ICASSP 2026 - 2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 979-8-3315-6701-9/26/$31.00 ©2026 IEEE | DOI: 10.1109/ICASSP55912.2026.11463859

SHRINKV: KEY-VALUE CACHE COMPRESSION WITH PROGRESSIVE HIDDEN STATES SHRINKING TO MITIGATE PREFILLING LATENCY

Jian Yuan[1]*, Ziwei He*[2]*, Zhouhan Lin*[1]*, Bo Jiang*[1*]

[1]Shanghai Jiao Tong University [2]Shanghai Innovation Institute
{yuanjian, bjiang}@sjtu.edu.cn ziweihe@outlook.com lin.zhouhan@gmail.com

ABSTRACT

The autoregressive attention mechanism in large language models (LLMs) enables the avoidance of redundant computations by storing Key-Value (KV) caches. Existing KV cache compression methods primarily target either the prefilling or decoding stage. However, most accelerate only the decoding phase, rendering the prefilling stage time-consuming for long prompts. Given that shallower layers demonstrate reduced tolerance to compression, we propose ShrinKV, a training-free KV cache compression strategy that reduces latency across both stages by progressively shortening hidden states at regular intervals and compressing KV states derived from these reduced representations. Experiments on needle-in-a-haystack tasks and various long context understanding tasks demonstrate that ShrinKV outperforms state-of-the-art methods, while speeding up the prefilling stage by over $2.4\times$ compared to the full caches.

***Index Terms*—** Large Language Models, KV Cache Compression, Hidden States Compression, Prefilling Acceleration

1. INTRODUCTION

Currently large language models (LLMs) are generally built on decoder-only transformers whose autoregressive attention mechanism relies on frequent access to key-value (KV) pairs from the prior context. KV caching enhances generation efficiency by storing these historical data to preclude redundant computations. Nevertheless, as input or generated sequence lengths grow, the storage footprint of KV caches expands proportionally, necessitating compression to mitigate memory overhead and avert bottlenecks [1, 2]. By employing KV cache compression strategies, LLMs can efficiently handle long-sequence tasks while preserving performance, thereby extending their applicability to diverse practical scenarios.

The inference of LLMs comprises prefilling and decoding. Prefilling latency directly impacts the response speed from user input to the generation of the first output token which is a core metric for user perception of interaction fluency [3, 4]. Most KV cache compression methods either target the decoding stage [5, 6] or compress the KV cache of long prompts into short sequences during prefilling for decoding-stage inference [1, 7, 8, 9]. In both scenarios, prefilling latency remains unimproved because such methods require computing full-length attention, which scales quadratically with input sequence length, prolonging the time-to-first-token (TTFT) for long prompts. GemFilter[10] reduces TTFT by compressing prompts into shorter versions . However, since this approach compresses the prompt prior to the most compression-sensitive initial layer, it requires a large KV cache budget to preserve accuracy. A recent method, FastKV [11], compresses hidden states subsequent to the model's middle layer. This method accelerates the prefill phase with minimal loss in accuracy. Nevertheless, because half of the layers continue to process full-length hidden states, prefill latency remains significant. Additional techniques to expedite prefilling exist in other dimensions such as sparse attention [12] or reducing memory access [13]. These approaches are orthogonal to KV cache compression methods.

In this paper, we propose ShrinKV, a KV cache compression strategy tailored to the prefilling stage, which further reduces prefilling latency. ShrinKV progressively **shrin**ks hidden states at regular layer intervals and further compresses **KV** states derived from these shrunken hidden states into the cache. This progressive compression approach is motivated by the observation that hidden states exhibit decreasing sensitivity to compression as layers are closer to the output. This aligns with intuition: the closer a layer is to the output, the fewer subsequent layers will be affected by its compression, and this can be verified across numerous tasks. The left graph in Fig. 2 plots the accuracy degradation across layers 4, 8, and 12 under different hidden state shrinkage ratios on Qasper, suggesting that layers closer to the input text exhibit lower compression tolerance. A parallel phenomenon is observed in KV cache compression: LLM layers closer to the output layer exhibit lower sensitivity to KV cache compression [4, 14, 15]. ShrinKV presents the first KV compression approach with recurrent hidden state compression and the hidden state length difference versus existing methods is shown in Fig. 1.

