SCA: Selective Compression Attention for Efficiently Extending the Context Window of Large Language Models

Anonymous ACL submission

Abstract

Large language models (LLMs) have achieved impressive performance across various domains, but the limited context window and the expensive computational cost of processing long texts restrict their more comprehensive application. In this paper, we propose Selective Compression Attention (SCA), a general and effective method to expand the context window and reduce memory footprint by compressing the KV cache of LLMs. Specifically, through preliminary experiments, we found that the KV cache contains many similar vectors, resulting in information redundancy, which can be compressed by retaining representative vectors and discarding others. Therefore, SCA continuously selects the most distinctive vectors to keep through a greedy algorithm, reducing information loss during compression. Extensive experiments on various tasks verify the effectiveness of our method. Compared with existing methods, SCA can significantly reduce the impact on model performance under the same compression ratio. Furthermore, the context window of LLMs can be efficiently expanded using SCA without any training, which can even achieve better performance than specially fine-tuned long context models.

1 Introduction

Transformer-based (Vaswani et al., 2017) large language models (LLMs) have excellent capabilities, which have extensively promoted the development of various natural language processing applications (Wolf et al., 2019; Thoppilan et al., 2022; Touvron et al., 2023a; OpenAI, 2023) and provided a possibility for artificial general intelligence. However, due to their huge size, their deployment is very expensive. In particular, the quadratic cost of attention layers and the growing KV cache make the overhead of LLMs unacceptable when processing long texts, which limits the application and development of LLMs in long context scenarios.

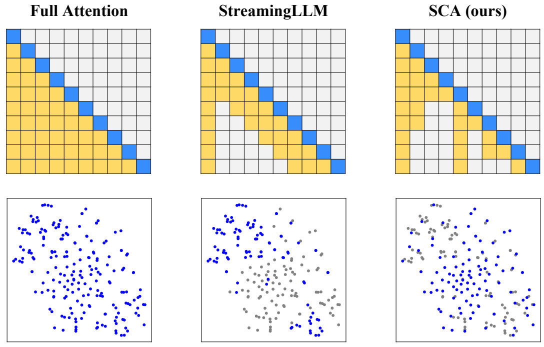


Figure 1: Upper plots illustrate attention maps applying different methods. Lower plots show the distribution of vectors retained by different attention methods after t-SNE dimensionality reduction. The distribution of SCA retained vectors is closer to the original distribution than StreamingLLM, so it can keep more information.

Significant efforts have been made to improve the efficiency and extend the context window for LLMs. For example, some methods (Beltagy et al., 2020; Xiao et al., 2023; Zhang et al., 2023) use a manually set sparse attention mode to limit the maximum size of the attention calculation window. However, they will lose valuable information, causing the performance to decrease significantly. There are some other methods (Wu et al., 2022; Wang et al., 2023b) that only use the retrieved most relevant chunks to calculate attention but still need to keep the complete KV cache. Another works improve efficiency by changing the model structure (Kitaev et al., 2020; Gu and Dao, 2023). However, such methods require retraining or fine-tuning the model, making their application costly.

Therefore, this paper aims to propose a method that can overcome the shortcomings of previous approaches. Specifically,

1. It can effectively compress the KV cache and expand the context window of LLMs.

2. It can retain most original information in the KV cache during the compression process and

reduce the impact on performance.

3. It is model-independent, plug-and-play, and does not require training or fine-tuning.

We first conducted a preliminary experiment to explore the feasibility. Fortunately, we found that the KV cache has many redundant vectors that could be deleted. Specifically, the cosine similarity between many vectors in the KV cache is extremely high. The similar vectors provide similar information when calculating attention. Therefore, we can compress the KV cache and retain its original information by reserving representative vectors and removing similar redundant vectors.

Based on the preliminary experimental results, we propose the Selective Compression Attention (SCA) method, which can effectively compress the KV cache, improve the efficiency of LLMs, and extend their context window. Specifically, our method uses a greedy algorithm to select the least redundant vector based on the current retained result to reserve, ensuring that more different information can be kept at each step during compression. When the KV cache length reaches a given maximum threshold, it can be compressed using the SCA approach to provide free space, allowing LLMs to receive more context. Moreover, unlike the recently proposed AutoCompressors (Chevalier et al., 2023), our method does not require fine-tuning and can be easily applied to any LLMs.

To verify the effectiveness of our proposed method, we conduct extensive experiments on different LLMs and datasets. On the one-shot and zero-shot short text tasks, the performance after using SCA to compress the KV cache is almost the same as the original full attention, verifying that SCA can retain most of the original information during compression. For the long context tasks, our method can effectively extend the LLMs' original context window and ensure the fluency and accuracy of the generated results. Especially, SCA can still achieve 100% accuracy on the passkey retrieval task after extending the context window size of Llama2-13B-Chat (Touvron et al., 2023b) to 12k. Furthermore, using SCA to extend the context length of Vicuna1.5-7b (Zheng et al., 2023) to 16k can even perform better than the fine-tuned Vicuna1.5-7b-16k on real long context tasks.

In summary, our main contributions are the following: (1) We analyze and verify the feasibility of compressing the KV cache. By exploring the similarities between vectors, the preliminary experiment demonstrates that the KV cache contains much redundant information. (2) We propose an efficient and plug-and-play approach, which can compress the KV cache and keep most of the original information by retaining the representative vectors. (3) We conduct extensive experiments to show the powerful potential of our method, which can effectively extend the context window and reduce the memory footprint for different LLMs.

2 Related Work

Extensive research has been done on efficient inference and context window extension of LLMs.

