stages of our method.

In Algorithm 2, we detail the inference phase that incorporates dynamic KV cache compression based on the entropy-driven head groupings derived during the preparation phase. The process begins with initialization, where the group labels for each attention head are loaded, and the cache size is configured accordingly.

During the **Prefill** stage, for each transformer layer l, the algorithm evaluates the entropy shift between Q_l and Q_{l-1}. If the difference exceeds a threshold ϵ, the hidden states are compressed to minimize redundant information. Subsequently, each head's KV cache is selectively compressed based on its assigned group label, balancing efficiency with retention of critical information. The MLP block is then forwarded as usual.

In the **Decoding** stage, for every new token, each layer performs head-wise attention using a dynamically updated KV cache. The cache is expanded by concatenating the new token with existing head-specific cache entries, enabling efficient autoregressive decoding. After processing through the MLP, a new token is generated, and the KV cache is updated accordingly. This dynamic approach ensures computational and memory efficiency while maintaining model performance.