Specifically, ShrinKV compresses hidden states at regular layer intervals, where a portion of hidden states with the highest importance scores is retained. These importance scores are determined by the product of the attention weights and the magnitude of value states. The rationale for our importance score lies in the inconsistency between attention weights and value states. Counterintuitively, the inconsistency between attention weights and value states persists across various tasks. For example, in the right graph of Fig. 2, we plot the overlap ratio between the token index sets of the top 10% attention weights and those of the top/bottom 10% value state norms on HotpotQA. Tokens with large attention weights overlap with those corresponding to large-norm value states in only about 20% of cases. Meanwhile, their overlap with small-norm value states averages nearly 10% and is even higher in early layers, revealing that large attention weights frequently coincide with value states of small norm. Our contributions are as follows:

- We propose ShrinKV, a KV cache compression method that reduces prefill latency by progressively compressing hidden states at regular layer intervals.
- We evaluate ShrinKV on NIAH and LongBench, verifying that ShrinKV achieves competitive performance against state-of-the-art baselines under the same cache size.
- Our latency analysis shows that ShrinKV accelerates the prefill phase by over $2.4\times$ compared to using the full cache.

$*$ Bo Jiang is the corresponding author.

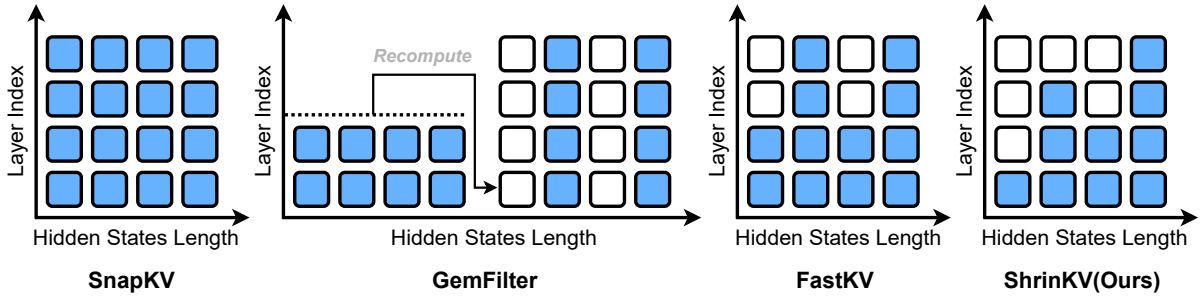

Fig. 1. The length of hidden states across model layers varies among different KV compression approaches. The blue squares represent retained hidden states, and the white squares represent discarded hidden states.

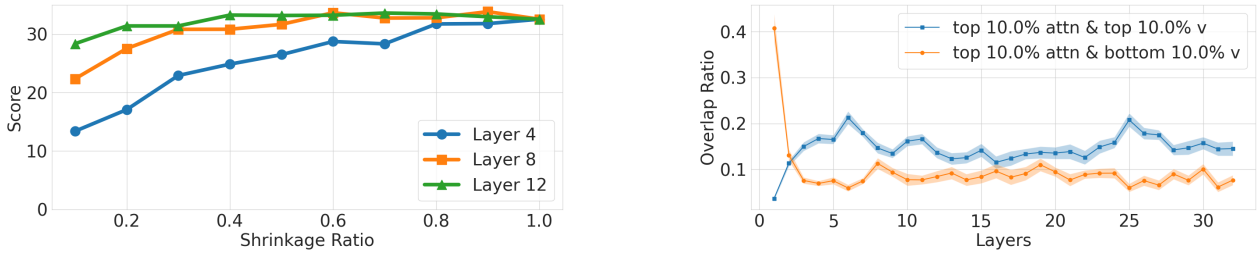

Fig. 2. (left) The accuracy on Qasper of compressing hidden states after 4, 8 and 12 layers with a cache size 128 on Mistral-7B-Instruct-v0.3. (right) Overlap ratio between token index of top 10% attention weights and token index of top/bottom 10% value states' norms on HotpotQA.

