KVSharer: Efficient Inference via Layer-Wise Dissimilar KV Cache Sharing

Anonymous ACL submission

Abstract

The development of large language models (LLMs) has significantly expanded model sizes, resulting in substantial GPU memory requirements during inference. The key and value storage of the attention map in the KV (key-value) cache accounts for more than 80% of this memory consumption. Nowadays, most existing KV cache compression methods focus on intra-layer compression within a single Transformer layer but few works consider layer-wise compression. In this paper, we propose a plug-and-play method called *KVSharer*, which shares the KV cache between layers to achieve layer-wise compression. Rather than intuitively sharing based on higher similarity, we discover a counterintuitive phenomenon: sharing dissimilar KV caches better preserves the model performance. Experiments show that *KVSharer* can reduce KV cache computation by 30%, thereby lowering memory consumption without significantly impacting model performance and it can also achieve at least 1.3 times generation acceleration. Additionally, we verify that *KVSharer* is compatible with existing intra-layer KV cache compression methods, and combining both can further save memory.

1 Introduction

Recently, large language models (LLMs) built on the Transformer (Vaswani et al., 2017) architecture have demonstrated remarkable abilities (Touvron et al., 2023; Cai et al., 2024; Yang et al., 2024a; Brown, 2020; Jiang et al., 2023). However, these impressive capabilities come with increased model size, leading to significant GPU memory costs during inference. The memory consumption of LLM during inference primarily comes from model parameters and the KV cache. The KV cache, widely used for efficient inference, stores keys and values from the attention mechanism, allowing for reuse in subsequent generation processes to improve inference speed, but also substantially increasing memory consumption. Typically, the KV cache accounts for 80% of the total memory usage during the inference phase, making it essential to optimize the KV cache to reduce memory consumption for efficiency inference (Yang et al., 2024b; Zhang et al., 2024b), particularly for long-context scenario (Bai et al., 2023; Chen et al., 2024).

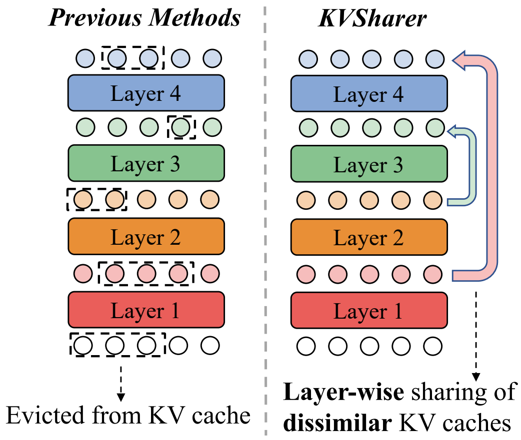
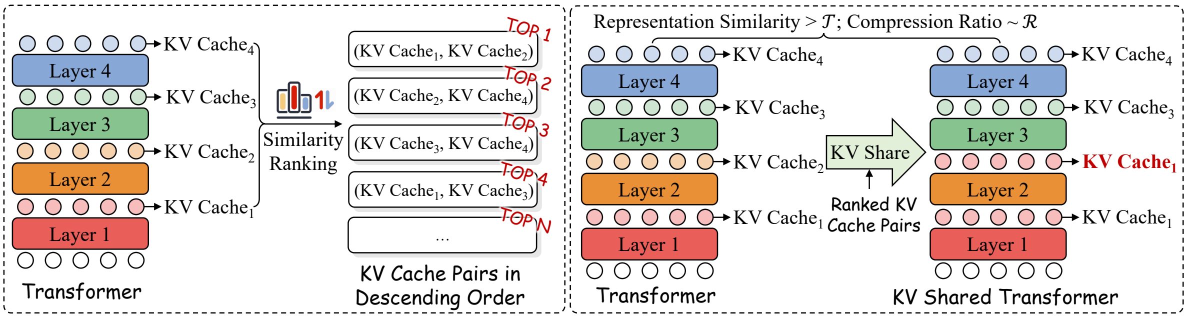


Figure 1: Previous methods primarily focus on discarding Keys and Values within layers. In contrast, we share KV caches across layers based on their dissimilarity.

Recent research has seen a proliferation of methods aimed at KV cache compression (Zandieh et al., 2024; Xu et al., 2024; Yang et al., 2024b; Zhang et al., 2024b,a; Dong et al., 2024). However, these efforts have predominantly focused on intra-layer KV cache compression within individual Transformer layers. In contrast, layer-wise KV cache compression strategies, which calculate the KV cache for only a subset of layers to minimize memory usage, remain largely unexplored. The limited existing work on layer-wise KV cache compression typically requires additional training to maintain performance (Wu and Tu, 2024; Liu et al., 2024a).

In this paper, we propose *KVSharer*, a plug-and-play method for compressing the KV cache of well-trained LLMs. Contrary to the intuitive expectation of sharing similar KV caches, our method leverages a counterintuitive observation: sharing dissim-



(a) KV Cache Similarity Ranking (b) KV Cache Sharing Strategy Searching

Figure 2: An illustration of the strategy searching process of the *KVSharer*. For a given LLM, process (a) performs inference on the calibration dataset and computes the Euclidean distance between flattened KV cache vectors from any two layers, sorting pairs in descending order. (b) KV cache pairs are sequentially replaced, ensuring the final hidden-state similarity with the original model exceeds threshold $\mathcal{T}$ until the KV cache compression ratio reaches $\mathcal{R}$.

ilar KV caches during inference causes minimal performance degradation. The paradox in this discovery lies in that previous methods for sharing parameters or activation values have always relied on replacing similar values (Dehghani et al., 2018; Reid et al., 2021; Cao et al., 2024). In contrast, we are the first to show that, in the context of KV caches, model performance can be effectively maintained by sharing dissimilar layer-wise KV caches. Leveraging this observation, *KVSharer* employs a search strategy to identify the KV cache sharing strategy across different layers during inference. *KVSharer* significantly reduces GPU memory consumption while maintaining most of the model performance. For example, it retains over 95% of the model performance while using only 70% of the original memory. As a layer-wise KV cache compression technique, *KVSharer* is compatible with existing intra-layer KV cache compression methods, offering a complementary approach to memory optimization in LLMs. *KVsharer* is also a general method and not task-specific, meaning that once a sharing strategy is found on a general calibration dataset, it can be directly applied to any downstream task. Our key contributions are summarized as:

- We observe a counterintuitive phenomenon where sharing dissimilar KV caches minimally impacts performance. Leveraging this, we propose *KVSharer*, a layer-wise KV cache sharing mechanism for efficient inference without retraining.
- Experiments with PPL (Perplexity) and downstream benchmarks show that *KVSharer* reduces GPU memory usage with minimal impact on performance while improving generation speed.
- *KVSharer* is compatible with intra-layer KV cache compression, allowing further memory reduction while preserving performance.

