arXiv:2408.07092v2 [cs.LG] 18 Aug 2024

Post-Training Sparse Attention with Double Sparsity

Shuo Yang [1,3] **Ying Sheng**[2] **Joseph E. Gonzalez**[1] **Ion Stoica**[1] **Lianmin Zheng**[1]
[1]UC Berkeley [2]Stanford [3]Shanghai Jiao Tong University

Abstract

The inference process for large language models is slow and memory-intensive, with one of the most critical bottlenecks being excessive Key-Value (KV) cache accesses. This paper introduces "Double Sparsity," a novel post-training sparse attention technique designed to alleviate this bottleneck by reducing KV cache access. Double Sparsity combines token sparsity, which focuses on utilizing only the important tokens for computing self-attention, with channel sparsity, an approach that uses important feature channels for identifying important tokens. Our key insight is that the pattern of channel sparsity is relatively static, allowing us to use offline calibration to make it efficient at runtime, thereby enabling accurate and efficient identification of important tokens. Moreover, this method can be combined with offloading to achieve significant memory usage reduction. Experimental results demonstrate that Double Sparsity can achieve $\frac{1}{16}$ token and channel sparsity with minimal impact on accuracy across various tasks, including wiki-2 perplexity, key-value retrieval, and long context benchmarks with models including Llama-2-7B, Llama-2-70B, and Mixtral-8x7B. It brings up to a $14.1\times$ acceleration in attention operations and a $1.9\times$ improvement in end-to-end inference on GPUs. With offloading, it achieves a decoding speed acceleration of $16.3\times$ compared to state-of-the-art solutions at a sequence length of 256K. Our code is publicly available at `https://github.com/andy-yang-1/DoubleSparse`.

1 Introduction

Large Language Models (LLMs) have significantly advanced machine learning capabilities, enabling a wide range of applications from natural language processing to complex problem-solving tasks (OpenAI, 2023; Touvron et al., 2023; Google, 2023). However, their inference remains costly and slow due to token-by-token decoding. This decoding process exhibits low arithmetic intensity, making it largely memory-bound. During decoding, access to two types of memory is required: model weights and the Key-Value (KV) cache in the self-attention layers (Vaswani et al., 2017). Both can be very large and thus become bottlenecks. When the batch size is large or the sequence length is long, the size of the KV cache can easily surpass that of the model weights (Pope et al., 2023). While extensive research has focused on reducing access to model weights through quantization and sparsification, the reduction of access to the KV cache has received less attention.

In this paper, we explore methods to reduce access to the KV cache during inference, thereby making attention computation more bandwidth-efficient and accelerating its execution. Our focus is on post-training methods that can be directly applied to a pre-trained model to provide wall-clock acceleration without requiring excessive additional training or fine-tuning overhead. Prior work has attempted to leverage quantization (Hooper et al., 2024; Liu et al., 2024b), compression (Nawrot et al., 2024), and sparsity (Zhang et al., 2024; Anagnostidis et al., 2024; Ge et al., 2024; Ribar et al., 2023) to achieve these goals. Among them, sparsity holds significant potential if a high sparsity ratio can be achieved. The intuition of sparsification is that not every token is equally important for decoding the next token. Therefore, during the decoding process, we can rely on a small subset of important tokens to compute the self-attention, achieving nearly the same results. While the approach

Preprint. Under review.

of sparse attention seems intuitive, previous research has struggled to find a post-training sparse attention method that maintains high accuracy while being runtime-efficient.

The primary challenge in post-training sparse attention lies in accurately and efficiently identifying important tokens. A naive approach entails calculating the entire attention weight matrices and then sorting the tokens based on the accumulated attention weights. Although this method can precisely identify important tokens, it fails to offer a runtime speedup, as it requires computing the full attention weight matrices, which is precisely the step we aim to avoid. Previous studies have proposed various methods for selecting important tokens; however, these methods either lead to significant accuracy losses or fail to achieve practical wall-clock acceleration. Notably, H2O (Zhang et al., 2024) employs a dynamic strategy that maintains a small fixed-size cache of important tokens. Due to its limited size, it must evict many tokens. Since it cannot predict future important tokens, it often inadvertently evicts them, leading to accuracy degradation. SparQ (Ribar et al., 2023), in contrast, retains all tokens and dynamically selects important ones at each step. Yet, its design falls short of achieving the desired speedup and incurs considerable additional memory overhead. In summary, designing an efficient method capable of accurately identifying important tokens remains a significant challenge.

We propose "Double Sparsity," a method that leverages both token sparsity and channel sparsity to achieve accurate and efficient post-training sparse attention. Token sparsity refers to the sparse attention method mentioned above (Zhang et al., 2024), which uses only important tokens to compute self-attention. Channel sparsity, a new method we introduced, selects important tokens at runtime using significant feature channels. Our key insight is that while token sparsity is highly dynamic, channel sparsity exhibits relatively static behavior, enabling us to identify and select important channels through offline calibration. This static channel sparsity thus provides an efficient means to achieve dynamic token sparsity at runtime. Building on this concept, we carefully explored how to select important channels based on statistics from offline calibrations. Furthermore, once we can quickly identify important tokens for the current layer, we extend this process by predicting the important tokens of the next layer. We achieve this by utilizing the embedding similarity between adjacent layers. This approach enables us to offload the entire KV cache to host memory and prefetch only the important tokens to GPU memory, significantly reducing GPU memory footprint.