2. PRELIMINARY

LLMs comprise multiple layers of attention modules, where L denotes the number of layers. Typically, during the prefilling stage, for a specific layer l, LLMs compute key states and value states using the hidden states output by the previous layer, $\mathbf{x}_{l-1}$, and store these states in KV caches to avoid redundant computations in subsequent inference steps. Here, we provide an overview of how LLMs predict the first token from an input prompt. We first present a formulation of the calculation procedure within the attention module during the prefilling stage. Let D denote the dimension of hidden states and N denote the length of the prompt. Accordingly, the hidden states $\mathbf{x}_l$ have the shape of $N \times D$. Let H represent the number of heads in each layer, and let d represent the dimension of each head. $\mathcal{H}$ denotes the set of head indices $\{1, \cdots, H\}$. In layer l of the attention module, for head h, LLMs use three matrices $\mathbf{W}_Q^{l,h}, \mathbf{W}_K^{l,h}, \mathbf{W}_V^{l,h} \in \mathbb{R}^{D\times d}$ to transform hidden states into query states, key states and value states $\mathbf{Q}^{l,h}, \mathbf{K}^{l,h}, \mathbf{V}^{l,h} \in \mathbb{R}^{N\times d}$ respectively. We use $^{(1)}$ and $^{(2)}$ to specify the operational dimension of a function. The output of the attention module $\mathbf{O}_l \in \mathbb{R}^{N\times D}$ is defined as

$$\mathbf{O}^l = \mathrm{cat}^{(2)}_{h\in\mathcal{H}} \left\{ \mathrm{softmax}^{(2)} \left(\mathbf{Q}^{l,h} \mathbf{K}^{l,h\top} / \sqrt{d} \right) \mathbf{V}^{l,h} \right\} \mathbf{W}_O^l.$$

Here, $\mathbf{W}_O^l \in \mathbb{R}^{Hd\times D}$ transforms the weighted sum of value states into the output of the attention module. The input to the next layer, $\mathbf{x}_l$, is derived from the output of the current attention module $\mathbf{O}_l$ through a feed-forward network (FFN). The hidden states input to the first attention layer are the vector embeddings of the original prompt, denoted as $\mathbf{x}_0$. The final hidden states $\mathbf{x}_L$ are used to predict the first token following the prompt.

Algorithm 1 Compression Procedure of ShrinKV

Input: $\mathbf{x}_l$, Sequence Length N.
Parameter: Total Layers L, Observation Window w, Cache Size b, Groups G, Stages s, Shrinkage Ratio r.
Output: $\mathbf{x}_{l+1}, \mathbf{C}^l$.

▷ ***KV Compression in Attention Module:***
Get $\mathbf{Q}^{l,\mathcal{H}}, \mathbf{K}^{l,\mathcal{H}}, \mathbf{V}^{l,\mathcal{H}}$ from $\mathbf{x}_l$.
$\mathbf{A}_w^{l,\mathcal{H}} = \mathrm{softmax}^{(2)} \left(\mathbf{Q}^{l,\mathcal{H}}[-w:] \mathbf{K}^{l,\mathcal{H}\top} / \sqrt{d} \right)$.
$\mathbf{s}^{\mathcal{H}} = \mathrm{sum}^{(1)} \left(\mathbf{A}_w^{l,\mathcal{H}} \right) \odot \| \mathbf{V}^{l,\mathcal{H}\top} \|_2^{(1)}, \mathbf{s}_p^{\mathcal{H}} = \mathrm{maxpool} \left(\mathbf{s}^{\mathcal{H}} \right)$.
$\mathbf{I}^{l,g} = \mathrm{TopK} \left(\sum_{h=(g-1)H/G+1}^{gH/G} \mathbf{s}_p^h, b \right), g \in \mathcal{G}$.
$\mathbf{C}^{l,K} = \left\{ \mathbf{K}^{l,Hg/G} \left[\mathbf{I}^{l,\mathcal{G}} \right] \right\}, \mathbf{C}^{l,V} = \left\{ \mathbf{V}^{l,Hg/G} \left[\mathbf{I}^{l,\mathcal{G}} \right] \right\}$.
$\mathbf{C}^l = \left\{ \mathbf{C}^{l,K}, \mathbf{C}^{l,V} \right\}$.
▷ ***Inter-layer Hidden States Compression:***
Get $\mathbf{O}^l$ from l-layer attention module.
if $\frac{L}{s} \mid l$ and $l < L$ **then**
  $\mathbf{x}'_{l+1} = \mathrm{FFN}(\mathbf{O}^l), \mathbf{S}_p^{\mathcal{H}} = \mathrm{maxpool} \left(\mathbf{s}^{\mathcal{H}} \right)$.
  $\mathbf{I}^l = \mathrm{TopK} \left(\sum_{h=1}^{H} \mathbf{S}_p^h, \max\left(b, \lfloor N \cdot r \rfloor\right) \right)$.
  $\mathbf{x}_{l+1} = \mathbf{x}'_{l+1} \left[\mathbf{I}^l \right]$.
else
  $\mathbf{x}_{l+1} = \mathrm{FFN}(\mathbf{O}^l)$.
end if