2 Related Work

2.1 KV cache compression

Most existing KV cache compression methods focus on intra-layer compression within a single transformer layer. Techniques like StreamingLLM (Xiao et al., 2023), H2O (Zhang et al., 2024b), Scissorhands (Liu et al., 2024b), PyramidInfer (Yang et al., 2024b), FastGen (Ge et al., 2023), and SnapKV (Li et al., 2024) achieve sparsification by discarding unimportant tokens or optimizing key-value storage within layers. However, these methods operate only within individual layers and do not address layer-wise KV cache compression.

Recently, a few approaches have explored layer-wise KV cache compression. MiniCache (Liu et al., 2024a) merges KV caches across layers to enhance throughput, LCKV (Wu and Tu, 2024) caches KVs for fewer layers to save memory, CLA (Brandon et al., 2024) introduces inter-layer attention for KV sharing, and YOCO (Sun et al., 2024) enforces KV reuse across layers. However, these methods require additional model training. In contrast, we propose the first layer-wise KV cache compression method for well-trained LLMs that requires no further training and is compatible with existing intra-layer compression techniques.

Algorithm 1 Workflow of Strategy Searching

Require: LLM $\mathcal{M}$, Target Shared KV Cache Layers $\mathcal{C}$, Calibration Dataset $\mathcal{D}$, Threshold for representation similarity $\mathcal{T}$

Ensure: Sharing Strategy $\mathcal{Z}$

$\mathcal{S} \leftarrow$ Euclidean_KV_Dis($\mathcal{M}$, $\mathcal{D}$) ▷ Perform inference on the calibration dataset $\mathcal{D}$, calculate the Euclidean distances between KV caches of all layer pairs, and record them as $\mathcal{S}$

$\mathcal{R} \leftarrow$ Descend_Rank($\mathcal{S}$) ▷ Sort KV cache layer pairs by Euclidean distance in descending order

$\mathcal{Z} \leftarrow \emptyset$ ▷ Initialize candidate sharing strategy as $\mathcal{Z}$

$\mathcal{P} \leftarrow 0$ ▷ Initialize current number of shared layers as $\mathcal{P}$

for each r in $\mathcal{R}$ **do**

  $\mathcal{Z} \leftarrow \mathcal{Z} \cup r$ ▷ Add the current pair r to the candidate set

  $\mathcal{M}_{tmp} \leftarrow$ Sharing_KV($\mathcal{M}$, $\mathcal{Z}$) ▷ Apply layer-wise KV cache sharing to $\mathcal{M}$ according to the current candidate strategy and get candidate model $\mathcal{M}_{tmp}$

  $s \leftarrow$ Avg_Cos_Sim($\mathcal{M}_{tmp}$, $\mathcal{M}$, $\mathcal{D}$) ▷ Compute the similarity of the final layer hidden-state between the two models on the calibration dataset as s

  if $s <= \mathcal{T}$ **then**

    $\mathcal{Z} \leftarrow \mathcal{Z} \setminus r$ ▷ Discard the pair r if the output similarity falls below the threshold

  else

    $\mathcal{P} \leftarrow \mathcal{P} + 1$ ▷ Find a replacement and increase the shared layers $\mathcal{P}$ by 1

    if $\mathcal{P} == \mathcal{C}$ **then**

      return $\mathcal{Z}$ ▷ Return the currently found optimal strategy

    end if

  end if

end for

return None

2.2 Attention Map & Parameter sharing

Since the introduction of Transformer-based pretrained language models (PLMs) like BERT (Devlin et al., 2018), attention map sharing and parameter sharing have been explored to enhance model efficiency. Lazyformer (Ying et al., 2021) reuses lower-layer attention maps in higher layers, improving throughput. Xiao et al. (2019) share attention weights across layers to speed up machine translation inference, while Takase and Kiyono (2021) propose rule-based parameter sharing strategies for efficiency. Shim et al. (2023) evaluate various attention map sharing methods comprehensively.

In the era of LLMs, parameter and attention map sharing have been widely adopted. Multi-Query Attention (MQA) (Shazeer, 2019) and Grouped-Query Attention (GQA) (Ainslie et al., 2023) optimize efficiency by sharing attention queries and keys within layers. Cao et al. (2024) analyze attention map and parameter similarity in LLMs, proposing sharing strategies to reduce memory usage. However, none of these works have extended to the KV cache. They all rely on replacing layers with higher parameter similarity or activation values, which aligns with intuition, whereas we replace dissimilar KV cache.

3 KVSharer

The main steps of *KVSharer* are divided into two parts. First, for a given LLM, it searches a sharing strategy, a list that specifies which layers' KV caches should be replaced by those of other specific layers. Then, during the subsequent prefill and generation processes on all the tasks, the KV caches of the relevant layers are directly replaced according to this list, enabling efficient inference.

3.1 Strategy Searching

To heuristically search for a sharing strategy, we infer on a calibration dataset, calculate Euclidean distances between KV caches of all layer pairs, and sort them in descending order. We then sequentially replace KV caches, ensuring output consistency with the original model. The process is detailed in Algorithm 1 and Figure 2.

3.1.1 Initialization

For a given LLM $\mathcal{M}$ and target shared KV cache layers $\mathcal{C}$, we use a calibration dataset $\mathcal{D}$ of plain sentences. Forward computations are performed with both the shared KV cache model and the original model, ensuring their output cosine similarity exceeds the threshold $\mathcal{T}$.

3.1.2 Searching

KV Cache Similarity Calculation & Initialization (1-4) First, we perform a forward pass using the original model $\mathcal{M}$ on the calibration dataset $\mathcal{D}$, saving the KV cache for each layer during the forward pass of each sentence. Then, we average the KV cache for each layer across all samples to obtain the average KV cache for each layer. Finally, we flatten the keys and values of the KV cache for each layer into a one-dimensional vector, and then average the keys and values separately to represent the KV cache for that layer. We then calculate the Euclidean distance between the KV cache representations of any two layers to obtain $\mathcal{S}$. We then sort $\mathcal{S}$ in descending order to get $\mathcal{R}$, since a larger Euclidean distance indicates a lower similarity. Consequently, dissimilar layer pairs are prioritized. We then set two variables, $\mathcal{Z}$ and $\mathcal{P}$, to record the candidate KV cache sharing strategy and the current number of shared layers.