An intuitive idea is manually setting sparse attention to limit computational complexity (Beltagy et al., 2020; Ding et al., 2023; Han et al., 2023). For example, StreamingLLM (Xiao et al., 2023) only retains the most recent tokens and several initial tokens for stable attention computation. StreamingLLM can perform language modeling of millions of tokens. However, it loses much original information and cannot truly enhance LLMs' ability to remember and use long contexts. Recently, Han et al. (2023) proposed H_2O, a heuristic KV cache eviction policy. H_2O compresses the KV cache by evicting tokens with the smallest accumulated attention score. However, the score calculated only based on the current KV cache is one-sided, which may cause it to discard tokens needed in the future. Unlike the previous methods, our approach selects the most representative vectors based on the vector distribution of the KV cache so that more different information can be retained, significantly reducing the information loss during the compression process.

The second type of method retrieves the most relevant chunk in the KV cache for the attention calculation (Wu et al., 2022; Zhong et al., 2022; Wang et al., 2023b; Lu et al., 2024). Although these methods can reduce the overhead of attention calculation, they still need to store the complete KV cache. Therefore, they can not solve the problem of the KV cache increasing linearly as the context length increases. When the context is very long, they need to offload the KV cache to the CPU, increasing communication overhead between the GPU and the CPU. In contrast, SCA can ensure the KV cache size does not exceed a given threshold, significantly reducing the memory footprint of LLMs when processing long contexts.

Another type of work changes the model struc-

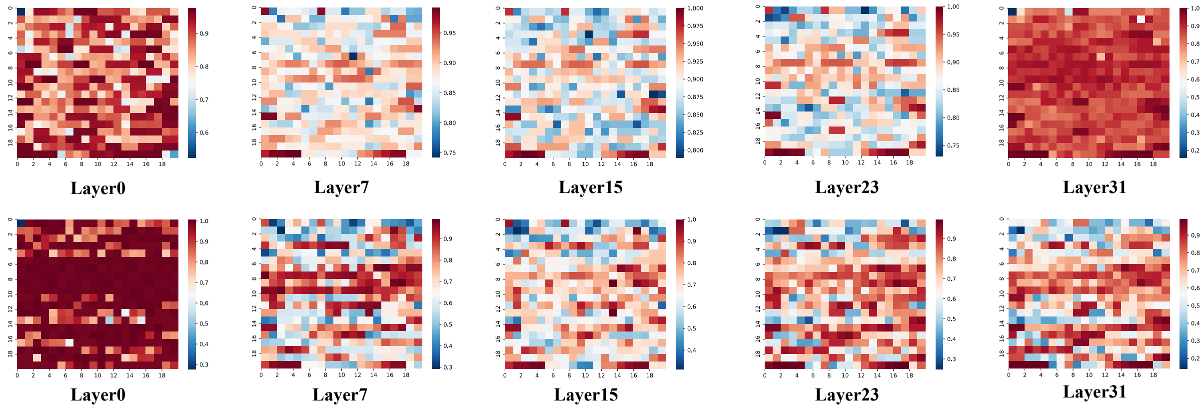


Figure 2: Visualization of the redundancy of each token vector in Key (Upper) and Value (Lower) caches at different layers of Llama2-7B. To facilitate visualization, we convert the 400 vector redundancy into a 20×20 matrix. The high redundancy of a token vector indicates that there are other vectors in the cache that are very similar to it.

ture to make it more efficient (Dai et al., 2019; Kitaev et al., 2020; Peng et al., 2023). For example, Transformer-XL (Dai et al., 2019) uses a segment-level recurrence mechanism to expand its receptive field and capture longer dependencies while fixing the attention window size. Reformer (Kitaev et al., 2020) proposes a new attention module that uses locality sensitive hashing attention to reduce the computational cost from quadratic to superlinear complexity. However, such methods require re-training, making their deployment on LLMs costly. In contrast, our approach is plug-and-play and can be easily adapted to any LLMs.

3 Preliminary Experiment

In this section, we carefully explore the characteristics of the KV cache in LLMs. Specifically, we conducted experiments to answer two questions: (1) Is there information redundancy in the KV cache? (2) Can the KV cache be effectively compressed by only retaining representative vectors?

3.1 Experimental Setup

We conducted experiments on the validation set of PG19 (Rae et al., 2019) based on Llama2-7B and Llama2-7B-Chat (Touvron et al., 2023b). Specifically, the books in the PG19 validation set are truncated from the right, allowing LLMs to encode fixed-length contexts and obtain their corresponding KV cache. Then, we measure the degree of information redundancy by the cosine similarity between different vectors in the KV cache. Similar key and value vectors have similar meanings in the latent space, and the information they provide in attention calculations is also similar. Therefore, we designed an information redundancy metric based on cosine similarity between vectors:

$$\text{redundancy} = \frac{\sum_{i=1}^{n} \text{redundancy}_i}{n} \qquad (1)$$
$$\text{redundancy}_i = \max\left(\text{sim}\left(w_i, w_{\neq i}\right)\right)$$

where n represents the number of vectors in the matrix W. Since Llama2 uses RoPE (Su et al., 2021) positional encoding, when calculating the redundancy of the key matrix, we first add position information to it to make it consistent with the form of attention calculation. Furthermore, considering the tokens' integrity, we calculate the cosine similarity after concatenating the vectors of all heads for each token. Finally, we average the redundancy of all layers to measure the overall redundancy of the KV cache generated by the LLMs.

Context length	Llama2-7B	Llama2-7B-Chat
200	0.89/0.67	0.88/0.64
400	0.89/0.69	0.88/0.66
800	0.89/0.70	0.88/0.67
1600	0.89/0.71	0.88/0.67
3200	0.88/0.72	0.88/0.69

Table 1: Redundancy of Key/Value cache of different context lengths in Llama2-7B and Llama2-7B-Chat.

