HSHARE: FAST LLM DECODING BY HIERARCHICAL KEY-VALUE SHARING

Anonymous authors
Paper under double-blind review

ABSTRACT

The frequent retrieval of Key-Value (KV) cache data has emerged as a significant factor contributing to the inefficiency of the inference process in large language models. Previous research has demonstrated that a small subset of critical KV cache tokens largely influences attention outcomes, leading to methods that either employ fixed sparsity patterns or dynamically select critical tokens based on the query. While dynamic sparse patterns have proven to be more effective, they introduce significant computational overhead, as critical tokens must be reselected for each self-attention computation. In this paper, we reveal substantial similarities in KV cache token criticality across neighboring queries, layers, and heads. Motivated by this insight, we propose HShare, a hierarchical KV sharing framework. HShare facilitates the sharing of critical KV cache token indices across layers, heads, and queries, which significantly reduces the computational overhead associated with query-aware dynamic token sparsity. In addition, we introduce a greedy algorithm that dynamically determines the optimal layer-level and head-level sharing configuration for the decoding phase. We evaluate the effectiveness and efficiency of HShare across various tasks using three models: LLaMA2-7b, LLaMA3-70b, and Mistral-7b. Experimental results demonstrate that HShare maintains accuracy with an additional sharing ratio of $1/8$, while delivering up to an $8.6\times$ speedup in self-attention operations and a $2.7\times$ improvement in end-to-end throughput compared with FlashAttention. The source code will be made publicly available upon publication.

1 INTRODUCTION

The development of Large Language Models (LLMs), like the GPT and LLaMA series (OpenAI, 2024; Touvron et al., 2023), has marked a major breakthrough in artificial intelligence, dramatically improving performance in natural language processing tasks such as translation (Zhu et al., 2023; Pal et al., 2024) and summarization (Zhang et al., 2023; Liu et al., 2023b). However, in long-context scenarios, such as multi-turn dialogues (Yi et al., 2024; Teng et al., 2024), document-based question answering (Abdel-Nabi et al., 2023; Rasool et al., 2024), and code completion (Yang et al., 2023; Eghbali & Pradel, 2024), LLMs face significant speed challenges with token-by-token decoding. A key factor contributing to this slowdown is the handling of the Key-Value (KV) memory cache since as the context length expands, the size of this KV cache grows correspondingly, resulting in longer access times and increased memory overhead.

Existing works have introduced several approaches to address this issue. Since it has been demonstrated that a small portion of the tokens can dominate the accuracy of token generation, many works choose to only load these critical tokens into the KV cache to reduce the inference latency while maintaining accuracy. Among them, StreamingLLM (Xiao et al., 2023) treats the initial tokens (also referred to *sink* tokens) and the recent tokens as critical tokens and performs sparsity in a fixed pattern. H2O (Zhang et al., 2024b) introduces a greedy policy that dynamically retains a balance of recent and heavy tokens. These two methods alleviate both storage and retrieval pressures through KV cache eviction. However, they tend to lose a significant amount of historical information, leading to a substantial decline in performance as their sparsity increases.

On the other hand, Quest (Tang et al., 2024) points out the criticality of tokens can change with different query tokens. To this end, Quest introduces a query-aware token sparsity algorithm, which uses the maximum and minimum values of each hidden dimension at page granularity to measure the

Table 1: Comparison of the selection of critical tokens in different token sparsity methods.

Methods	Selection mode	Efficiency	Memory cost	Accuracy
StreamingLLM (Zhang et al., 2024b)	Fixed pattern	High	Low	Low
H2O (Xiao et al., 2023)	Dynamic+Local	High	Low	Low
Quest (Tang et al., 2024)	Dynamic	Low	High	Median
DS (Yang et al., 2024)	Dynamic	Median	High	High
HShare (Ours)	Dynamic+Sharing	High	High	High

query-aware criticality. Similarly, DoubleSparse (DS) (Yang et al., 2024) proposes to select critical tokens dynamically by calculating and sorting approximate attention weights with important channels only. Although these two methods only load critical tokens, they retain all the KV cache, allowing them to enhance speed while effectively maintaining accuracy. However, for Quest and DS, the requirement evaluation for each dynamic selection introduces computational overhead, which results in a significant drawback of these approaches. Take DS as an example, with a sequence length of 2k and a batch size of 8, the process of selecting important tokens consumes nearly 50% of the total runtime in their token sparse attention. The comparison of different methods is shown in Tab. 1.

In this work, we further observe that the criticality of KV cache tokens demonstrates significant similarity across different layers, heads within the same layer, and between adjacent queries. Building on this observation and to mitigate the computational overhead associated with the dynamic selection of critical tokens mentioned above, we propose HShare, a hierarchical key-value sharing framework that operates at three levels: layers, heads, and queries. Specifically, we introduce an algorithm to design a sharing configuration for the layer and head levels based on the similarity of critical KV cache token indices. To ensure the sharing accuracy, we compute the corresponding configuration online for each batch of samples after the prefilling phase, the configuration is then applied to the entire decoding phase. This online calculation introduces only a minor increase in the prefill phase's time without incurring any additional overhead during the decode phase. At the query level, we simply share critical token indices between adjacent queries, as their proximity results in higher similarity. By retaining the full KV cache, selectively loading only the critical portions, and implementing hierarchical multi-level sharing, HShare maintains accuracy while reducing the time spent selecting critical KV tokens, significantly improving decoding latency compared to existing methods.

We use LLaMA2-7b-chat (Touvron et al., 2023), LLaMA3-70b (Dubey et al., 2024), and Mistral-7b (Jiang et al., 2023a) to evaluate the accuracy of HShare across three benchmarks: GSM8K (Cobbe et al., 2021), COQA (Reddy et al., 2019), and LongBench (Bai et al., 2023). Experimental results demonstrate that, under the same token sparsity, HShare can maintain accuracy with an additional sharing ratio of 1/8, offering competitive performance to state-of-the-art (SOTA) methods. Furthermore, we evaluate the efficiency of HShare and the results show that HShare can achieve up to 8.6× self-attention latency reduction compared with FlashAttention-2 (Dao, 2023) and 2.7× throughput improvement in end-to-end inference compared with GPT-fast (PyTorch, 2023). In summary, **the contributions of this paper are:**

- We systematically analyze the criticality of KV cache tokens across all the levels: different layers, different heads within the same layer, and adjacent queries. Our empirical findings on LLaMA2-7b-chat show that all three levels exhibit substantial similarity.
- We propose a hierarchical critical KV token indices sharing framework with a greedy algorithm to dynamically determine the sharing configuration. To the best of our knowledge, this is the first work to introduce the concept of sharing critical KV cache token indices.
- We evaluate HShare in terms of accuracy and efficiency, with results showing that it maintains model performance with a sharing ratio $1/8$ while achieving up to 8.6× reduction in self-attention latency and 2.7× improvement in inference throughput.