Sharing Strategy Searching (5-18) Based on the values in $\mathcal{R}$, we sequentially select a pair of layers r to add to $\mathcal{Z}$ for sharing. When sharing, we replace the layer closer to the output with the one closer to the input, as the layers near the input end in LLMs are more sensitive, and modifying them could result in significant performance degradation (Cao et al., 2024; Yang et al., 2024c).

We then apply the candidate strategy $\mathcal{Z}$ by directly replacing the KV cache of one layer with another during the forward pass. Using the model with KV cache sharing and the original model, we perform inference on the calibration dataset to obtain the output representation from the last layer. We then average these representations across different sentences. If the cosine similarity between the averaged output representations of the two models exceeds the threshold $\mathcal{T}$, we retain the current pair replacement r; otherwise, we discard it. This iteration continues until the predefined number of compressed layers $\mathcal{C}$ is reached. At the end of the iteration, we obtain an optimal KV cache sharing strategy $\mathcal{Z}$ through the heuristic search.

3.2 Inference With KV cache Sharing

After deriving the sharing strategy $\mathcal{Z}$, we apply it to all inference tasks, including prefill and generation.

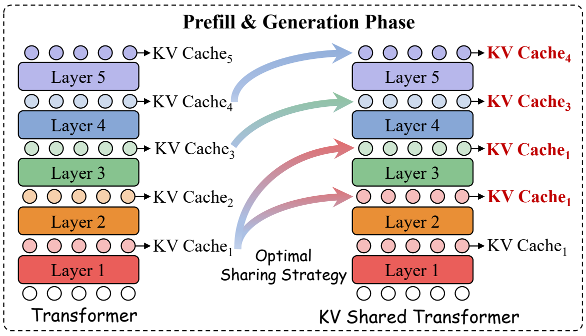


Figure 3: During the inference process of prefill and generation, according to the currently found optimal sharing strategy, *KVSharer* directly copy the result of the KV cache from a previously computed layer to the current layer during the forward computation.

As shown in Figure 3, when a layer's KV cache is shared based on $\mathcal{Z}$, it is directly copied from the corresponding layer, and subsequent computations proceed as in the original model.

4 Experiments

4.1 Models

To evaluate the effectiveness of the proposed *KVSharer*, we perform experiments on widely-used English LLMs, specifically Llama2-7B and 13B (Touvron et al., 2023). We also examine its effectiveness on bilingual LLMs, namely InternLM2-7B and 20B (Cai et al., 2024), which support both Chinese and English. For main experiments, we utilize the chat versions of Llama2-7B, InternLM2-7B, InternLM2-20B and Llama2-13B. We choose these two model series because they offer open-source models in a relatively complete range of different sizes and versions (Base or Chat). Additionally, we include experiments on the advanced Mistral-7B-Instruct-v0.3 (Jiang et al., 2023) to validate the universality of our method.

4.2 Benchmarks

To evaluate the model's broad capabilities, we use the OpenCompass framework (Contributors, 2023), focusing on five areas: Reasoning, Language, Knowledge, Examination, and Understanding, with selected benchmarks for each category. **Reasoning**: CMNLI (Xu et al., 2020), HellaSwag (HeSw) (Zellers et al., 2019), PIQA (Bisk et al., 2019). **Language**: CHID (Zheng et al., 2019), WSC (Levesque et al., 2012). **Knowledge**: CommonSenseQA (CSQA) (Talmor et al., 2018), BoolQ (Clark et al., 2019). **Examination**: MMLU (Hendrycks et al., 2021), CMMLU (Li et al., 2023b). **Understand-**

LLM	Layer	Average	Percent	Reasoning	Language	Knowledge	Examination	Understanding
Llama2-7B	32	46.55	100%	60.83	40.67	68.67	38.89	33.03
	28	52.89	113.6%	60.73	54.41	71.86	36.18	44.73
	24	45.58	97.9%	57.74	51.00	60.52	33.12	31.15
	20	38.55	82.8%	53.68	38.52	53.90	26.94	25.37
Llama2-13B	40	58.13	100%	62.90	60.56	75.67	46.67	49.67
	35	56.32	96.9%	61.31	57.33	74.56	46.16	47.77
	30	51.97	89.4%	61.35	47.38	73.66	46.03	40.51
	25	40.50	69.7%	57.46	43.75	52.39	35.39	21.97
InternLM2-7B	32	68.63	100%	62.00	71.30	76.37	64.46	69.82
	28	66.57	97.0%	60.81	66.99	75.32	60.26	69.36
	24	66.59	97.0%	62.19	65.37	74.66	62.70	68.71
	20	65.01	94.7%	61.10	63.74	74.28	62.54	65.51
InternLM2-20B	48	70.82	100%	70.66	67.34	77.88	66.26	72.33
	42	69.80	98.6%	69.02	66.84	77.39	65.94	70.75
	36	68.99	97.4%	66.82	65.55	77.41	65.45	70.76
	30	66.96	94.5%	66.58	59.62	77.41	65.07	68.48
Mistral-7B	32	64.13	100%	64.78	62.76	79.03	53.49	62.53
	28	61.56	96.0%	64.40	57.88	77.35	50.02	60.05
	24	56.67	88.4%	63.28	50.15	76.18	45.23	52.55
	20	50.53	78.8%	61.40	49.72	71.50	35.50	40.02

Table 1: The main results of our experiments. "Layer" represents the number of layers where the KV cache is actually computed. We present the average values of the model across different aspects of tasks and the average scores of all tasks as percentages relative to the full KV cache.

ing: Race-High/Middle (H/M) (Lai et al., 2017), XSum (Narayan et al., 2018), C3 (Sun et al., 2020). Evaluations use OpenCompass official scripts in zero-shot or few-shot settings, with two modes: perplexity (PPL) and generation (GEN) [1]. GEN is used for CHID and XSum, while both PPL (WSC_P) and GEN (WSC_G) are applied to WSC. Other benchmarks are evaluated in PPL mode. Scores are computed by OpenCompass, with higher values indicating better performance.

4.3 Settings

We configure the compression rates for each model at 12.5%, 25%, and 37.5% by setting the target shared layers $\mathcal{C}$, as subsequent results show that the models can maintain relatively good performance within this range. For all the models, we randomly select 30 sentences from English Wikipedia as the calibration dataset where each sentence has 64 tokens. We set $\mathcal{T}$ to 0.5 for all the models [2]. All experiments related to the PPL evaluation are conducted on a Wikipedia dataset consisting of 200 sentences, where the token length of each sentence is set to 2048. We perform experiments on a server equipped with 4 Nvidia A100 80GB GPUs.