3. METHODOLOGY

This paper introduces ShrinKV, a simple yet effective training-free KV cache compression method designed to accelerate the prefilling stage by compressing of hidden states between consecutive layers.

Table 1. Longbench evaluation using Llama3.2-1B-Instruct and Mistral-7b-Instruct-v0.3 with KV cache size 512 and 128.

Methods	Llama3.2-1B-Instruct							Mistral-7b-Instruct-v0.3						
	SD-QA	MD-QA	Summ.	Few shot	Synthetic	Code	**Avg.**	SD-QA	MD-QA	Summ.	Few shot	Synthetic	Code	**Avg.**
Full	27.85	24.00	25.90	61.47	3.00	41.03	31.61	39.23	38.01	28.82	70.70	51.25	62.00	47.30
Cache Size = 512														
SnapKV	24.82	22.70	21.68	56.94	2.75	39.56	28.94	35.87	35.06	24.91	68.39	50.50	61.35	44.77
GemFilter	20.91	**25.64**	**21.71**	50.11	**3.43**	29.26	26.28	30.99	33.49	24.12	63.04	26.08	40.76	36.78
FastKV	**24.63**	23.25	21.39	54.29	2.75	40.49	**28.57**	**35.82**	34.90	24.29	67.36	50.00	60.78	44.30
ShrinKV(8 Stages)	23.78	23.57	21.41	**54.54**	2.68	40.89	**28.57**	34.39	35.24	24.38	66.41	49.00	**61.31**	43.87
ShrinKV(4 Stages)	23.48	23.11	21.33	54.33	2.93	**41.33**	28.45	35.57	**35.36**	**24.50**	**67.84**	**50.25**	60.74	**44.49**
Cache Size = 128														
SnapKV	21.00	21.42	18.47	49.40	4.02	38.67	26.02	31.61	32.20	21.87	59.08	48.00	55.74	40.11
GemFilter	15.30	19.11	17.52	38.21	2.59	26.56	20.54	19.47	16.70	19.09	48.68	3.89	33.21	24.13
FastKV	**21.08**	**21.13**	17.67	**48.52**	2.46	39.54	**25.57**	30.16	32.23	21.15	59.30	48.75	55.31	39.79
ShrinKV(8 Stages)	20.43	20.75	17.82	47.76	2.38	**40.49**	25.38	29.54	32.15	**21.42**	59.51	45.03	54.94	39.23
ShrinKV(4 Stages)	20.64	20.96	**17.90**	48.31	**2.84**	39.63	25.53	**30.37**	**32.45**	21.36	**60.45**	**49.25**	**55.48**	**40.21**

ShrinKV estimates the importance score of each token using the corresponding attention weights and value states. At regular layer intervals, the method performs inter-layer compression on hidden states according to these scores. Within each attention module, our method compresses the key and value states into the KV cache with predefined size, where these compressed states serve as the historical context for the decoding stage. The entire algorithm is detailed in Algorithm 1.