2 Related Work

2.1 Long-context Model

Recently, significant efforts have been made to extend the context windows of LLM, both in academia (e.g., LongChat (Li et al., 2023), Yarn-LLaMA-2 (Peng et al., 2023)) and industry (e.g., GPT-4

Turbo, which supports up to 128K tokens (OpenAI, 2024)). These extended-context LLMs excel in tasks such as multi-turn conversation comprehension and meeting summarization by providing enhanced in-context learning and improved performance on complex reasoning tasks. However, this advancement comes with notable trade-offs, including increased computational demands, higher memory usage, and greater bandwidth requirements, leading to elevated costs and longer per-token latencies (Aminabadi et al., 2022; Pan et al., 2024; Chen et al., 2024b). For example, (Pan et al., 2024) utilizes host SSD memory to alleviate the long sequence KV cache memory requirements at the expense of long latency whereas (Chen et al., 2024b) offloads the attention computation on low-cost GPU devices to support economical long-sequence LLM inference. These solutions try to tackle the Long-context model from the computing system level, however, it could be better solved from the algorithm advancements such as LLM quantization (Liu et al., 2024) and sparsification (Tang et al., 2024).

2.2 Efficient Inference of LLMs

Several techniques, such as speculative decoding (Leviathan et al., 2023; Miao et al., 2024), parameter sharing (Chen et al., 2024a), and quantization (Lin et al., 2024b;a), have been proposed to improve inference efficiency. Here we focus on accelerating the attention mechanism, which poses significant time costs due to its quadratic complexity with respect to sequence length, particularly in long-context scenarios. Decoder-only transformers, trained with masked self-attention where each token depends only on preceding tokens, enable the use of key-value activation caching (KV cache) to bypass redundant computations (Pope et al., 2023). However, the KV cache can grow significantly in size. For instance, during inference with the OPT-175B model (Zhang et al., 2022) using a batch size of 512, a prompt length of 512, and an output length of 32, the KV cache demands 1.2TB of memory for storage and communication just to generate a single token (Liu et al., 2024), which poses a grand challenge for system memory and bandwidth. To tackle the challenge, one path is to reduce the KV cache storage by quantization to 2-bit or even 1-bit. Existing works (Liu et al., 2024) report to achieve 2-bit quantization without accuracy loss and (Zhang et al., 2024a) quantizes the KV cache to 1-bit with minor accuracy loss. Another orthogonal path is to focus only on the critical tokens in the KV caches, also known as token sparsity, which we will review in Sec 2.3.

2.3 Sparse Attention

Early works such as sparse transformer (Child et al., 2019), reformer (Kitaev et al., 2020), and longformer (Beltagy et al., 2020) made efforts to reduce attention complexity through training. Recently, Jiang et al. (2023b) trained a language model to compress prompts into smaller sets of *gist* to reduce memory pressure during caching. However, such a compression strategy requires retraining the large language model and also increases the overhead during inference.

On the other hand, many works (Ribar et al., 2023; Zhang et al., 2024b; Tang et al., 2024) propose post-training sparse attention, primarily leveraging the observation of sparse attention scores, which allows focusing on important tokens without compromising performance. StreamingLLM (Xiao et al., 2023) identifies the initial and most recent tokens as critical, while Zhang et al. (2024b;a) propose to adopt the accumulated attention score as the indicator to identify the critical tokens in the KV cache. MInference (Jiang et al., 2024) introduces a dynamic sparse pattern identification algorithm for prefilling acceleration. Additionally, Quest (Tang et al., 2024) uses min and max values to assess the importance of each KV cache page, computing only the most relevant pages, while DS (Yang et al., 2024) focuses on selecting critical KV tokens by utilizing only important channels. However, these methods incur additional computational overhead for determining importance. In contrast, our work aims to minimize the need for such additional computations by sharing critical KV cache token indices across multiple levels.

3 Problem Formulation

We begin by defining the decoding self-attention process with selected critical key-value cache tokens, which we refer to as token sparsity attention. In the decoding stage, let the query matrix be denoted as $Q \in \mathbb{R}^{1\times d}$, the key matrix as $K \in \mathbb{R}^{n\times d}$ and the value matrix as $V \in \mathbb{R}^{n\times d}$, where n denotes the

sequence length. Here $K_{i,*}$ represents the i-th row of key matrix, corresponding to the i-th token in the current sequence. With these notations, we define the token sparsity attention.

Definition 3.1 (Token sparse attention, informal)**.** *Token sparsity attention only calculates the attention weights between the query matrix and the selected critical key tokens. Let N_c represent the number of KV critical tokens to be selected, and suppose we have the indices:*

$$CT = \{x_1, x_2, \ldots, x_{N_c} \mid x_i \in [0, n] \text{ and } i = 1, 2, \ldots, N_c\} \tag{1}$$

We then select the rows $\{K_{i,*} | i \in S\}$ corresponding to these indices from the key matrix to form a new key matrix $K_{CT} \in \mathbb{R}^{N_c \times d}$, also apply the same operation to the value matrix to form a new value matrix $V_{CT} \in \mathbb{R}^{N_c \times d}$. The token sparsity ratio is defined as $\frac{N_c}{n}$ and the token sparse attention is computed through the formula shown below:

$$y = \text{softmax}\left(\frac{Q \cdot K_{CT}^T}{\sqrt{d_h}}\right) \cdot V_{CT} \tag{2}$$

Normally, each attention block independently evaluates and selects its critical KV cache tokens, resulting in a corresponding set of indices, denoted as CT. We use the number of overlapping elements between two different sets to evaluate the similarity between them. Then the similarity between two critical KV cache token indices CT_A and CT_B can be formally written as:

$$sim(CT_A, CT_B) = \frac{|CT_A \cap CT_B|}{max(|CT_A|, |CT_B|)}, \tag{3}$$

here $|\cdot|$ represents the cardinality of a set and normally $|CT_A| = |CT_B| = N_c$. Next, we define Key-Value sharing.

Definition 3.2 (Key-Value sharing)**.** *Assuming that the critical KV cache token indices for attention block A are denoted as CT_A, if another attention block B directly reuses the indices from block A, represented as $CT_B \leftarrow CT_A$, we refer this to as key-value sharing between blocks B and A. In this case, CT_A is utilized by attention block B to construct K_{CT} and V_{CT}.*

4 METHOD

In this section, we first present the motivation of HShare, and then we introduce HShare in details.

4.1 MOTIVATION

Previous works have pointed out not all KV tokens hold equal significance, with a limited subset, known as critical tokens, contributing most to the attention output, and these critical tokens are found to be query-aware. In this paper, we further present three observations, with insights shown in Fig. 1.

Similar critical tokens between adjacent queries: From Fig. 1(a)-(d), we observe that within individual attention heads, the distribution of critical KV cache tokens (those associated with larger values in the attention matrix) gradually shifts with changes in the queries. However, the critical tokens across adjacent queries remain largely consistent. This observation aligns with intuition, as minor variations in query length do not significantly alter the overall sequence length, suggesting that the attention distribution remains stable. The similarity between adjacent queries presents an opportunity to reduce computational overhead by reusing the same critical KV cache tokens for every k consecutive query.