3.2 Experimental Results

The main experimental results are shown in Table 1. As we can see, the KV cache generated by LLMs has apparent information redundancy, whether the key or value matrix. Specifically, the average redundancy of the key matrix is between 0.88-0.89, and the average redundancy of the value matrix is between 0.64-0.72, which shows that most of the

Algorithm 1 SCA

1: **Input:** $K \in \mathbb{R}^{n\times d}$, $V \in \mathbb{R}^{n\times d}$, **m**
2: Initialize: R = [n], D = [1, 2, ..., n-1]
3: K' = Relative_Position(K)
4: Sim^K, Sim^V = $Cos\_Sim(K')$, $Cos\_Sim(V)$
5: **for** $i = 1$ to **m do**
6: Calculate Add^K and Add^V based on Sim^K and Sim^V respectively
7: t = $\arg\min_{j\in D}$ (Add$^K(k_j)$ + Add$^V(v_j)$)
8: R, D = R.append(t), D.remove(t)
9: **end for**
10: R = R.sort()
11: **Return** K[R], V[R]

token vectors in the matrix have other vectors that are very similar to them. In addition, as the length increases, the redundancy of the KV cache will also increase, especially for the value matrix. This experimental result provides us with the possibility to compress the KV cache by retaining representative token vectors and deleting redundant vectors.

For a more fine-grained analysis, we visualized the redundancy of each token vector in the KV cache at different layers of Llama2-7B when the input context length is 400. As shown in Figure 2, most of the token vectors have high redundancy, indicating that there are other vectors in the matrix that are very similar to them. These results further demonstrate that we can effectively compress the KV cache and maintain the original information by selecting one representative from the set of similar vectors to retain. Furthermore, we find that the first token vector of the KV cache in the first and last layers does not have other similar vectors, indicating that it has unique information. This observation provides another explanation for StreamingLLM and LM-Infinite (Han et al., 2023) methods, i.e., if the initial tokens are discarded, their unique information will be lost, resulting in a sharp decline in the performance of the model.

4 Method

This section details the proposed approach. First, we present the problem definition in 4.1, then introduce the design ideas of our method in 4.2, and give the implementation details in 4.3.

4.1 Problem Definition

Through the preliminary experiment, we found that the KV cache of LLMs has a lot of redundant information, and it can be effectively compressed by retaining representative tokens and discarding other redundant vectors. In this way, we can improve the computational efficiency and extend the context window for LLMs.

Therefore, the problem we want to solve can be defined as a matrix compression task. Specifically, given the matrix $W = (w_1, w_2, \ldots, w_n)$, it contains **n** vectors. Our goal is to select **m** vectors from these **n** vectors to retain and delete other vectors, thereby obtaining the compressed matrix $W^* = (w_1^*, w_2^*, \ldots, w_m^*)$. In addition, we require that the minimum amount of information is lost during the compression process. The information amount of the compressed matrix W^* is inversely proportional to the redundancy. The lower the redundancy, the more information W^* contains, and the less information is lost during the compression process. Consequently, our final goal is to propose a method that can compress W into W^* and ensure that the redundancy of W^* is minimal.

4.2 Selective Compression Attention

Determining the best selection strategy with the lowest redundancy presents a combinatorial challenge, which makes it difficult to find the optimal solution in a reasonable time. Therefore, we use a greedy algorithm to effectively obtain the local optimal selection result for matrix compression.

Based on the principle of greedy algorithm, we divide the original problem into multiple subproblems and obtain the final result through multi-step calculation. At each step, we select one vector to retain, thereby obtaining the final result through **m** steps. Specifically, for step t, knowing $W_{t-1}^* = (w_1^*, w_2^*, \ldots, w_{t-1}^*)$, our goal is to select one of the unretained vectors from W to add to W_{t-1}^* and ensure that the redundancy of the resulting W_t^* matrix is minimal. According to the redundancy metric in Equation (1), the change in redundancy of W_t^* compared to W_{t-1}^* after adding w_t^* consists of two parts. First, adding w_t^* may cause the most similar vector of each vector in W_{t-1}^* to change, resulting in their redundancy increases:

$$\text{Add}_1 = \sum_{i=1}^{t-1} \max\left(0, \text{sim}\left(w_i^*, w_t^*\right) - \text{redundancy}_i\right) \quad (299)$$

Second, the redundancy caused by the similarity between w_t^* itself and the retained vectors:

$$\text{Add}_2 = \max\left(\text{sim}\left(w_t^*, w_{<t}^*\right)\right)$$

Therefore, to ensure local optimality, for each step, we select the vector that leads to the smallest increase in the redundancy value of the two parts to retain ($Add = Add_1 + Add_2$). The main idea of our method is to preserve vectors with different meanings as much as possible so that the vector distribution of the compressed matrix can be similar to that before, thus reducing the loss of information (Figure 1). Furthermore, because the Add values of all candidate vectors can be calculated in parallel, the time required for each step is very short, ensuring the efficiency of our method.

4.3 Implementation Details

The implementation of Selective Compression Attention is summarized in Algorithm 1. For the KV cache compression, we have several important details to consider.

First, because LLMs generally pay more attention to the most recent tokens (Xiao et al., 2023; Han et al., 2023; Zhang et al., 2023), we retain the most recent one or more tokens during initialization to ensure that the most recent important information is not lost (Line 2 of Algorithm 1).

Second, most existing LLMs use relative position encoding (Zeng et al., 2022; Touvron et al., 2023b; Biderman et al., 2023; Team, 2023; Zheng et al., 2023). Therefore, we will first add position information to the key matrix and then calculate its similarity (Lines 3-4) to ensure that it is consistent with the attention calculation process.

Third, since the vectors in the key and value matrices correspond to each other, their selection results must also be the same. Otherwise, the attention calculation results will seriously deviate from the original results. Therefore, in each step, we will consider the redundancy of key and value matrices together to make the selection (Lines 7-8).

Our method only focuses on the KV cache, which is general and can be applied to any LLMs. Moreover, our method does not require any training and is plug-and-play, thus significantly reducing the difficulty and cost of its deployment.

5 Experiments

In this section, we conduct extensive experiments to verify the effectiveness of our proposed method. Specifically, we first verify whether using SCA to compress the KV cache affects the performance of the LLMs in Section 5.1. Then, we verify the context window extension capability of our method based on a variety of tasks, including language modeling tasks (Seciton 5.2), passkey retrieval tasks (Section 5.3), and real long context tasks in the L-Eval benchmark (Section 5.4). Finally, we conduct fine-grained ablation experiments in Section 5.5 to further analyze our approach.