3.1. Hidden States Compression

ShrinKV compresses hidden states after layers nL/s for $n = 1, \cdots, s-1$, where s represents the number of shrinkage stages and satisfies $s \mid L$. This procedure is illustrated in lines 9-12 of Algorithm 1. When the hidden state compression process is triggered, the algorithm first calculates token-level importance scores based on attention weights and value states, and designates the set of token indices corresponding to the highest scores as the retention index set $\mathbf{I}^l$. Following SnapKV [8], $\mathrm{maxpool}$ is applied to the importance scores prior to index selection, preserving contextual coherence. ShrinKV reduces the hidden state length to $r\times$ its former length at each compression step where $r \in (0, 1)$. The index $\mathbf{I}^l$ is determined by the positions of tokens corresponding to the top $\lfloor N \cdot r \rfloor$ values in the token-wise importance scores $\mathbf{S}_p^h$. For hidden states scheduled for compression after layer l, the uncompressed states are denoted as $\mathbf{x}'_{l+1}$ in the algorithm, and the compressed ones as $\mathbf{x}_{l+1}$. The compressed hidden states $\mathbf{x}_{l+1}$ are computed by retaining elements from $\mathbf{x}'_{l+1}$ according to the coordinates specified in $\mathbf{I}^l$. After hidden state compression, subsequent attention calculations during the prefilling stage operate on the compressed sequence length rather than the original prompt length, thereby reducing the computational time required for prefilling.

The importance score of each KV pair in a layer is calculated as the product of attention weights and the norm of value states, as outlined in lines 2–3 of Algorithm 1. To efficiently estimate the attention weights during compression, we employ a recent observation window of size w within query states following SnapKV [8]. Since the attention module output relies on weighted value state sums via attention weights and we observe an inconsistency between attention weights and value states as shown in the right graph of Fig. 2, our algorithm computes head-wise importance scores $\mathbf{s}^{\mathcal{H}}$ by multiplying estimated attention weights with value state norms.

3.2. KV Cache Compression

ShrinKV compresses key states and value states to obtain the KV cache. We present our algorithm for KV cache compression in the context of GQA [16], a widely used efficient attention computation method that allows multiple query states to share the same KV. Suppose the groups of heads are $\mathcal{G} = \{1, \cdots, G\}$, with each group containing H/G heads. The KV cache compression procedure is illustrated in lines 4–7 of Algorithm 1. For each token, we sum the pooled head-wise importance scores $\mathbf{s}^{\mathcal{H}}$ within the same group to compute the group-wise importance score, as shown in line 5 of Algorithm 1. The retention indices of key states and value states, denoted as $\mathbf{I}^{l,g}$, correspond to the positions of tokens with the highest group-wise importance scores. The key cache $\mathbf{C}^{l,K}$ and value cache $\mathbf{C}^{l,V}$ are then selected using these retention indices.

4. EXPERIMENTS

We evaluate ShrinKV on Needle-in-a-Haystack (NIAH) [17] and long context understanding tasks in LongBench [18]. We compare ShrinKV with three state-of-the-art KV cache compression methods: SnapKV [8], GemFilter [10], and FastKV [11]. For GemFilter the full-length prompt is propagated to the middle layer to facilitate its prompt compression. For FastKV, hidden states are reduced to half their original length. For our method, we set the shrinkage ratio to 0.70 and 0.84 for 4-stage and 8-stage compression, respectively, in ShrinKV. Under this setting, the length of hidden states in each layer is less than or equal to that of FastKV. Empirical results show that our model not only incurs less precision loss during compression but also achieves significant acceleration during the prefill stage.