Similar sparse-patterns across different heads and layers: The distribution of critical tokens results in different sparse patterns for attention heads. However, as illustrated in Fig. 1(a)-(d), we can also see that some sparse patterns are consistently observed across various attention heads. For example, the sparse pattern of head 12 in layer 8 is very similar to that of head 22 in the same layer. Additionally, head 9 and head 16 in layer 9 also share a similar sparse pattern, indicating that the similarity between heads exists not only within the same layer but also across different layers. This uniformity suggests that a sparse pattern derived from one head can be effectively applied to others, enabling a shared strategy for selecting critical KV cache tokens. This approach can further improve the speed and efficiency of the attention mechanism.

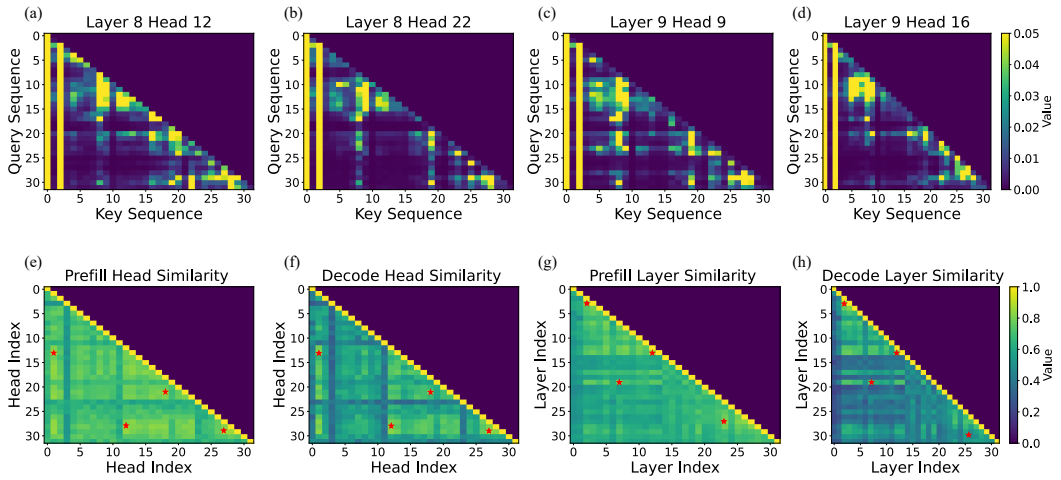

Figure 1: (a)-(d): Illustrations of self-attention matrices from a specific head in a particular layer of the LLaMA2-7b-chat model, corresponding to similar self-attention patterns observed across different layers and heads. (e)-(h) Similarity matrices of critical token indices across different heads and layers during the prefill and decode stages. The element in the i-th row and j-th column represents the similarity (Eq. 3) between the i-th head (layer) and the j-th head (layer). The red stars represent the top-k largest values in the matrix.

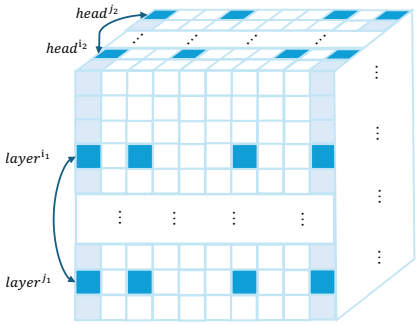

Figure 2: Share the same critical token indices between layers and heads.

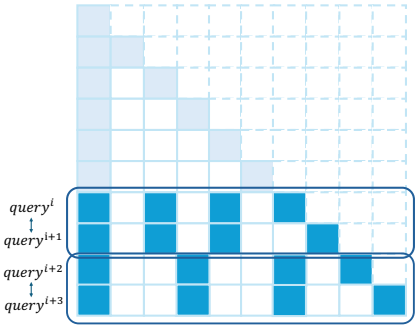

Figure 3: Share the same critical token indices between adjacent queries.

Consistency relationships between prefill and decoding phases: Fig. 1(e) and Fig. 1(f) present the head similarity matrices of layer 8 in terms of prefill stage and decoding stage, respectively. Fig. 1(g) and Fig. 1(h) present the layer similarity matrices of the whole network in terms of the prefill stage and decoding stage, respectively. We observe that the similarity matrix between prefill and decoding phases exhibits a remarkable consistency for both head-level and layer-level. This continuity underscores the potential that applying the hierarchical sharing strategy computed from the prefill phase to the decoding phase.

Motivated by the above observations, we propose HShare, a hierarchical sharing framework that shares the same critical KV cache tokens across queries, heads, and layers. HShare not only takes advantage of dynamic sparsity but also significantly reduces the computational overhead associated with critical tokens through an effective sharing mechanism.

4.2 Hierarchical Key-Value Sharing Framework

Inspired by recent works (Zhang et al., 2024b; Xiao et al., 2023; Yang et al., 2024), in this paper, we select critical tokens from three perspectives: initial tokens (also referred to as sink tokens), the most recent window, and significant tokens in the middle. The dark blue positions in Fig.2 and Fig.3 represent the indices of these critical tokens. Given that the decoding stage is autoregressive, each

decoding step requires passing through all L layers of the attention blocks. In the case of multi-head attention and group query attention, each attention block involves multiple parallel attention head H computations. Consequently, to generate n tokens, a total of $L \times H \times n$ attention computations are necessary, resulting in $L \times H \times n$ re-selections of critical tokens.

To reduce the overall time spent on evaluating and selecting critical tokens and further accelerate the decoding, we introduce a hierarchical key-value sharing framework. This framework is designed to facilitate key-value sharing from three perspectives: across layers, heads, and queries. Specifically, denote the attention computation of head h in layer l of the query q as $A_{l,h,q}$, then the key-value sharing between layers l_1 and l_2 can be expressed as: $CT_{A_{l_2,h,q}} \leftarrow CT_{A_{l_1,h,q}}$.

Similarly, the key-value sharing between heads h_1, h_2 and queries q_1, q_2 can be expressed as: $CT_{A_{l,h_1,q}} \leftarrow CT_{A_{l,h_2,q}}, \quad CT_{A_{l,h,q_1}} \leftarrow CT_{A_{l,h,q_2}}$.

It should be noted that since there is a temporal dependency between the attention computations of layers and queries, it is necessary to satisfy $l_1 < l_2$ and $q_1 < q_2$ here.