We use a single NVIDIA RTX A6000 48GB GPU for experiments. During inference, we use the greedy search for LLMs to generate results. We mainly compare several advanced baselines, including:

- StreamingLLM (Xiao et al., 2023): when the KV cache's length reaches the threshold, the most recent and first four tokens are retained.

- Sparse Attention: uses a stride of 2 to retain tokens in the KV cache. If multiple compressions are performed, its effect is similar to the Dilated Attention (Ding et al., 2023).

- H_2O (Zhang et al., 2023): retains most recent tokens and the tokens with higher accumulated attention scores in the KV cache.

Method	IMDB	RACE	AG News	Cosmos QA	Avg.
Full	91.6	35.4	76.0	35.4	59.6
Stream	80.2	32.0	66.8	34.0	53.3
Sparse	50.6	15.4	70.8	20.6	39.6
H_2O	**90.0**	34.8	71.8	33.2	57.5
SCA	**90.0**	**35.4**	**75.8**	**34.8**	**59.0**

Table 2: The performance of different compression methods on Llama2-7B. We compress the KV cache to 50% of its original length and then predict the results. The average context lengths of the four datasets are 1048, 462, 367, and 184 tokens.

Considering the token's integrity, SCA is performed in units of tokens during compression. Specifically, we concatenate the vectors of all attention heads in the KV cache to construct the token vector for SCA. Moreover, based on the experimental results in Section 5.5, we only use the SCA algorithm for the last layer and let all layers share the selection results to further improve efficiency.

5.1 The Impact of Compression

Setting We selected four commonly used natural language processing datasets: IMDB (Maas et al., 2011), RACE (Lai et al., 2017), AG News (Zhang et al., 2015), and Cosmos QA (Huang et al., 2019), including sentiment classification, reading comprehension and text classification tasks, and conducted experiments based on Llama2-7B. For each dataset,

Model	Method	PG19					ArXiv				
		4k	8k	16k	32k	64k	4k	8k	16k	32 k	64k
Llama2-7B	Full	6.5	165.6	$>10^3$	OOM	OOM	3.8	100.9	$>10^3$	OOM	OOM
	Local	6.5	171.9	947.6	$>10^3$	$>10^3$	3.8	132.1	681.4	$>10^3$	$>10^3$
	Stream	6.5	6.8	6.9	7.0	7.1	3.8	3.6	3.3	3.1	3.0
	Sparse	6.5	7.0	7.0	7.1	7.2	3.8	3.6	3.4	3.1	3.1
	H_2O	6.5	6.8	7.0	7.3	7.7	3.8	3.5	3.3	3.1	3.1
	SCA	6.5	**6.7**	6.9	7.0	7.1	3.8	3.5	3.3	3.1	3.0
Llama2-7B-Chat	Full	6.5	204.4	$>10^3$	OOM	OOM	3.8	180.9	$>10^3$	OOM	OOM
	Local	8.6	343.1	$>10^3$	$>10^3$	$>10^3$	5.2	226.1	947.0	$>10^3$	$>10^3$
	Stream	8.6	9.0	9.2	9.4	9.5	5.2	4.8	4.5	4.2	4.1
	Sparse	8.6	9.0	9.3	9.6	9.9	5.2	4.8	4.5	4.2	4.1
	H_2O	8.6	8.9	9.2	9.9	11.4	5.2	4.8	4.5	4.2	4.1
	SCA	8.6	**8.8**	**9.0**	**9.2**	**9.4**	5.2	**4.7**	**4.4**	**4.1**	**4.0**

Table 3: Perplexity on PG19 and ArXiv of Llama2-7B and Llama2-7B-Chat with different compression methods. "Local" means only the most recent token is retained during compression. "OOM" means out-of-memory.

we randomly sample 500 instances from their test sets. Because the KV cache of few-shot in-context learning naturally has a lot of redundant information, it is simple to compress. Therefore, we try to reduce the number of demonstrations to increase the compression difficulty. Specifically, we adopt the one-shot for IMDB and AG News, and the zero-shot for RACE and Cosmos QA. For all compression methods, we set the compression ratio to 50%. For H_2O and SCA, we first let them keep the 25% target retention number of the most recent tokens and then select 75% from the remaining tokens. Finally, we use accuracy to evaluate the model performance.

Results We show the evaluation results in Table 2. As we can see, the performance of Sparse Attention is the worst, which shows that the method based on fixed stride loses much original information during the compression process. The performance of Stream and H_2O is better than Sparse Attention, but they still lead to a significant decrease in the model's accuracy on some datasets. In contrast, SCA can achieve competitive performance with Full Attention (without compression) on all datasets, which shows that it can retain most of the original information during compression, allowing the model to still make correct predictions. More experimental results are shown in Appendix A, and our method can perform well under different compression ratios.