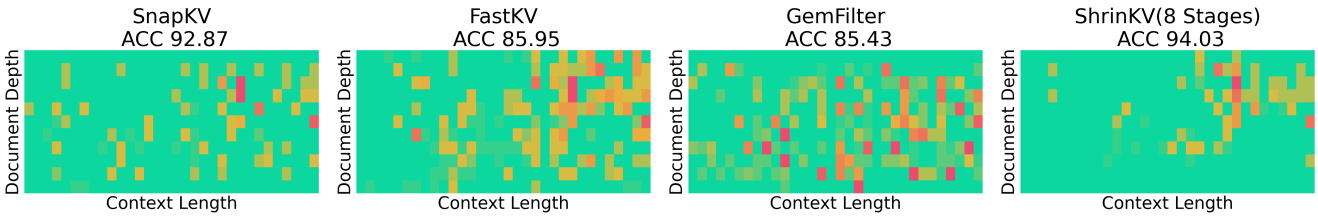

Fig. 3. The NIAH evaluation for Mistral-7B-Instruct-v0.3 with a KV cache size of 64. The horizontal axis represents context length, and the vertical axis represents the needle's depth percentage.

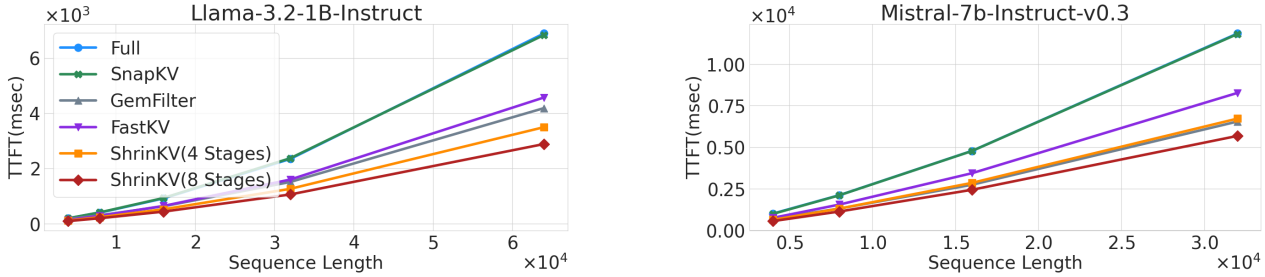

Fig. 4. TTFT evaluation for different methods with various input sequence length.

4.1. TTFT Evaluation

We evaluate the time-to-first-token (TTFT) for different methods across varying input sequence lengths on an NVIDIA RTX 3090 GPU as shown in Fig. 4, with a cache size of 512. In the left graph, we plot the TTFT of SnapKV, FastKV, ShrinKV, and full cache scenarios using Llama3.2-1B-Instruct [19]. As shown in the graph, ShrinKV achieves a 2.4× acceleration compared to full cache and SnapKV at a sequence length of 64k and exhibits a smaller TTFT than FastKV at the same length. The advantage of ShrinKV in TTFT becomes more pronounced as sequence length increases. The right graph presents the results using Mistral-7b-Instruct-v0.3 [20]. Here, our method maintains improved TTFT compared to SnapKV and FastKV. Although 4-stage ShrinKV exhibits slightly slower TTFT than GemFilter, our method demonstrates significantly less accuracy degradation on downstream tasks under the same conditions, as shown in the LongBench evaluation results in Table 1.

4.2. Needle-in-a-Haystack

The NIAH task evaluates LLMs by hiding a specific key piece of information ("the needle") within a lengthy text corpus ("the haystack") and prompting the model to extract it. We conduct NIAH under the classical setting [17]. We conduct NIAH evaluations for Mistral-7B-Instruct-v0.3 with a KV cache size of 64. The experiments systematically vary 32 text lengths up to 32k and 10 different needle positions, ranging from 0% to 100% depth percentage, creating a total of 320 scenarios to test retrieval accuracy. We use a red-to-green gradient to represent accuracy from low to high and generate heatmaps for the accuracy of different methods, as shown in Fig. 3. The average accuracy of each method is given at the top of its corresponding heatmap, revealing that ShrinKV with 8-stage compression achieves higher accuracy than SnapKV, GemFilter, and FastKV. Specifically, ShrinKV provides a 1.16% average accuracy improvement over SnapKV, which propagates the full prompt across all layers.