Similar to the token sparsity ratio, we define the sharing ratio as a value less than or equal to 1, where a smaller value indicates a higher degree of sharing and greater computational savings. Let the total number of layers and heads be denoted as n_l and n_h, respectively. Suppose k_l layers and k_h heads share critical KV token indices with other layers or heads. The sharing ratios for layers and heads are then defined as $ratio_l = 1 - \frac{k_l}{n_l}$ and $ratio_h = 1 - \frac{k_h}{n_h}$, respectively. For query-level sharing, let the total number of decoding tokens (queries) be n_q, and suppose k_q queries bypass the selection of critical token indices by sharing key-value indices with other queries. The query-level sharing ratio is defined as $ratio_q = 1 - \frac{k_q}{n_q}$. Since the sharing across layers, heads, and queries is mutually independent, the total sharing ratio is given by:

$$\textit{sharing ratio} = (1 - \frac{k_l}{n_l}) \times (1 - \frac{k_h}{n_h}) \times (1 - \frac{k_q}{n_q}) \tag{4}$$

4.3 Algorithm for Sharing Configuration Design

Algorithm 1 Layer(Head) Sharing Algorithm

Input: layer(head) number $n_l(n_h)$, critical token indices matrix List $L = \{M_1, M_2, ...M_{n_l(n_h)}\}$, share number $k_l(k_h)$
Initialize similarity matrix $S \leftarrow zeros(n, n)$
for $i = 1$ to $n_l(n_h)$ **do**
  $set_i \leftarrow$ set(M_i.flatten())
  for $j = 1$ to $i - 1$ **do**
    $set_j \leftarrow$ set(M_j.flatten())
    overlap $\leftarrow |set_i \cap set_j|$
    $S_{ij} \leftarrow$ overlap / M_i.numel()
  end for
end for
for $i = 1$ to $n_l(n_h)$ **do**
  Initialize sharing config $C[i] \leftarrow i$
end for
for $i = 1$ to $k_l(k_h)$ **do**
  row_index, column_index $\leftarrow$ argmax(S)
  S[row_index, :]$\leftarrow 0$
  S[:, row_index] $\leftarrow 0$
  C[row_index] $\leftarrow$ column_index
end for
Output: Sharing config C of layer(head).

For layer-level and head-level sharing, as demonstrated in Sec. 4.1, the similarity of the critical KV cache token indices shows consistency across the different queries. Therefore, we can design a sharing configuration by calculating the similarity of the critical token indices between layers and heads after prefilling, and then apply this configuration to the entire decoding phase.

Here, we use the layer-level as an example for a detailed explanation. After each prefill step, we first obtain the critical KV cache token indices matrix list for all layers $L = M_1, M_2, \ldots, M_{n_l}$, where n_l represents the number of layers. Next, we compute the pairwise similarities between the critical KV cache token indices of all layers, producing a similarity matrix S_l. Based on the desired sharing ratio, we then perform key-value sharing between the most similar layers. Specifically, if the layer-level sharing ratio is β, meaning that $k_l = (1 - \beta) \times n_l$ layers should share critical KV token indices, we iteratively select the pair (i, j) with the highest similarity (where $j < i$), allowing the i-th layer to reuse the indices of the j-th layer. Since layer i now reuses the critical token indices of another layer and does not compute its own, we remove the i-th row and i-th column from the similarity matrix.

Similarly, for all heads within each layer, we apply the same method to determine the sharing configuration. Note that each layer has its own head-level sharing configuration. Algo. 1 outlines the process of calculating the similarity matrix and designing the layer (or head) sharing configuration. Formally, the output sharing configuration C represents:

$$c_i = \begin{cases} j, & \text{if the } i\text{-th layer (or the } i\text{-th head) shares with the } j\text{-th layer (or the } j\text{-th head),} \\ i, & \text{if the } i\text{-th layer needs to compute critical tokens separately.} \end{cases} \tag{5}$$

Since the sharing configuration can differ across samples, we compute it dynamically for each batch of samples.

Query-level. As discussed in Sec. 4.1, we observe that attention weights between adjacent query tokens tend to exhibit significantly higher similarity compared to those between distant query tokens. Specifically, for query i and query $i+1$, the importance rankings of the tokens from token 0 to token $i-1$ are largely similar. Based on this observation, we group every a adjacent query to share a single critical token set, resulting in a sharing ratio of $\frac{1}{a}$ between queries. Fig.3 illustrates the sharing mechanism between two adjacent queries.

5 Experiment

5.1 Accuracy Evaluation

5.1.1 Setup

We evaluate HShare on GSM8K (Cobbe et al., 2021), COQA (Reddy et al., 2019), and six datasets in LongBench (Bai et al., 2023): **LCC** (Guo et al., 2023), **RepoBench-P** (Liu et al., 2023a), **TriviaQA** (Joshi et al., 2017), **Qasper** (Dasigi et al., 2021), 2WikiMultihopQA(**2WikiMQA**) (Ho et al., 2020) and **GovReport** (Huang et al., 2021). For our evaluation, we select three widely-used models: LLaMA2-7b-chat (Touvron et al., 2023), LLaMA3-70b (Gliwa et al., 2019), and Mistral-7b (Jiang et al., 2023a), representing a range of architectures from *multi-head attention* (MHA) to *group query attention* (GQA), and from *dense* to *mixture of experts* (MoE). As baselines, we include four state-of-the-art methods: two KV cache eviction algorithms **StreamingLLM** (Xiao et al., 2023), **H2O** (Zhang et al., 2024b) and two query-aware token sparsity algorithms **Quest** (Tang et al., 2024), **DS** (Yang et al., 2024). It should be noted that some baselines skip sparsity in certain layers or during the prefill stage. To ensure a fair comparison, we apply token-sparse attention to all layers across both the decode and prefill stages. Here, we also include the vanilla attention as **Original**, which serves as the baseline representing lossless results, without applying any sparsity in either the prefill stage or the decoding stage.

5.1.2 Results on GSM8K, COQA

Datasets and Metrics. We use the LM-Eval framework to conduct zero-shot inference on two generation tasks: GSM8K and COQA. GSM8K consists of around 8,000 elementary school math problems and COQA is designed for evaluating dialogue-based question-answering systems. The average context length for these two datasets is approximately 500 and 2k respectively. Here we report the GSM8K with flexible exact-match accuracy and strict exact-match accuracy, while COQA with em score.

Inference Details. For a fair comparison, all methods select $N_c = 128$ critical KV cache tokens and do token sparse attention, with a token sparsity ratio of approximately $1/4$ and $1/16$ for the two datasets respectively (since GSM8K consists of math problems, we select a higher sparsity level, whereas COQA involves story-based question-answering, so we choose a lower sparsity level.) For HShare, we select critical tokens from three aspects, with $x = 8$ sink tokens, $y = 32$ recent tokens, and $z = 88$ critical tokens in the middle. Following the approach in Yang et al. (2024), we load the heavy channel of the KV cache in 4-bit precision to compute the approximate attention weights, then sort them to select the top-z tokens in the middle. We evaluate the effectiveness of our proposed HShare using LLaMA2-7b-chat, LLaMA3-70b, and Mistral-7b, selecting two sharing ratios **3/4-3/4-1/2** and **1/2-1/2-1/2** of HShare for comparison against other methods. Here **a-b-c** means HShare with layer-sharing ratio a, head-sharing ratio b, and query-sharing ratio c.