4.4 Main Result

We conduct experiments on each dataset, calculate the average score for each aspect, the average score across all tasks, and the percentage of the average score for all tasks using *KVShare* compression relative to the average score with the full KV cache in Table 1. Detailed results can be found in Table 8 of Appendix A.1.

Llama2-7B and InternLM2-7B each have 32 layers, while Llama2-13B and InternLM2-20B have 40 and 48 layers, respectively. To evaluate performance, we apply different numbers of compressed layers to the four models at compression rates of 12.5%, 25%, and 37.5%. Additionally, we include models with full KV cache for comparison. Based on the main results, *KVSharer* exhibits minimal performance degradation compared to the full KV cache in the vast majority of tasks. Notably, when the compression rate is 25% or less, the performance remains close to 90%, and in some cases, even exceeds 95%. Furthermore, the model does not suffer significant performance drops in any specific aspect, as no individual score approaches zero. These results demonstrate that *KVSharer* effectively preserves the model's overall and task-specific performance. To present the results in Ta-

[1] https://opencompass.readthedocs.io/en/latest/get_started/faq.html

[2] When strategy searching, the similarity of the last layer's hidden state between the compressed model and the original model is usually greater than 0.8. We set a threshold of 0.5 to avoid rare cases of model output collapse. Since this situation is infrequent, we do not perform an ablation study on $\mathcal{T}$.

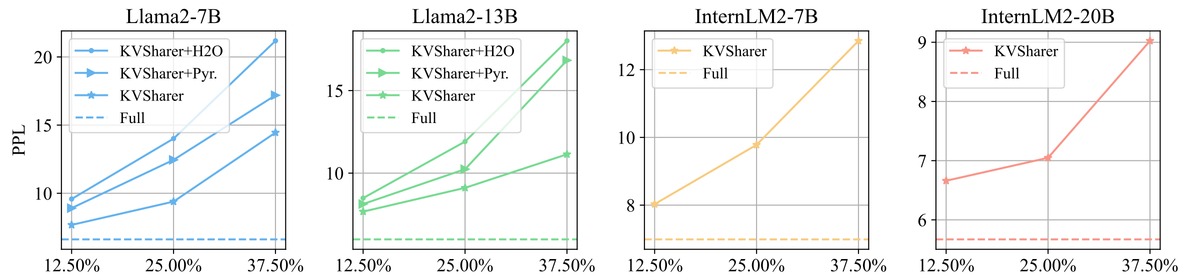


Figure 4: The model's perplexity on the Wikipedia dataset at different compression rates. "+H2O" and "+Pyr." refer to the additional use of the H2O and PyramidInfer for intra-layer compression.

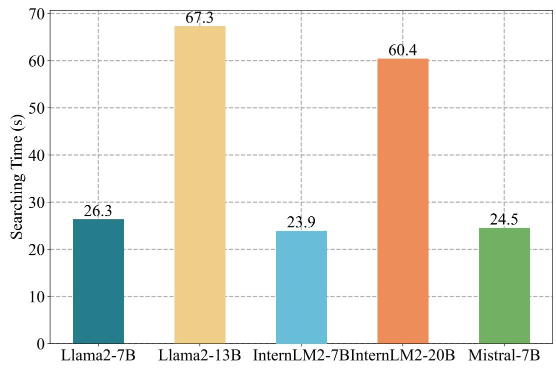


Figure 5: The searching time cost by *KVSharer* for different models. The search time is typically around 60 seconds or less.

ble 1 more intuitively, we show the average performance of each model across all tasks at different compression rates, as illustrated in Appendix A.1 Figure 7. It can also be observed that *KVSharer* can maintain the model's performance well with a compression rate of 25% or less, and even improves the average performance of the model at a 12.5% compression rate on Llama2-7B.

We also validate the larger Llama2-70B model using several benchmarks and PPL, discovering that *KVSharer* is also effective for it, maintaining most of its performance, as in Appendix A.2.

4.5 Strategy Searching Time

To evaluate the time consumption of *KVSharer*, we also test the time required for the most time-consuming part of the algorithm, Strategy Searching, as shown in Figure 5. The results show that searching for a sharing strategy on the models takes approximately one minute or less. This is expected, as Strategy Searching only requires the model to perform several inferences on a calibration dataset consisting of a few to several dozen sentences, a process that can be completed within minutes on a GPU. Note that our sharing strategy is general rather than task-specific, allowing for only one search per model, which significantly reduces the time required.

4.6 Compatibility with Intra-layer Compression

Since *KVSharer* is a layer-wise KV cache compression method, it is inherently orthogonal to intra-layer KV cache techniques. Therefore, we explore the effectiveness of combining it with existing intra-layer KV cache methods. Specifically, we combine it with H2O (Zhang et al., 2024b) and PyramidInfer (Yang et al., 2024b), which are popular intra-layer compression methods. We conduct experiments on Llama2-7B and Llama2-13B, first using *KVSharer* to identify 8 layers for shared KV cache, effectively calculating the KV cache for only 24 out of the 32 layers. Then, these two layer-wise compression methods are further applied for an additional 20% compression. The reproduction of PyramidInfer and H2O can be found in the Appendix B. We present the changes in PPL after adding H2O and PyramidInfer in Figure 4. At 12.5% and 25% *KVSharer* compression rates, both methods cause only a slight increase in PPL. The impact of PyramidInfer on PPL is lower compared to H2O, which is expected since PyramidInfer generally maintains better model performance.

Figure 4 also shows the PPL of InternLM2 and Llama2 series under different *KVSharer* compression rates. At compression rates up to 25%, the PPL remains below 15, or even 10, ensuring good generation quality. Case studies of the model's outputs are provided in Appendix C, Table 12.

4.7 Memory Cost & Inference Speed

In this section, we aim to explore the memory savings and the impact on inference speed brought by *KVSharer*. Specifically, we test the memory consumption, prefill time, and generation speed of Llama2-13B-Chat under the following settings:

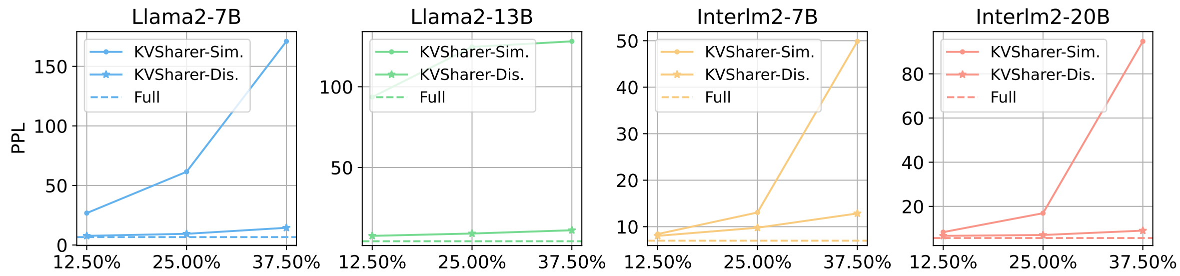


Figure 6: The model's PPL when using *KVSharer* with similarity-based sharing (+Sim.) and dissimilarity-based sharing (+Dis.). The PPL for dissimilarity-based sharing is significantly better than for similarity-based sharing.