4.3. Long Context Understanding

LongBench [18] is a comprehensive evaluation benchmark designed to assess the long-text understanding abilities of large language models across various tasks. We perform LongBench evaluations for Llama-3.2-1B-Instruct and Mistral-7B-Instruct-v0.3 with KV cache sizes of 512 and 128. As shown in Table 1, we provide average scores by category and the overall average score for 16 tasks belonging to 6 categories. The best score among FastKV, GemFilter, and ShrinKV is highlighted in bold. Using Llama-3.2-1B-Instruct, ShrinKV achieves scores close to those of FastKV and SnapKV with a cache size of 128. GemFilter attains the best scores for most task categories but has a lower average score compared to FastKV and ShrinKV. Using Mistral-7B-Instruct-v0.3, ShrinKV achieves higher average scores than FastKV and GemFilter across different cache sizes. As summarized in the table, our method achieves an overall score improvement of 7.71 over GemFilter and 0.19 over FastKV with a cache size of 512. With a cache size of 128, ShrinKV achieves an overall score improvement of 16.08 over GemFilter and 0.42 over FastKV.

5. CONCLUSION

In this paper, we propose ShrinKV, a training-free KV cache compression strategy for the prefill stage that reduces prefill latency. Observing that sensitivity to compression decreases as the layers of LLMs deepen, ShrinKV progressively shrinks hidden states after a regular number of layers and compresses KV states. Motivated by the inconsistency between attention weights and the magnitudes of value states, we use the product of attention weights and the norm of value states as importance scores for retaining KV and hidden states. Evaluations on NIAH tasks and the multi-task benchmark LongBench confirm that ShrinKV yields competitive results compared to state-of-the-art methods. Latency analysis demonstrates that ShrinKV can accelerate prefill by more than 2.4× relative to using a full KV cache.

6. REFERENCES

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

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

[3] Zijian Lew, Joseph B Walther, Augustine Pang, and Wonsun Shin, "Interactivity in online chat: Conversational contingency and response latency in computer-mediated communication," *Journal of Computer-Mediated Communication*, vol. 23, no. 4, pp. 201–221, 2018.

[4] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al., "Cachegen: Kv cache compression and streaming for fast large language model serving," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 38–56.

[5] Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, et al., "R-kv: Redundancy-aware kv cache compression for training-free reasoning models acceleration," *arXiv preprint arXiv:2505.24133*, 2025.

[6] Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov, "Rocketkv: Accelerating long-context llm inference via two-stage kv cache compression," *arXiv preprint arXiv:2502.14051*, 2025.

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

[8] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen, "Snapkv: Llm knows what you are looking for before generation," *Advances in Neural Information Processing Systems*, vol. 37, pp. 22947–22970, 2024.

[9] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou, "Identify critical kv cache in llm inference from an output perturbation perspective," *arXiv preprint arXiv:2502.03805*, 2025.

[10] Zhenmei Shi, Yifei Ming, Xuan-Phi Nguyen, Yingyu Liang, and Shafiq Joty, "Discovering the gems in early layers: Accelerating long-context llms with 1000x input token reduction," *arXiv preprint arXiv:2409.17422*, 2024.

[11] Dongwon Jo, Jiwon Song, Yulhwa Kim, and Jae-Joon Kim, "Fastkv: Kv cache compression for fast long-context processing with token-selective propagation," *arXiv preprint arXiv:2502.01068*, 2025.

[12] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al., "Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention," *Advances in Neural Information Processing Systems*, vol. 37, pp. 52481–52515, 2024.

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

[14] Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao, "Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference," in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 3258–3270.

[15] Zicong Tang, Shi Luohe, Zuchao Li, Baoyuan Qi, Guoming Liu, Lefei Zhang, and Ping Wang, "Spindlekv: A novel kv cache reduction method balancing both shallow and deep layers," *arXiv preprint arXiv:2507.06517*, 2025.

[16] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 4895–4901.

[17] Gregory Kamradt, "Needle in a haystack - pressure testing llms," 2023.

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

[19] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al., "The llama 3 herd of models," *arXiv e-prints*, pp. arXiv–2407, 2024.

[20] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed, "Mistral 7b," 2023.