Table 2: **Evaluation of different methods on GSM8K and COQA** and the best result (exclude origin) in each column is highlighted in bold. Complexity* refers to the theoretical time complexity for each method to select critical KV cache tokens, where $\mathcal{O}(1)$ denotes constant time complexity, and T represents the theoretical computation time for a dense attention mechanism.

Model	Architecture	Method	GSM8K(flexible/strict)↑	COQA ↑	Complexity* ↓
LLaMA2-7b-chat	Dense/MHA	Original	0.2297/0.2297	0.5997	-
		StreamingLLM	0.0485/0.0000	0.2515	0
		H2O	0.0558/0.0108	0.3615	$\mathcal{O}(1)$
		Quest	0.0371/0.0364	0.5513	$0.125T$
		DS	**0.1630/0.1622**	**0.6270**	$0.0625T$
		Ours (3/4-3/4-1/2)	0.1554/0.1456	0.6013	$0.018T$
		Ours (1/2-1/2-1/2)	0.1319/0.0743	0.5960	$0.008T$
LLaMA3-70b	Dense/GQA	Original	0.8067/0.8052	0.7085	-
		StreamingLLM	0.3897/0.0311	0.3473	0
		H2O	0.4329/0.3288	0.6100	$\mathcal{O}(1)$
		Quest	0.4708/0.4602	0.6900	$0.125T$
		DS	0.7233/0.7195	0.7012	$0.0625T$
		Ours (3/4-3/4-1/2)	**0.7460/0.7445**	**0.7075**	$0.018T$
		Ours (1/2-1/2-1/2)	0.7301/0.7225	0.6912	$0.008T$
Mistral-7b	MoE/GQA	Original	0.3821/0.3813	0.6758	-
		StreamingLLM	0.0849/0.0068	0.2905	0
		H2O	0.0902/0.0159	0.4225	$\mathcal{O}(1)$
		Quest	0.1302/0.0569	0.6170	$0.125T$
		DS	0.3093/0.3063	**0.6687**	$0.0625T$
		Ours (3/4-3/4-1/2)	**0.3101/0.3010**	0.6538	$0.018T$
		Ours (1/2-1/2-1/2)	0.2775/0.2707	0.6313	$0.008T$

Table 3: **Evaluation of different methods on six datasets of Longbench** and the best result in each column is highlighted in bold.

Method	LCC	RepoBench-P	TriviaQA	Qasper	2WikiMQA	GovReport	Average
Original	58.26	52.14	83.09	21.88	31.18	26.55	45.52
StreamingLLM	52.41	48.12	53.76	14.01	27.07	21.39	36.13
H2O	57.47	47.67	60.81	14.13	25.99	21.51	37.93
Quest	53.99	44.42	81.49	17.14	28.17	25.95	41.86
DS	**57.75**	48.78	83.46	21.94	28.77	**26.53**	44.54
Ours (7/8-3/4-1/2)	57.20	**50.22**	**84.03**	21.14	29.70	25.87	44.69
Ours (3/4-3/4-1/2)	56.29	49.67	83.92	**22.13**	**30.67**	25.76	**44.74**
Ours (1/2-1/2-1/2)	55.89	48.88	83.68	21.39	29.04	23.88	43.79

Results The results on GSM8K and COQA, shown in Tab. 2, indicate that HShare significantly outperforms other KV cache eviction methods (StreamingLLM and H2O). In addition, when compared to dynamic critical token selection approaches (Quest and DS), HShare incurs only minimal accuracy loss and even achieves higher accuracy on LLaMA3-70b. Overall, HShare provides the optimal balance between efficiency and accuracy.

5.1.3 Results on LongBench

Datasets and Metrics. We use LongBench to evaluate the performance of the proposed HShare across multiple long context benchmarks, we choose six datasets as representatives, including code completion: LCC and RepoBench-P; few-shot learning: TriviaQA; single-document QA: Qasper; multi-document QA: 2WikiMQA; summarization: GovReport. Here we report LCC and RepoBench-P with similarity score, TriviaQA, Qasper and 2WikiMQA with F1 score, as well as GovReport with rouge score.

Inference Details. All methods select $N_c = 512$ critical KV cache tokens, with a token sparsity ratio of approximately $1/8$. Similarly, we select critical tokens from three aspects: $x = 16$ sink tokens, $y = 64$ recent tokens, and $z = 432$ critical tokens in the middle. Using LLaMA2-7b-chat, we evaluate three sharing ratios of HShare: **7/8-3/4-1/2**, **3/4-3/4-1/2** and **1/2-1/2-1/2**.

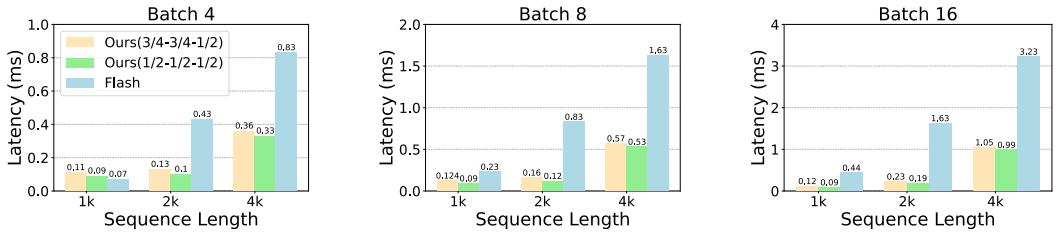

Figure 4: Latency(↓) of different methods across various batch sizes and sequence lengths.

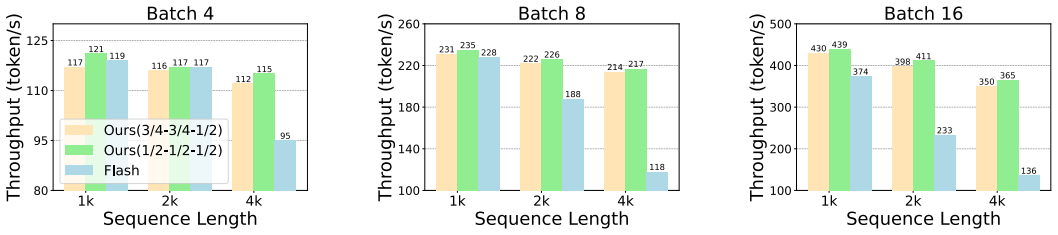

Figure 5: Throughput (↑) of different methods across various batch sizes and sequence lengths.

Results. Here, we only present the experimental results on six representative datasets in Tab. 3. We can learn that HShare outperforms StreamingLLM, H2O, and Quest by a notable margin, which demonstrates that HShare has the ability to maintain accuracy in long-context scenarios. Additionally, we can also find that HShare could achieve a competitive performance compared with DS under different sharing ratios. It is important to note that the sharing ratio of HShare serves as a hyperparameter that can influence the balance between effectiveness and efficiency. More detailed results and discussion can be found in Appendix A.1 and Appendix A.2.