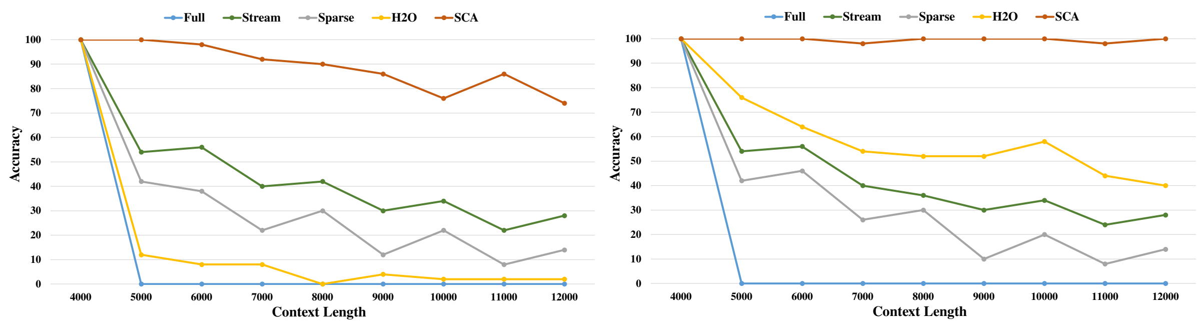
5.2 Performance on Language Modeling

Setting Excellent language modeling capability is essential for LLM to complete various tasks. We use the PG19 test set (Rae et al., 2019) and ArXiv corpora of RedPajama (Computer, 2023) to evaluate the language modeling ability of LLMs with different length contexts. For Arxiv, we randomly sample 100 samples for testing. We filter samples whose length is less than the required length and truncate content that exceeds the given length. To extend the context window of LLMs, whenever the length of the KV cache reaches 4000, we use a compression method to compress it to 2000 so that the model can accept new texts. For H_2O and SCA, we make them keep the 128 most recent tokens first[1]. Similar to previous work (Xiao et al., 2023; Ding et al., 2023; Zhang et al., 2023), we use perplexity (PPL) to measure the language modeling ability of the model.

Results As shown in Table 3, the PPL increases significantly when the input text length exceeds the LLM's context window size. By keeping the most recent tokens (Local), the memory footprint will not exceed the maximum as the context length increases, but it destroys the language modeling ability of LLMs. In contrast, other compression methods can keep PPL within an acceptable range after expanding the context window. In particular, our proposed SCA can achieve the lowest PPL in most cases, which can maintain LLMs' powerful language modeling ability even when the context window size is expanded 16×.

As we can see from the experimental results, it is relatively easy to make LLMs implement long text language modeling through compression. However, being able to perform language modeling does not

[1]If not specified below, this setting is used by default.



(a) Performance of different methods on Llama2-7B-Chat.

(b) Performance of different methods on Llama2-13B-Chat.

Figure 3: Passkey retrieval accuracy of two LLMs with different extended context window sizes. For different test lengths, we randomly generate 100 test samples for evaluation.

mean that LLMs can capture and exploit content in long texts (Xiao et al., 2023). Therefore, we further explore our method's effectiveness through other more complex tasks.

5.3 Performance on Passkey Retrieval Task

Setting Passkey retrieval (Mohtashami and Jaggi, 2023) is a synthetic task that requires LLMs to retrieve a simple passkey (a five-digit random number) from a long meaningless text sequence. This task randomly inserts the passkey into any position of the input context, which can test whether LLMs can be aware of and use information from different positions in the input context. We conduct experiments based on two LLMs of different sizes, Llama2-7B-Chat and Llama2-13B-Chat, to verify whether our method can find and retain valuable information during compression.

Results The experimental results are shown in Figure 3. It can be seen that when the input text length is 4k, both LLMs can achieve 100% accuracy, indicating they have strong passkey retrieval capabilities. However, when the length exceeds the context window size, the retrieval accuracy drops sharply to 0%. Moreover, even if the context window is expanded by existing methods, the accuracy still drops significantly for long texts. These results show that although the previous approaches can achieve language modeling for long texts, they cannot effectively discover and retain valuable information, resulting in the information corresponding to the passkey being deleted during compression.

In contrast, SCA can maintain high retrieval accuracy under extended context length. Especially on Llama-13B-Chat, even if the extended length is three times the original context window size, SCA can still achieve 100% accuracy. This verifies the effectiveness of SCA, which uses the distribution of token vectors in the KV cache as the principle for selection, allowing it to retain valuable information and still perform well after compression.

5.4 Performance on Real Long Context Tasks

Setting Language modeling and passkey retrieval tasks still cannot comprehensively reflect LLMs' long context capabilities. Therefore, to further verify the effectiveness of our method, we conducted experiments on the long context evaluation benchmark L-Eval (An et al., 2023). Since the evaluation of open-ended tasks has fairness issues and the closed-ended tasks can better reflect unbiased results, we only use L-Eval's closed-ended tasks to evaluate the model's performance, which includes various question styles such as multiple choice questions (Coursera, QuALITY, TOFEL), math problems (GSM), code understanding (CodeU), and true or false questions (SFiction). The evaluation metric used for these tasks is accuracy. Different from the previous setting, we extend the LLMs context window by compressing the KV cache length from 4000 to 3000. We use various methods to expand Llama2-7B-Chat and Vicuna1.5-7B (Zheng et al., 2023) with a 4k original window size to 8k and 16k and evaluate their performance.

Results As shown in Table 4, the performance of using StreamingLLM and H_2O to extend the context window is even worse than the original LLMs, which means they lose much information after multiple compression, so they cannot truly expand the context window for LLMs. In particular, although H_2O has little impact on the accuracy when compressing short text tasks, its performance on real long text tasks is poor, especially on Vicuna1.5-7B. These results show that the method based on atten-

Model	Tokens	Coursera	GSM	QuALITY	TOFEL	CodeU	SFiction	Avg.
Llama2-7B-Chat	4k	32.4	29.0	**37.6**	53.2	1.1	60.1	35.6
+ Stream	16k	23.0	18.0	30.7	53.2	1.1	53.9	30.0
+ H_2O	16k	32.1	14.0	35.1	53.2	1.1	58.6	32.4
+ SCA	8k	36.6	**31.0**	**37.6**	**57.2**	2.2	62.5	37.9
+ SCA	16k	**38.5**	**31.0**	**37.6**	**57.2**	2.2	**64.1**	**38.4**
Longchat1.5-7B-32k	32k	33.0	18.0	37.6	39.8	**3.3**	57.0	31.5
Vicuna1.5-7B	4k	36.2	19.0	38.1	51.3	3.3	56.3	34.0
+ Stream	16k	35.6	22.0	35.6	46.8	1.1	61.7	33.8
+ H_2O	16k	28.2	1.0	28.2	18.8	0.0	46.1	20.4
+ SCA	8k	**40.0**	**24.0**	**39.6**	53.2	4.4	66.4	37.9
+ SCA	16k	39.0	**24.0**	39.1	53.2	4.4	**69.5**	**38.2**
Vicuna1.5-7B-16k	16k	38.7	19.0	**39.6**	**55.4**	**5.5**	60.2	36.4

Table 4: Performance of different methods on closed-ended tasks of L-Eval benchmark. **Tokens** denotes the maximum input length. The input context is truncated from the right according to the given maximum length.