	SeqLen.	512+32	256+2048	512+2048	1024+4096
Full	Memory	28461	36095	51639	58177
	Prefill	0.088	0.047	0.088	0.193
	Gen.	11.0	18.0	18.2	18.7
	SeqLen.	512+32	256+2048	512+2048	1024+4096
KVSharer (25%)	Memory	28257 (99%)	31403 (87%)	37049 (72%)	37231 (64%)
	Prefill	0.087	0.046	0.087	0.191
	Gen.	13.9 (×1.26)	29.8 (×1.66)	30.0 (×1.65)	28.7 (×1.53)
	SeqLen.	512+32	256+2048	512+2048	1024+4096
KVSharer (25%) + H2O	Memory	24852 (87%)	26195 (73%)	30891 (60%)	31591 (54%)
	Prefill	0.090	0.044	0.089	0.190
	Gen.	14.1 (×1.28)	29.2 (×1.62)	28.3 (×1.55)	27.1 (×1.45)
	SeqLen.	512+32	256+2048	512+2048	1024+4096
KVSharer (25%) + Pyr.	Memory	23195 (81%)	26059 (72%)	30141 (58%)	31417 (54%)
	Prefill	0.089	0.048	0.089	0.195
	Gen.	14.5 (×1.31)	33.8 (×1.88)	34.1 (×1.87)	33.4 (×1.79)

Table 2: Memory usage (MB), prefill time (s) and generation speed (tokens/s) of the Llama2-13B-Chat. "SeqLen." represents the "input length" + "maximum output length". "Gen." represents the "Generation".

Full KV cache, KVSharer with 25% compression, KVSharer with 25% compression + H2O, and KVSharer with 25% compression + PyramidInfer, across different input and maximum output lengths. We show the results in Table 2.

When sentence length is short (e.g., 512+32), *KVSharer* shows minimal memory savings, as memory is dominated by the model itself. However, as length increases, the effect becomes significant, reaching up to 30% savings at 256+2048 tokens.

In terms of speed, although there is no acceleration during the prefill phase, there is a significant acceleration during the generation phase as our results also show at least 1.2 times acceleration. When the length reaches 512+2048, it can provide over 1.6 times acceleration during the generation.

Layer	PPL	Quality	SFiction	Coursera	MultiDoc2Dial
32	6.26	35.14	8.14	11.01	8.01
28	7.88	29.29	6.91	19.19	6.53
24	10.05	25.74	5.60	10.07	5.68

Table 3: The performance of longchat-7b-v1.5-32k with KVSharer under different compression rates. PPL represents the perplexity on the Wikipedia corpus with the sentence length of 16k. Quality, SFiction, Coursera, and MultiDoc2Dial are subsets of L-EVAL.

After adding PyramidInfer and H2O, the memory usage is further reduced. Additionally, PyramidInfer further accelerates the generation speed.

4.8 Long-Context Scenario

As long-context is a key application of efficiency inference, we evaluate KVSharer using PPL and the L-EVAL benchmark (An et al., 2023). We use longchat-7b-v1.5-32k (Li et al., 2023a), a model fine-tuned from LLaMA 2 for extended context. PPL is tested on a Wikipedia corpus with the 16k sentence length, and the benchmark evaluation includes four L-EVAL subsets: Quality, SFiction, Coursera, and MultiDoc2Dial. Results are shown in Table 3. In terms of PPL, KVSharer shows a similar impact on model performance in long-context and short-context scenarios. At a 25% compression rate (Layer=24), the model maintains good PPL, demonstrating effective preservation of generation capability in the long-context scenario. On the subsets of L-EVAL, KVSharer is able to retain most of the performance and even achieves improvements on some subsets, such as Coursera. This further demonstrates that KVSharer remains effective in long-context scenarios.

Similarity \ Layer	28	24	20
Similar	8.96	15.68	424.81
Dissimilar	8.57	15.11	30.67

Table 4: The PPL results of using cosine similarity as the metric for KV cache similarity.

5 Ablation Study

5.1 Sharing by KV Cache Similarity or Dissimilarity?

We adopt a counterintuitive strategy by compressing during inference through sharing dissimilar KV caches instead of the intuitive approach of sharing similar ones. This section demonstrates experimentally that dissimilarity-based sharing outperforms similarity-based sharing. We modify Algorithm 1 to sort KV caches in ascending order of Euclidean distance while keeping other steps unchanged, and test it on the four models from the main experiment.

Figure 6 shows that similarity-based sharing results in significantly higher PPL, often doubling or more compared to dissimilarity-based sharing at the same compression rates, supporting dissimilarity-based approach in KVSharer.

5.2 Effect of Different Similarity Metrics

In Algorithm 1, we use the Euclidean distance to measure the similarity between the KV caches of any two layers. In this section, we also explore whether cosine similarity can be used as an alternative metric. Specifically, we conduct experiments using Llama-2-7B-Chat, replacing the Euclidean distance with cosine similarity while keeping other settings unchanged. We set different actual computed layers, as well as similarity-based and dissimilarity-based sharing, as shown in Table 4. The results indicate that when cosine similarity is used instead of Euclidean distance similarity, the observed pattern also remains consistent: leveraging dissimilarity for sharing performs better than using similarity for sharing. Moreover, since models with the same compression rate achieve better PPL when using Euclidean distance for sharing compared to cosine similarity, we chose to use Euclidean similarity as the metric.

5.3 Random Sharing v.s. KVSharer

KVSharer compresses KV caches by sharing dissimilar caches, prompting us to test whether random sharing can achieve similar results. We randomly replace 25% of layers' KV caches, keeping other settings unchanged, and evaluate performance on benchmarks and PPL, averaging results over three runs. As shown in Table 5, random sharing significantly increases PPL, reaching up to 50 for Llama2-13B, compared to *KVSharer*'s PPL under 10. Benchmark performance also drops by around 30%. These results show that random sharing fails to maintain performance, while *KVSharer*'s search-based approach finds a more effective strategy. However, the results reveal surprising findings: randomly sharing the KV cache does not lead to performance collapse, such as a complete failure or a PPL explosion. This suggests potential redundancy in the KV cache or a lesser-than-expected impact of self-attention keys and values on hidden-state calculations. We will investigate this further in future work [3].