5.2 Efficiency Evaluation

5.2.1 Setup

All experiments are conducted on a machine with Xeon(R) Platinum 8336C CPU, one A100 GPU, and 128G RAM.

Inference Details. Following Yang et al. (2024), we utilized PyTorch to approximate attention, selecting the top-z tokens from the middle and further incorporating x initial indices and y recent indices to form the complete set of critical token indices. The kernel for head-level sharing and the attention over critical KV tokens are designed using OpenAI Triton. Meanwhile, since HShare designs key value sharing between different layers and queries, we require an additional cache to store the critical token indices that need to be shared. Then for attention modules where the computation of critical token indices is bypassed due to sharing, we directly load the corresponding part from the cache. It is important to note that the additional storage required here is minimal, as we only need to store integers with a complexity of $\mathcal{O}(N_c)$. As for end-to-end testing, our implementation is based on GPT-fast (PyTorch, 2023), with the full attention module being replaced by our token sparsity self-attention module.

Baseline. To evaluate self-attention operator speedup, we use FlashAttention2 (Dao, 2023) as our baseline, which is an optimized attention mechanism designed to improve the speed and efficiency of attention computation and ranks among the fastest attention mechanisms. For the evaluation of end-to-end inference speedup, we take GPT-fast (PyTorch, 2023) as our baseline, which is acknowledged as the SOTA implementation for LLaMA models on the A100 GPU. Further comparison with other token sparsity algorithms can be referred to Appendix A.3.

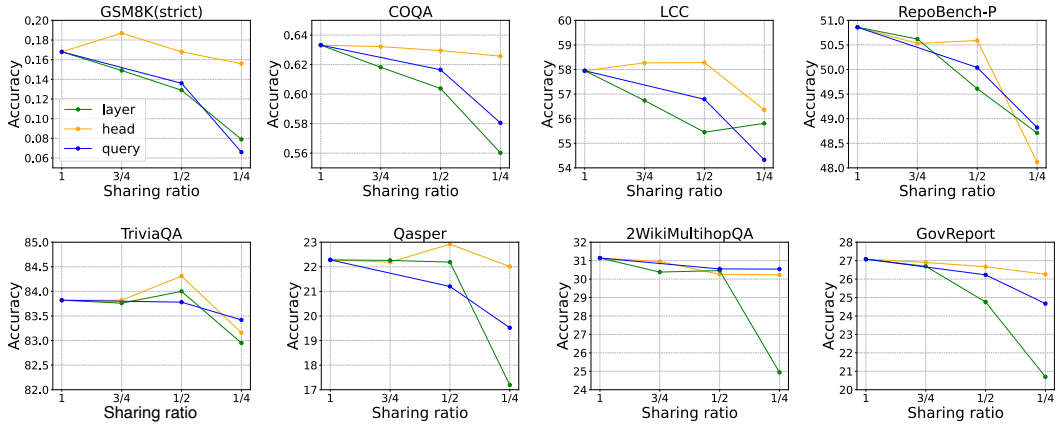

Figure 6: Results of HShare with different sharing ratios on eight datasets.

5.2.2 Self-Attention Operator Speedup

We conduct the self-attention latency evaluation on a single A100 GPU with batch sizes ranging from 4 to 16 and sequence lengths from 1k to 4k, with a token sparsity ratio of $1/8$. We simulate the hierarchical sharing framework and average the latency over 1000 times self-attention computations. The results are shown in Fig. 4, which demonstrate that our method can significantly improve the average latency of self-attention compared with FlashAttention-2. Especially when the batch size is 16 and the sequence length is 2k, HShare can achieve up to an $8.6\times$ reduction in self-attention latency. However, we observe that in the case when batch size is 4 and the sequence length is 1k, our token-sparsity self-attention module performs slightly worse than the baseline. This may be because, in smaller workloads, GPU underutilization prevents token sparsity from improving speed and selecting critical token indices adds overhead instead.

5.2.3 End-to-End Inference Speedup

Similarly, we conduct end-to-end inference evaluation with the same batch sizes and sequence lengths, and Fig. 5 reports the corresponding results. We find that HShare consistently outperforms GPT-fast and achieves up to $2.7\times$ throughput acceleration, especially with larger batch sizes and longer sequence lengths.

5.3 Ablation Study

HShare with different sharing ratios. We conduct ablation studies under different sharing ratios on eight datasets and the results are shown in Fig. 6. Specifically, for layer-level and head-level sharing, we selected sharing ratios of $3/4$, $1/2$, and $1/4$. For query-level sharing, we group queries in sets of 2 and 4, corresponding to sharing ratios of $1/2$ and $1/4$, respectively. The results indicate that for all three levels, a smaller sharing ratio tends to result in greater accuracy loss on average. However, on certain datasets like TriviaQA and GSM8K, sharing a subset of heads leads to improved performance. Additionally, compared to head-level sharing, we observe that layer-level and query-level sharing have a greater negative impact on accuracy. Overall, the sharing ratios **3/4-3/4-1/2** and **1/2-1/2-1/2** emerge as more reasonable options. More ablation studies can be found in Appendix A.4.

6 Conclusion and Discussion

In this paper, we first systematically analyze and reveal the similarity of critical KV cache tokens across layers, heads, and query levels. Then we introduce HShare, a hierarchical framework for sharing critical KV cache token indices at all three levels to reduce the overhead associated with selecting critical KV tokens. Additionally, we propose a greedy selection algorithm for efficient sharing at the layer and head levels. Extensive evaluations show that HShare preserves model

accuracy with an additional sharing ratio of $1/8$, while achieving up to $8.6\times$ speedup in self-attention operations and a $2.7\times$ increase in end-to-end throughput.

REFERENCES

Heba Abdel-Nabi, Arafat Awajan, and Mostafa Z Ali. Deep learning-based question answering: a survey. *Knowledge and Information Systems*, 65(4):1399–1485, 2023.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6077–6086, 2018.

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

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

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. *Proceedings of Machine Learning and Systems*, 6:1–13, 2024a.

Shaoyuan Chen, Yutong Lin, Mingxing Zhang, and Yongwei Wu. Efficient and economic large language model inference with attention offloading. *arXiv preprint arXiv:2405.01814*, 2024b.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

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

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

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

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

Aryaz Eghbali and Michael Pradel. De-hallucinator: Iterative grounding for llm-based code completion. *arXiv preprint arXiv:2401.01701*, 2024.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*, 2019.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, pp. 12098–12107. PMLR, 2023.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*, 2021.

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. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023a.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023b.

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

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can context length of open-source llms truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6: 87–100, 2024a.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *arXiv preprint arXiv:2405.04532*, 2024b.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023a.

Yixin Liu, Kejian Shi, Katherine S He, Longtian Ye, Alexander R Fabbri, Pengfei Liu, Dragomir Radev, and Arman Cohan. On learning to summarize with large language models as references. *arXiv preprint arXiv:2305.14239*, 2023b.

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

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024.