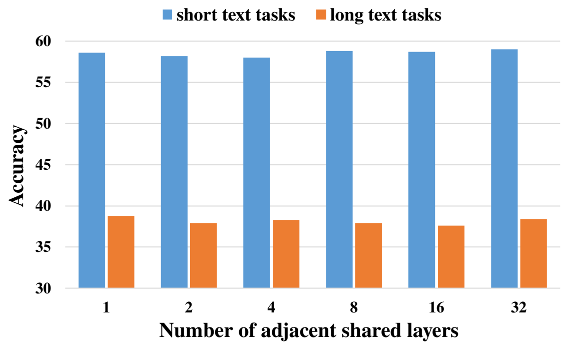


Figure 4: Average accuracy on short and long text tasks using different sharing strategies. Setting the number of adjacent shared layers to 32 indicates that the selection result is calculated only based on the KV cache of the last layer and shared with all layers for compression.

tion scores to select important tokens can not apply to long texts because the current accumulated attention scores cannot reflect its importance for distant future predictions.

In contrast, using SCA to extend the context window can achieve better performance than the original LLMs. Specifically, expanding the context window size of the two LLMs to 16k can improve the accuracy by **2.8** and **4.2**, respectively. Moreover, even compared with two specially fine-tuned long context LLMs, Longchat1.5-7B-32k (Dacheng Li and et al., 2023) and Vicuna1.5-7B-16k, our method still performs better. These experimental results show that compared with previous methods, by retaining more different representative vectors, SCA can keep enough original information in the KV cache even after multiple compressions, thereby ensuring excellent performance. More experimental results on Llama2-13B-Chat and Vicuna1.5-13B are shown in Appendix B.

5.5 Ablation Experiments

Setting Although SCA can compress the KV caches of all layers in parallel, calculating the selection results for each layer requires many computing resources. Therefore, we test the performance of sharing the selection results between adjacent layers on the short text tasks in Section 5.1 and the long text tasks in Section 5.4. Specifically, we set 6 different sharing strategies for Llama2-7B and Llama2-7B-Chat and evaluate their performance.

Results As shown in Figure 4, different sharing strategies have little impact on performance. We believe this is because the vector relationship of the KV caches in most layers is similar, i.e., if two tokens' vectors are similar in the last layer, their vectors in other layers are also likely to be similar. Therefore, we share selection results in all layers to improve efficiency. Due to space constraints, more analysis experiments are presented in Appendix C.

6 Conclusion

In this paper, we first explore the characteristics of the KV cache and verify the feasibility of compressing it by retaining representative vectors and discarding others. Based on these experimental results, we propose a general and plug-and-play method called SCA, which adopts a greedy algorithm to minimize the information loss during the compression process. Extensive experiments on various tasks demonstrate the effectiveness of our approach, which can compress the KV cache with little impact on the model performance. Furthermore, SCA can easily and efficiently expand the context window of LLMs, and its performance is even better than the fine-tuned long context LLMs.

7 Limitations

Although we conduct experiments on various long text tasks, it still has limitations and cannot comprehensively evaluate the performance of LLMs after expanding the context window. How to effectively and accurately evaluate LLMs' long context handling capability remains an open question. In the future, we will explore better evaluation methods to verify the effectiveness of our approach.

In addition, our proposed method is general, but in this paper, we only focus on its performance on large language models. Recently, multimodal large language models (Zhu et al., 2023; Liu et al., 2023; OpenAI, 2023) have attracted widespread attention from researchers. Since they need to receive input from different modalities, they require a larger context window. In the future, we will further explore the performance of SCA on multimodal large language models.

References

Chen An, Shansan Gong, Ming Zhong, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. 2023. L-eval: Instituting standardized evaluation for long context language models. *ArXiv*, abs/2307.11088.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *ArXiv*, abs/2004.05150.

Stella Biderman, Hailey Schoelkopf, Quentin G. Anthony, and et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. *ArXiv*, abs/2304.01373.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. *ArXiv*, abs/2305.14788.

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Anze Xie Dacheng Li, Rulin Shao and et al. 2023. How long can open-source llms truly promise on context length?

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *Annual Meeting of the Association for Computational Linguistics*.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher R'e. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *ArXiv*, abs/2205.14135.

Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, and Furu Wei. 2023. Longnet: Scaling transformers to 1, 000, 000, 000 tokens. *ArXiv*, abs/2307.02486.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *ArXiv*, abs/2312.00752.

Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2023. Lm-infinite: Simple on-the-fly length generalization for large language models. *ArXiv*, abs/2308.16137.

Lifu Huang, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Cosmos qa: Machine reading comprehension with contextual commonsense reasoning. In *Conference on Empirical Methods in Natural Language Processing*.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *ArXiv*, abs/2001.04451.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark. Association for Computational Linguistics.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *ArXiv*, abs/2304.08485.

Yi Lu, Xin Zhou, Wei He, Jun Zhao, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Longheads: Multi-head attention is secretly a long context processor. *ArXiv*, abs/2402.10685.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Amirkeivan Mohtashami and Martin Jaggi. 2023. Landmark attention: Random-access infinite context length for transformers. *ArXiv*, abs/2305.16300.

OpenAI. 2023. Gpt-4 technical report.

Bo Peng, Eric Alcaide, Quentin G. Anthony, and et al. 2023. Rwkv: Reinventing rnns for the transformer era. In *Conference on Empirical Methods in Natural Language Processing*.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *ArXiv*, abs/1911.05507.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *ArXiv*, abs/2104.09864.