LLM	Strategy	BoolQ	PIQA	HeSw	PPL
Llama2-7B	KVSharer	72.39	74.37	63.97	9.39
	Random	50.67	59.15	44.97	21.29
Llama2-13B	KVSharer	78.20	76.71	72.40	9.11
	Random	40.69	51.21	42.99	51.41
InternLM2-7B	KVSharer	80.37	79.49	73.22	9.78
	Random	63.33	61.73	58.13	13.58
InternLM2-20B	KVSharer	80.61	80.96	75.84	7.05
	Random	61.43	64.11	58.39	18.50

Table 5: Model performance using KVSharer and random sharing strategies at a 25% compression rate.

6 Conclusion

In this paper, we introduce *KVSharer*, a layer-wise KV cache sharing method designed for efficient inference. By counterintuitively sharing dissimilar KV caches, *KVSharer* reduces memory usage and boosts prefill speed during inference. Our experiments show that *KVSharer* maintains over 90% of the original performance of mainstream LLMs while reducing KV cache computation by 30%. It can also provide at least 1.3 times acceleration in generation. Additionally, *KVSharer* can be integrated with existing intra-layer KV cache compression methods to achieve even greater memory savings and faster inference. We also explore the effectiveness of the dissimilarity-based sharing approach and perform ablation studies on several components of the method.

[3]Further ablation studies on the calibration dataset, compatibility with quantized models, and applicability to Base models are provided in Appendix A.3.

Limitations

KVSharer is based on empirical observations, demonstrating that compression can be achieved by sharing dissimilar KV caches. A theoretical analysis of this counterintuitive phenomenon is left for future work.

Layer-wise KV cache compression is rare, and existing methods require training, whereas ours is training-free, leaving no directly comparable baselines. We are exploring suitable baselines for future comparisons.

In the long-context scenario, L-EVAL metrics like Rouge may be influenced by output length, potentially affecting objectivity. We plan further experiments in scenarios like the "Needle-In-A-Haystack" benchmark.

References

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*.

Chenxin 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. *Preprint*, arXiv:2307.11088.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language. *Preprint*, arXiv:1911.11641.

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*.

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. 2024. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*.

Zouying Cao, Yifei Yang, and Hai Zhao. 2024. Headwise shareable attention for large language models. *arXiv preprint arXiv:2402.11819*.

Zhi Chen, Qiguang Chen, Libo Qin, Qipeng Guo, Haijun Lv, Yicheng Zou, Wanxiang Che, Hang Yan, Kai Chen, and Dahua Lin. 2024. What are the essential factors in crafting effective long context multi-hop instruction datasets? insights and best practices. *arXiv preprint arXiv:2409.01893*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. 2024. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. *arXiv preprint arXiv:2402.09398*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

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*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

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

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. 2015. Skip-thought vectors. *arXiv preprint arXiv:1506.06726*.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E. Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023a. How long can open-source llms truly promise on context length?

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. 2023b. Cmmlu: Measuring massive multitask language understanding in chinese. *Preprint*, arXiv:2306.09212.

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*.

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

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

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *Preprint*, arXiv:1808.08745.

Machel Reid, Edison Marrese-Taylor, and Yutaka Matsuo. 2021. Subformer: Exploring weight sharing for parameter efficiency in generative transformers. *arXiv preprint arXiv:2101.00234*.

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

Kyuhong Shim, Jungwook Choi, and Wonyong Sung. 2023. Exploring attention map reuse for efficient transformer neural networks. *arXiv preprint arXiv:2301.12444*.

Kai Sun, Dian Yu, Dong Yu, and Claire Cardie. 2020. Investigating prior knowledge for challenging chinese machine reading comprehension. *Transactions of the Association for Computational Linguistics*.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*.

Sho Takase and Shun Kiyono. 2021. Lessons on parameter sharing across layers in transformers. *arXiv preprint arXiv:2104.06022*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

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*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Haoyi Wu and Kewei Tu. 2024. Layer-condensed kv cache for efficient inference of large language models. *arXiv preprint arXiv:2405.10637*.

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

Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. 2019. Sharing attention weights for fast transformer. *arXiv preprint arXiv:1906.11024*.

Liang Xu, Hai Hu, Xuanwei Zhang, Lu Li, Chenjie Cao, Yudong Li, Yechen Xu, Kai Sun, Dian Yu, Cong Yu, et al. 2020. Clue: A chinese language understanding evaluation benchmark. *arXiv preprint arXiv:2004.05986*.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. 2024. Think: Thinner key cache by query-driven pruning. *arXiv preprint arXiv:2407.21018*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

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

Yifei Yang, Zouying Cao, and Hai Zhao. 2024c. Laco: Large language model pruning via layer collapse. *arXiv preprint arXiv:2402.11187*.

Chengxuan Ying, Guolin Ke, Di He, and Tie-Yan Liu. 2021. Lazyformer: Self attention with lazy update. *arXiv preprint arXiv:2102.12702*.

Amir Zandieh, Insu Han, Vahab Mirrokni, and Amin Karbasi. 2024. Subgen: Token generation in sublinear time and memory. *arXiv preprint arXiv:2402.06082*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *Preprint*, arXiv:1905.07830.

Zhenyu Zhang, Shiwei Liu, Runjin Chen, Bhavya Kailkhura, Beidi Chen, and Atlas Wang. 2024a. Q-hitter: A better token oracle for efficient llm inference via sparse-quantized kv cache. *Proceedings of Machine Learning and Systems*, 6:381–394.

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

Chujie Zheng, Minlie Huang, and Aixin Sun. 2019. ChID: A large-scale Chinese IDiom dataset for cloze test. In *ACL*.

A Supplementary Results

A.1 Main Result

Figure 7 shows the average performance of each model across all tasks at different compression rates in the main results. Table 8 presents the detailed results of the main results.

A.2 Experiments on Large-size LLMs

Due to limitations in computational resources, we only validate the effectiveness of *KVSharer* on a subset of benchmarks and using PPL on the Llama2-70B model as shown in Table 6. We set the compression rates to 12.5% and 25%, and find that *KVSharer* effectively maintains most of the model's performance.