OpenAI. Introducing gpt-4o: our fastest and most affordable flagship mode, 2024. URL https://platform.openai.com/docs/models. Last accessed 28 September 2024.

Proyag Pal, Alexandra Birch-Mayne, and Kenneth Heafield. Document-level machine translation with large-scale public parallel corpora. In *The 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 13185–13197. Association for Computational Linguistics (ACL), 2024.

Xiurui Pan, Endian Li, Qiao Li, Shengwen Liang, Yizhou Shan, Ke Zhou, Yingwei Luo, Xiaolin Wang, and Jie Zhang. Instinfer: In-storage attention offloading for cost-effective long-context llm inference. *arXiv preprint arXiv:2409.04992*, 2024.

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

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

PyTorch. Accelerating generative ai with pytorch 2.0. https://pytorch.org/blog/accelerating-generative-ai-2/, May 2023.

Zafaryab Rasool, Stefanus Kurniawan, Sherwin Balugo, Scott Barnett, Rajesh Vasa, Courtney Chesser, Benjamin M Hampstead, Sylvie Belleville, Kon Mouzakis, and Alex Bahar-Fuchs. Evaluating llms on document-based qa: Exact answer selection and numerical extraction using cogtale dataset. *Natural Language Processing Journal*, pp. 100083, 2024.

Siva Reddy, Danqi Chen, and Christopher D Manning. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference. *arXiv preprint arXiv:2312.04985*, 2023.

Jaewon Son, Jaehun Park, and Kwangsu Kim. Csta: Cnn-based spatiotemporal attention for video summarization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18847–18856, 2024.

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

Zeyu Teng, Yong Song, Xiaozhou Ye, and Ye Ouyang. Fine-tuning llms for multi-turn dialogues: Optimizing cross-entropy loss with kl divergence for all rounds of responses. In *Proceedings of the 2024 16th International Conference on Machine Learning and Computing*, pp. 128–133, 2024.

Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279, 2021.

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

Guang Yang, Yu Zhou, Xiang Chen, Xiangyu Zhang, Tingting Han, and Taolue Chen. Exploitgen: Template-augmented exploit code generation based on codebert. *Journal of Systems and Software*, 197:111577, 2023.

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

Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. A survey on recent advances in llm-based multi-turn dialogue systems. *arXiv preprint arXiv:2402.18013*, 2024.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

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

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

Zixuan Zhang, Heba Elfardy, Markus Dreyer, Kevin Small, Heng Ji, and Mohit Bansal. Enhancing multi-document summarization with cross-document graph-based information extraction. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 1696–1707, 2023.

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. Multilingual machine translation with large language models: Empirical results and analysis. *arXiv preprint arXiv:2304.04675*, 2023.

A MORE EXPERIMENTS

A.1 MORE RESULTS ON LONGBENCH

The results across the 16 commonly used English datasets from Longbench are shown in Tab. 4. It can be seen that HShare achieves the highest scores in the most sub-datasets. On average, HShare outperforms StreamingLLM, H2O, and Quest, and achieves competitive results with DS under a sharing ratio of 7/8-3/4-1/2.

Table 4: **Evaluation of different methods on sixteen datasets in Longbench** and the best result in each row (exclude original) is highlighted in bold.

Dataset \ Method	Original	StreamingLLM	H2O	Quest	DS	Ours (7/8-3/4-1/2)	Ours (3/4-3/4-1/2)	Ours (1/2-1/2-1/2)
MultiNews	26.22	22.93	22.24	25.74	25.98	**26.30**	25.86	24.83
Musique	8.65	4.56	6.32	6.65	7.16	**8.15**	7.63	7.41
HotpotQA	27.72	20.77	25.62	23.30	24.83	25.13	24.98	**25.81**
Qasper	21.88	14.01	14.13	17.14	21.94	21.14	**22.13**	21.39
2WikiMQA	31.18	27.07	25.99	28.17	28.77	29.70	**30.67**	29.04
Repobench-P	52.14	48.12	47.67	44.42	48.78	50.22	**49.67**	48.88
TriviaQA	83.09	53.76	60.81	81.49	83.46	**84.03**	83.92	83.68
Trec	64.50	40.50	44.00	61.00	**61.50**	59.00	59.00	57.50
Qmsum	20.91	18.81	19.28	**20.80**	20.22	20.40	20.01	20.28
NarrativeQA	18.83	9.52	11.83	14.53	15.76	15.29	16.05	**16.40**
GovReport	26.55	21.39	21.51	25.95	**26.53**	25.87	25.76	23.88
LCC	58.26	52.41	57.47	53.99	**57.75**	57.20	56.29	55.89
Passage-Count	2.77	2.54	2.29	**4.77**	2.21	2.27	2.27	2.50
Samsum	41.01	37.37	38.57	40.67	41.48	**42.06**	40.73	40.13
Passage-Retrieval-EN	6.5	3.5	3.5	3.5	9.0	**9.5**	9.0	6.0
MultifieldQA-EN	36.15	20.75	21.10	21.61	**37.55**	36.80	34.46	34.21
Average	32.90	24.86	26.40	29.61	32.06	**32.07**	31.78	31.11

A.2 SYSTEM EFFICIENCY ANALYSIS

Table 5: Attention latency (ms ↓) of different methods across various batch sizes and sequence lengths.

BS	Seqlen	Flash	StreamingLLM	H2O	Quest	DS	Ours (7/8-3/4-1/2)	Ours (3/4-3/4-1/2)	Ours (1/2-1/2-1/2)
8	1k	0.230	**0.030**	0.088	0.200	0.141	0.140	0.124	0.090
	2k	0.830	**0.038**	0.093	0.460	0.241	0.191	0.160	0.120
	4k	1.630	**0.420**	0.470	0.850	0.733	0.602	0.570	0.530
16	1k	0.440	**0.030**	0.089	0.280	0.233	0.161	0.120	0.093
	2k	1.630	**0.073**	0.110	0.770	0.422	0.270	0.230	0.190
	4k	3.230	**0.800**	0.850	2.21	1.350	1.080	1.041	0.990

Table 6: Throughput (↑) of different methods across various batch sizes and sequence lengths.

BS	Seqlen	Flash	StreamingLLM	H2O	Quest	DS	Ours (7/8-3/4-1/2)	Ours (3/4-3/4-1/2)	Ours (1/2-1/2-1/2)
8	1k	228	**264**	240	228	228	330	231	235
	2k	188	**252**	234	206	213	220	222	226
	4k	118	**243**	228	152	201	212	214	217
16	1k	374	**465**	441	410	423	428	430	439
	2k	233	**452**	416	287	360	393	398	411
	4k	136	**422**	396	175	286	336	350	365

Here, we provide system efficiency comparisons as the same settings in Sec. 5.1.1 (including two KV cache eviction algorithms **StreamingLLM** and **H2O**, two query-aware token sparsity algorithms **Quest** and **DS**). The results are shown in Tab. 5 (attention latency) and Tab. 6 (end-to-end throughput).