MosaicML NLP Team. 2023. Introducing mpt-7b: A new standard for open-source, commercially usable llms. Accessed: 2023-05-05.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, and et al. 2022. Lamda: Language models for dialog applications. *ArXiv*, abs/2201.08239.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, and et al. 2023a. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin R. Stone, and et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Neural Information Processing Systems*.

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

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. 2023b. Augmenting language models with long-term memory. *ArXiv*, abs/2306.07174.

Thomas Wolf, Lysandre Debut, Victor Sanh, and et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Yuhuai Wu, Markus Norman Rabe, DeLesley S. Hutchins, and Christian Szegedy. 2022. Memorizing transformers. *ArXiv*, abs/2203.08913.

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

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*.

Zhenyu (Allen) Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *ArXiv*, abs/2306.14048.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Haotong Zhang, Joseph Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *ArXiv*, abs/2306.05685.

Zexuan Zhong, Tao Lei, and Danqi Chen. 2022. Training language models with memory augmentation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5657–5673, Abu Dhabi, United Arab Emirates.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *ArXiv*, abs/2304.10592.

A More Experimental Results on Short Text Tasks

Besides the 50% compression ratio, we further tested the performance of Llama2-7B with different compression ratios on four short text tasks in Section 5.1. As shown in Figure 5, the model's performance decreases as the compression ratio increases. However, even when the compression ratio is set to 70%, SCA can still achieve relatively high accuracy. In particular, on IMDB and RACE datasets, SCA only needs to retain 30% of the KV cache to achieve a higher accuracy than Stream retaining 60%. This result further illustrates the effectiveness of our proposed selection strategy, which can significantly reduce information loss during compression. Furthermore, SCA can perform better than the original model in some cases. We believe this may be because SCA discards some redundant noise in the KV cache, allowing the model to make better predictions. In addition, although H_2O can achieve competitive performance with SCA, it is only suitable for short text tasks. On long text tasks (Section 5.3 and 5.4), its performance is significantly worse than our method.

B More Experimental Results on L-Eval

To verify the generality of our proposed method, we also tested it on Llama2-13B-Chat and Vicuna1.5-13B. The experimental results are shown in the Table 5. As we can see, our approach can still achieve significantly better performance than baselines. Using SCA to expand the context window of the two models by four times can indeed improve their capabilities on real long text tasks.

In addition, similar to the results on Llama2-7B-Chat and Vicuna1.5-7B, we found that H_2O performs much better on Llama2-13B-Chat than

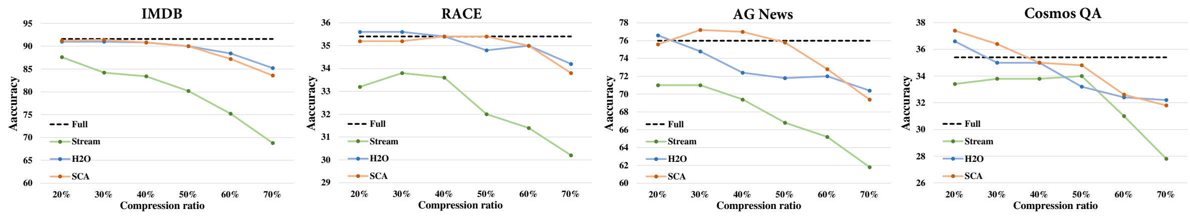


Figure 5: Performance of different methods on four short text tasks (IMDB, RACE, AG News, Cosmos QA) based on different compression ratios. The higher the compression ratio, the less KV cache is retained. **Full** represents the original model performance with full KV cache.

Model	Tokens	Coursera	GSM	QuALITY	TOFEL	CodeU	SFiction	Avg.
Llama2-13B-Chat	4k	36.1	**39.0**	41.1	62.8	1.1	52.3	38.7
+ Stream	16k	28.5	32.0	36.6	58.4	**2.2**	54.7	35.4
+ H_2O	16k	38.1	36.0	38.6	59.5	0.0	53.1	37.6
+ SCA	16k	**38.4**	**39.0**	**41.6**	**63.6**	**2.2**	**56.3**	**40.2**
Vicuna1.5-13B	4k	39.4	36.0	47.0	65.8	**3.3**	57.0	41.4
+ Stream	16k	35.2	22.0	29.2	53.2	**3.3**	58.6	33.6
+ H_2O	16k	24.6	1.0	32.2	21.6	1.1	43.0	20.6
+ SCA	16k	**43.9**	**37.0**	48.0	66.9	**3.3**	**61.7**	**43.5**
Vicuna1.5-13B-16k	16k	40.7	36.0	**54.0**	**68.4**	0.0	**61.7**	**43.5**

Table 5: Performance of different methods on closed-ended tasks of L-Eval benchmark based on Llama2-13B-Chat and Vicuna1.5-13B. Vicuna1.5-13B-16k is a version specially fine-tuned based on long text data.

on Vicuna1.5-13B. This result shows that H_2O's KV cache eviction policy has limitations and is unsuitable for some LLMs. In contrast, our method can improve the performance of the four different LLMs on long text tasks, which shows that compressing the KV cache based on its vector distribution is more versatile than other methods.

C Analysis

C.1 The Efficiency of SCA

We conduct experiments to compare the efficiency of using our method to expand the context window (Vicuna1.5-7B+SCA) with the fine-tuned long context LLM (Vicuna1.5-7B-16k) during inference. Specifically, based on the PG19 test set, we let both models generate 1000 new tokens based on the context of 15000 length. During inference, we use Flash Attention (Dao et al., 2022) and set the batch size to 1. We measure model efficiency using average latency and memory footprint.

The experimental results are shown in the Figure 6. Expanding the context window through our approach can achieve more efficient inference than the fine-tuned model regarding inference speed and memory footprint. In particular, our SCA method can reduce memory usage by 54.8% compared to the Full Attention of Vicuna1.5-7B-16k, making it possible to use LLMs for long text tasks in low computing resource scenarios.