LLM	Layer	BoolQ	PIQA	HeSw	PPL
Llama2-70B	80	86.45	79.61	78.49	4.25
	70	84.59	76.93	77.01	5.59
	60	83.73	75.11	75.57	7.01

Table 6: The model performance achieved by applying *KVSharer* with different compression rates on Llama2-70B.

A.3 More Ablation Studies

A.3.1 Effect of Different Calibration Datasets

To investigate the impact of different calibration datasets, we replace the Wikipedia dataset with a randomly selected, equally sized subset of the BookCorpus dataset (Kiros et al., 2015). We set the compression rate to 25% and rerun the experiments, keeping all other settings unchanged.

LLM	Calibration Dataset	BoolQ	PIQA	HeSw	PPL
Llama2-7B	Wikipedia	72.39	74.37	63.97	9.39
	BookCorpus	72.01	74.10	64.05	9.15
Llama2-13B	Wikipedia	78.20	76.71	72.40	9.11
	BookCorpus	78.34	76.81	72.18	9.17
InternLM2-7B	Wikipedia	80.37	79.49	73.22	9.78
	BookCorpus	80.37	79.49	73.22	9.78
InternLM2-20B	Wikipedia	80.61	80.96	75.84	7.05
	BookCorpus	81.08	80.53	75.46	7.01

Table 7: Model performance at a 25% compression rate using Wikipedia and BookCorpus as calibration dataset. For each model, using a subset of BookCorpus as the calibration dataset has little impact on *KVSharer* compared to using a subset of the Wikipedia dataset.

The results are shown in Table 7. The findings indicate that using the two different calibration

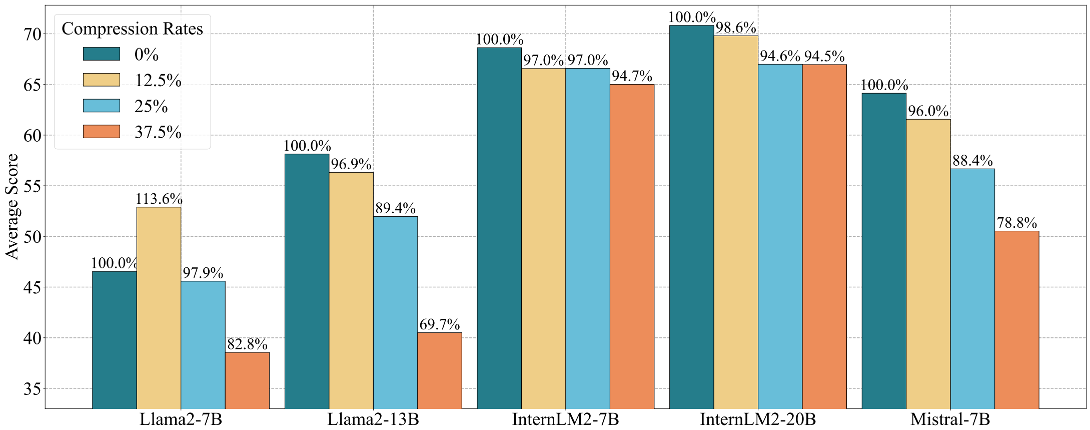


Figure 7: The percentage of the model's average score at different compression rates relative to the full KV cache model.

LLM	Layer	Reasoning			Language			Knowledge		Examination		Understanding			
		CMNLI	HeSw	PIQA	CHID	WSC_P	WSC_G	CSQA	BoolQ	MMLU	CMMLU	$Race_H$	$Race_M$	XSum	C3
Llama2 -7B	32	32.98	71.35	78.18	46.04	37.50	38.46	66.67	70.67	45.92	31.86	35.51	33.15	19.68	43.78
	28	35.11	70.37	76.71	42.08	63.46	57.69	69.62	74.10	38.63	33.74	53.95	55.92	23.24	45.81
	24	34.89	63.97	74.37	37.62	55.77	59.62	48.65	72.39	38.38	27.87	30.33	31.27	21.30	41.70
	20	34.49	55.11	71.44	32.18	52.61	30.77	48.65	59.14	28.46	25.42	22.81	23.19	16.81	38.68
Llama2 -13B	40	35.06	75.41	78.24	48.02	66.35	67.31	69.78	81.56	54.64	38.71	58.46	64.07	25.84	50.30
	35	34.27	72.84	76.82	46.04	63.46	62.50	68.71	80.40	53.87	38.44	58.18	64.14	20.30	48.44
	30	34.93	72.40	76.71	44.06	53.85	44.23	69.12	78.20	53.88	38.19	53.60	60.45	0.71	47.29
	25	34.93	64.07	73.39	33.17	58.65	39.42	39.80	64.98	40.81	29.97	25.13	25.00	0.04	37.70
Intern. -7B	32	33.09	73.30	79.60	82.18	61.54	70.19	69.53	83.21	65.98	62.94	84.19	89.00	33.56	72.55
	28	33.07	72.64	76.71	83.66	51.92	65.38	69.70	80.95	58.12	62.40	83.68	89.00	32.43	72.33
	24	33.87	73.22	79.49	81.68	45.19	69.23	68.96	80.37	63.11	62.29	83.33	88.72	30.62	72.16
	20	33.44	72.23	77.64	78.71	42.31	70.19	68.47	80.09	63.27	61.81	80.96	86.84	25.14	69.10
Intern. -20B	48	54.01	76.57	81.39	86.63	50.00	65.38	74.05	81.71	66.55	65.98	86.51	90.25	33.04	79.51
	42	50.14	76.17	80.74	85.15	50.00	65.38	73.59	81.19	66.17	65.70	86.48	90.39	26.63	79.51
	36	43.65	75.84	80.96	84.16	56.73	55.77	74.20	80.61	65.98	64.92	86.13	90.60	26.47	79.84
	30	43.98	75.89	79.87	83.66	42.31	52.88	72.73	82.08	65.32	64.82	86.11	90.67	17.48	79.67
Mistral -7B	32	32.99	78.59	82.75	48.51	67.31	72.45	74.86	83.21	62.62	44.37	75.30	79.25	34.59	60.99
	28	32.99	78.87	81.34	47.03	57.69	68.91	73.55	81.16	58.21	41.83	71.73	77.09	31.38	60.00
	24	32.99	76.07	80.79	47.52	36.54	66.39	73.55	78.81	52.61	37.85	57.66	62.19	30.36	60.00
	20	32.99	73.62	77.58	47.52	36.54	65.10	66.99	76.02	41.06	29.94	41.02	44.99	28.63	45.42

Table 8: The main results of our experiments. "Layer" represents the number of layers where the KV cache is actually computed.

datasets has almost no impact on model performance, with only minimal differences in performance across several benchmarks and PPL. For InternLM2-7B, the same sharing strategy is identified with both datasets, further indicating that *KVSharer* is not sensitive to the calibration dataset.