From the results, it can be observed that KV cache eviction algorithms have a clear advantage in system efficiency compared to query-aware token sparsity algorithms. Specifically, StreamingLLM achieves the fastest speed as it applies a fixed sparsity pattern. However, both StreamingLLM

and H2O fall short in terms of accuracy. In contrast, HShare could balance the trade-off between effectiveness and efficiency compared with other query-aware dynamic token sparsity methods. For example, under the configuration of (7/8-3/4-1/2), HShare not only achieves competitive accuracy results over DS but also gains over a 9.17% improvement in e2e throughput when the batch size is 16 and the sequence length is 2k. Notably, as the sequence length increases, the advantage of HShare will be further amplified. For example, when the batch size is 16 and the sequence length is 4k, HShare is 17.48% faster than DS.

A.3 Trade-off accuracy and efficiency Analysis

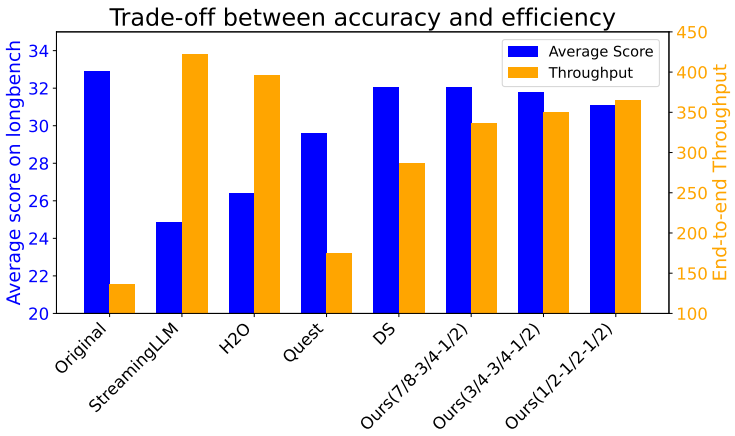

Figure 7: The average score on Longbench and the end-to-end throughput (BS=16, Seqlen=4k) of different methods

Table 7: Attention latency (ms ↓) and GSM8K accuracy of HShare under different sharing ratios.

Sharing Ratio	Attention Latency(ms)	GSM8K(flexible/strict)
1/2	0.31	0.144/0.136
1/4	0.23	0.135/0.125
1/8	0.19	0.132/0.074
1/16	0.10	0.107/0.032

To further illustrate the trade-off between accuracy and efficiency, we present the average score on Longbench and the end-to-end throughput (BS=16, Seqlen=4k) of different methods. The evaluation uses the LLaMA2-7b-chat model, with the results depicted in Fig. 7. Among the compared methods, the original model utilizing FlashAttention2 achieves the highest average Longbench score, albeit at the cost of the lowest throughput. On the other hand, StreamingLLM achieves the highest throughput, significantly outpacing other methods in processing efficiency but exhibits the lowest average score. Notably, our method performs a balance between these two aspects, showing negligible performance degradation while maintaining relatively high throughput. This demonstrates the effectiveness of our approach in achieving competitive accuracy while significantly enhancing processing efficiency relative to other methods.

In addition, we provide the attention latency and GSM8K accuracy of our proposed HShare under different sharing ratios in Tab. 7 to reveal the tradeoff between accuracy and efficiency within HShare.

A.4 More Ablation Studies

HShare VS random share. We further conduct ablation studies to evaluate our layer(head) sharing scheme as shown in algorithm 1. We compare our greedy sharing scheme with random sharing scheme under the same sharing ratio and the corresponding results are shown in Fig. 8. The empirical results indicate that when critical token indices are shared by randomly selected layers and heads, the

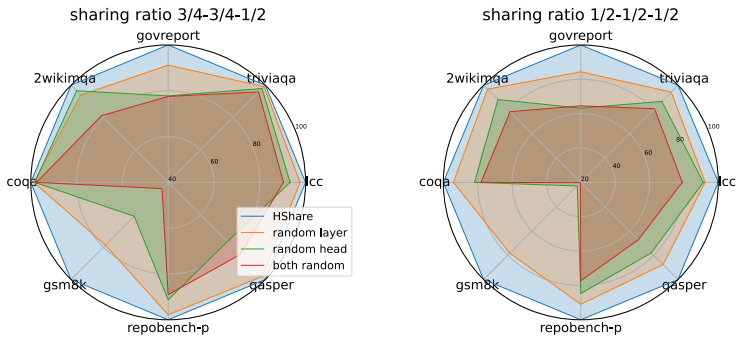

Figure 8: Comparison between HShare and random sharing at the layer level and head level. We take HShare as the 100% baseline and normalize the results of others accordingly.

accuracy decreases significantly across all datasets, further validating the effectiveness of sharing critical tokens between similar layers and heads.

Table 8: Performance of HShare on MultiNews.

Sharing Ratio	1K	2k	3k	4k
Ours-7/8-3/4-1/2	27.78	27.18	25.87	23.51
Ours-3/4-3/4-1/2	27.77	27.55	25.73	22.49
Ours-1/2-1/2-1/2	26.58	25.93	24.65	19.99

Ablation study on different context lengths. To test our method in document summarization with varying context lengths, we conduct experiments on the *MultiNews* dataset, which belongs to the document summarization category. We evaluate the proposed HShare across different context lengths and sharing ratios. The results are presented in Tab. 8. It should be noted that regardless of the length of the text, we consistently selected 256 critical tokens. The results show that the score decreases as the context length increases, and similarly, higher degrees of sharing also lead to a decline in performance. This suggests that when dealing with long context lengths, a more moderate sharing strategy is needed to maintain accuracy, whereas, for shorter texts, a higher degree of sharing can be applied.

B Discussion of potential adaptations

We believe HShare can effectively support transformer variants. HShare aims to optimize long sequence attention operations by sharing critical key and value indices between layers, heads, and queries. As such, convolutional or graph neural networks with attention can also benefit from the proposed HShare. Below are two examples of potential applications:

- Convolutional neural network (CNN) with attention: (Son et al., 2024) proposes to adopt CNNs to extract image features and use spatial-temporal attention to identify crucial frames. To apply HShare in this network, we can predict the indices of critical features (similar to critical tokens), which can then be shared across heads and layers. The potential adaptation involves using only the critical features for attention computation. Since video input exhibits temporal redundancy across nearby frames, critical feature indices can also be shared across frames. HShare can further be applied to other CNN-related works, such as VQA (Anderson et al., 2018).
- Graph neural network (GNN) with attention: (Veličković et al., 2017) adopts multi-head attention to extract features from a set of input nodes. HShare can be seamlessly applied to identify critical nodes and share their indices across heads and layers (if multiple layers are used). Depending on the problem, if the graph nodes are close and similar, the key indices of these nodes can also be shared across nodes, similar to query sharing in HShare. Similar works (Thekumparampil et al., 2018; Wu et al., 2021) can also benefit from applying HShare to reduce computational load.