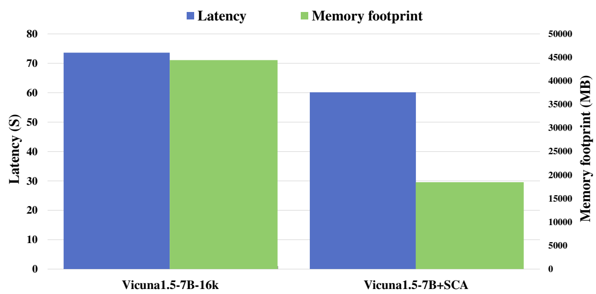


Figure 6: The average latency (s) and memory footprint (MB) of Vicuna1.5-7B-16k and Vicuna1.5-7B+SCA on the PG19 test set. We ask the models to generate 1000 new tokens based on the 15000 length context.

C.2 Redundancy at Different Layers of LLMs

In preliminary experiments (Section 3), we show the average redundancy of KV caches in all layers of LLMs but lack a fine-grained analysis of each layer. Therefore, we conducted experiments to explore the redundancy of different layers in Llama2-7B and Llama2-7B-Chat.

As shown in Figure 7, the redundancy change trends of the two LLMs are almost the same. Specifically, the redundancy difference at different layers is small for the key matrix. We believe

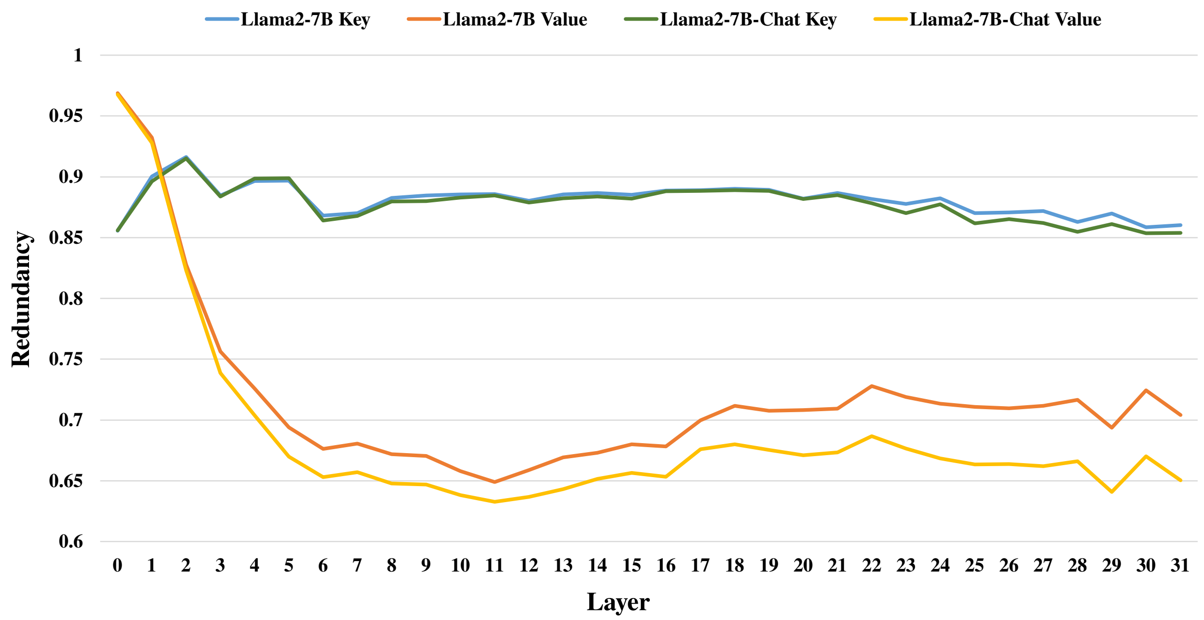


Figure 7: Redundancy of KV cache at different layers of Llama2-7B and Llama2-7B-Chat when the input context length is 3200. We add position information to the key matrix before calculating the redundancy.

Strategy	Short tasks	Long tasks
SCA	59.0	38.4
Based on Key	58.5	37.2
Based on Value	58.0	36.9
Max redundancy	46.2	30.6
Based on layer 0	57.4	37.5
Based on layer 12	57.7	36.9
Based on layer 24	58.0	37.7

Table 6: The performance of different selection strategies on short and long text tasks.

this may be because the position information added to the key matrix affects its vector distribution, making its redundancy value stable. For the value matrix, its redundancy is very large in the initial layer but decreases significantly after several layers, which suggests that LLMs can aggregate and compress information in their shallow layers (Wang et al., 2023a). After the 6th layer, its redundancy becomes stable and no longer changes drastically.

C.3 The Impact of Different Selection Strategies

To verify the superiority of the SCA selection strategy, we compared other different variants. Similar to Section 5.5, we tested the performance of different selection strategies on short and long text tasks. The different selection strategies include:

- Based on Key/Value: the selection is made based solely on the redundancy of the key or value matrix.
- Max redundancy: retain the token vector that most increases redundancy at each step. We force it to keep the initial tokens to ensure that it can perform language modeling.
- Based on layer n: selection result is calculated based on the layer n and shared with all layers

The experimental results are shown in Table 6. As we can see, SCA can achieve better performance than other variants. First, not considering the redundancy of key and value matrices in the KV cache together will lead to performance degradation. Second, retaining results with high redundancy will cause a sharp drop in accuracy because a large amount of useful information is removed during the compression process. These results further verify that our selection strategy is motivated and reasonable. Finally, the performance gap between the selection results calculated based on different layers of LLMs is small, but using the last layer has the best effect. We believe this is because the KV caches in deeper layers have a more significant impact on the prediction results of LLMs. The selection result calculated based on the last layer using the SCA algorithm can retain more information in the deep layer than calculated based on other layers, even if the relationships between vectors of the KV caches in most layers are similar.