A.3.2 Effect of Calibration Dataset Size

We also conduct an ablation study on the size of the calibration dataset, experimenting with different

LLM Version	Llama2-7B Base		Llama2-7B Chat		Llama2-13B Base		Llama2-13B Chat		InternLM2-7B Base		InternLM2-7B Chat		InternLM2-20B Base		InternLM2-20B Chat	
Layer	32	24	32	24	40	30	40	30	32	24	32	24	48	36	48	36
BoolQ	70.67	69.27	70.67	72.39	71.50	65.63	81.56	78.20	71.28	70.40	83.21	80.37	65.44	54.04	81.71	80.61
PIQA	78.18	76.66	78.18	74.37	79.71	75.35	78.24	76.71	80.30	79.00	79.60	79.49	82.10	81.23	81.39	80.96
HeSW	71.28	69.43	71.35	63.97	74.83	67.81	75.41	72.40	73.43	72.46	73.30	73.22	75.46	74.99	76.57	75.84
PPL	5.25	11.13	6.62	9.39	4.32	7.73	5.99	9.11	7.27	10.59	6.99	9.78	5.13	7.38	5.67	7.05

Table 9: Comparison of performance on different benchmarks and PPL between Chat and Base versions of the models at the same compression rate.

LLM	Calibration Dataset Size	BoolQ	PIQA	HeSw	PPL
Llama2-7B	10	72.01	74.21	63.54	9.48
	30	72.39	74.37	63.97	9.39
	50	72.41	74.00	63.98	9.33

Table 10: Ablation study on calibration dataset size conducted on Llama2-7B under 25% compression rate.

Layer	32	28	24	20
PPL	8.61	10.40	16.68	25.89

Table 11: The PPL of GPTQ-quantized Llama2-7B-Chat with KVSharer under different compression rates.

sizes selected from the Wikipedia dataset.

As shown in Table 10, the impact of calibration dataset size on *KVSharer* is also minimal, as the model still maintains good performance under a 25% compression rate. To mitigate the potential risk of obtaining suboptimal sharing strategies due to a smaller calibration dataset size, we recommend using a larger size.

A.3.3 Compatibility with quantized model

Since quantization is also a mainstream approach for efficiency inference, we further explore whether KVSharer is compatible with quantization methods. We conducted experiments using a GPTQ-quantized (Frantar et al., 2022) Llama2-7B-Chat model combined with KVSharer. The results are shown in Table 11. We also find that KVSharer does not significantly increase the model's PPL within a 25% compression rate, further demonstrating its effectiveness.

A.3.4 Effect of KVSharer on different Model Versions

Since the models used in our main experiments are all Chat versions, we also want to explore whether *KVSharer* can be effective on the Base versions of the models. We conduct comparative experiments using the Base versions of different models, setting the compression rate at 25%, and also comparing the results with those of the full KV cache.

We show the results in the Table 9. As shown in the result, *KVSharer* also works for Base models, as it similarly maintains a minor impact on both various tasks and PPL, comparable to its effect on the Chat model. This also demonstrates that *KVSharer* has strong generalizability.

B Details of Reproduction

For H2O [4] and PyramidInfer [5], we integrate their official code with our *KVSharer*. Specifically, we sparsify the KV caches for each layer sequentially according to their methods. If a particular layer's KV cache needs to utilize the sparsified KV cache from a previous layer based on *KVSharer*, we directly place the sparsified KV cache from that previous layer into the current layer. This process is used during both the strategy searching phase and the inference phase of KV sharing in *KVSharer*.

We first tune their respective hyperparameters on the full attention model to achieve approximately 20% compression rate, and then directly apply these hyperparameters to their combination with *KVSharer*.

C Case Study

Table 12 presents the outputs of different models for the same prompt. The results indicate that, under a 25% *KVSharer* compression rate, the models can still generate fluent and meaningful responses.

[4] https://github.com/FMInference/H2O
[5] https://github.com/mutonix/pyramidinfer

Prompt
`Hello, please give a brief introduction to the Earth.`
Llama2-7B
`The planet Earth is the third planet from the sun in our solar system and the only known inhabited world. With a mass of approximately 5.97 x 1worlds, Earth has a radius of about 6,311 kilometers at the equator and an average surface temperature of around 15 degrees CelsiVCentered on the axis of rotation, Earth rotates on its axis every 24 hours, which means it takes approximately 24 hours for the planet to rotate once on its axis. The planet orbits the sun at an average distance of about 149.6 million kilometers every 365 days.`
Llama2-13B
`The Earth is the third planet from the Sun and the fifth largest of the eight planets in our solar system. It is a terrestrial planet, meaning it is made up of rock and metal, and has a solid surface that is suitable for life. The Earth is approximately 93 million miles (150 million kilometers) away from the Sun and takes 365.25 days to complete one orbit around the star. The planet is composed of three main layers: the crust, the mantle, and the core. The crust is the outermost layer and is made up of a variety of minerals such as granite and basalt. The mantle is the largest layer and is made up of hot, viscous rock that can flow over long periods of time. The core is divided into two layers: the inner core is solid and made up of iron and nickel, while the outer core is molten and also made up of iron and nickel.`
InternLM2-7B
`The earth is an oblate spheroidal planet in our solar system and third from out of five known as "terrestrial" (rocky) planets. It's not quite round; it 'wobbles' slightly on its axis which we call precession or wobble for short!` `The two main features that you can see with your own eyes are: - A day/night cycle this means one side will be lighted at all times when they're facing their star while other parts have no direct sunlight but do get some indirect illumination through scattering by gasses & dust. This also has implications about what timezones people use around here!`
InternLM2-20B
`The earth is our home and it's where we live on this planet that has been around for about four billion years or so according to scientists who study these things called geologists (people with degrees from universities). It's made up of land masses known as continents which are separated by large bodies of water such as oceans like Pacific Ocean; there are also smaller seas including the Mediterranean Sea.` `There have always existed life forms living here but they were not human beings until millions upon millions year ago when early humans first appeared out into open air after evolving over time through a natural selection process involving genetic mutations occurring randomly throughout history leading them towards becoming more advanced than their earlier ancestors had ever dreamed possible!`

Table 12: Under a 25% *KVSharer* compression rate, different models respond based on the prompts. The results show that the model's responses remain fluent and meaningful.