Shown in Figure 1, we demonstrate that Double Sparsity can achieve both an $\frac{1}{16}$ token sparsity and an $\frac{1}{16}$ channel sparsity simultaneously, while incurring only a negligible accuracy loss across a broad array of benchmarks, including language modeling, question answering, and retrieval tasks. The sparsity directly leads to the reduction of memory access and runtime speedup. Double Sparsity accelerates the attention operation by up to $14.1\times$ at a sparsity level of $\frac{1}{16}$ on NVIDIA A10G and A100G GPUs, closely approaching the theoretical acceleration upper bound. It accelerates end-to-end inference for various workloads by up to $1.9\times$. When turning on offloading, it achieves a decoding throughput that is $16.3\times$ higher than the state-of-the-art offloading-based solutions at a sequence length of 256K.

2 Background

2.1 Preliminaries on Self-Attention and Notations

Attention computation is one of the major bottlenecks in LLM Inference, especially when the sequence length is large (Tay et al., 2022). This is caused by its quadratic computational complexity. Let d_h denote the head dimension, and S denote the number of tokens. We use the decoding step as an example to illustrate the self-attention computation. Each token carries three tensors to embed its information, which are called query, key, and value. In an attention layer, let $q \in \mathbb{R}^{d_h}$ represents the query tensor for input token, $K \in \mathbb{R}^{S \times d_h}$ represents the key tensor for all tokens, and $V \in \mathbb{R}^{S \times d_h}$ represents the value tensor for all tokens. The attention is obtained through the formula shown below:

$$y = softmax\left(\frac{q \cdot K^T}{\sqrt{d_h}}\right) \cdot V$$

2.2 Post-training Sparse Attention

In this work, we introduce the term "post-training sparse attention," analogous to "post-training quantization." Post-training sparse attention refers to techniques that exploit inherent model sparsity, such as token-level sparsity, to accelerate attention calculations without requiring additional training. In the field of LLMs, many works have utilized post-training sparse attention, including H2O, StreamingLLM (Xiao et al., 2024) and SparQ. However, these methods come with significant limitations, presenting serious challenges for post-training sparse attention.

3 Challenges in Post-Training Sparse Attention

In this section, we discuss prior research on post-training sparse attention, identifying the challenges and shortcomings that have prevented these approaches from achieving their full potential. More related work is included in Section 7.

3.1 Retrieval Accuracy

One of the most challenging issues for post-training sparse attention is maintaining retrieval accuracy. For instance, StreamingLLM discards earlier tokens, while H2O selectively drops tokens based on previous attention scores. Although discarding tokens can accelerate computations, this exclusion leads to the loss of critical information, potentially compromising the model's retrieval accuracy. As highlighted in Jelassi et al. (2024), this issue is inherent to techniques that rely on discarding tokens, prompting the exploration of sparse attention methods that preserve the complete KV cache.

3.2 Hardware Friendliness

Achieving wall-clock speedup poses a greater challenge while maintaining model retrieval accuracy, particularly because some post-training sparse attention techniques are not hardware-friendly. For instance, SparQ retains the complete KV cache and computes attention selectively on a subset of the KV cache based on the query. This approach theoretically allows for acceleration while maintaining accuracy. However, SparQ's method of selecting channels and tokens results in non-contiguous memory access, causing substantial L1/L2 cache misses and wasting GPU bandwidth with the standard 128-byte memory access. Despite being designed to accelerate processing, SparQ achieves only a modest 1.3 times speed increase in attention computations. Therefore, it is crucial to develop an algorithm that ensures continuous memory access patterns and avoids dynamic selection of channels to accelerate attention while preserving accuracy.

3.3 Memory Usage

Methods that preserve the complete KV cache inherently require substantial GPU memory consumption. To mitigate the heavy memory demand, the FlexGen (Sheng et al., 2023b) approach offloads the KV cache of each layer to the GPU only during the computation phase. However, a significant challenge arises because FlexGen needs to offload the complete KV cache, and the communication overhead can drastically affect overall system performance. Considering that selected tokens constitute just a small fraction of all tokens, the time taken to offload these specific tokens to the GPU is considerably less than the time required for offloading the entire KV cache as in FlexGen. By efficiently managing when and how data is transferred and processed, it's possible to significantly reduce both the time and memory overhead typically associated with maintaining a full KV cache.

To address these challenges, we propose two post-training sparse attention techniques. In Section 4, we introduce Double Sparsity, which accelerates attention by up to 16 $\times$ with minimal additional memory consumption. In Section 5, we present Double Sparsity-Offload, which reduces memory usage to 1/16 without increasing latency.

4 Double Sparsity

Based on the insights of Section 3, we propose Double Sparsity, a hardware-friendly and bandwidth-efficient post-training sparse attention mechanism. This approach overcomes the challenges highlighted in previous post-training sparse attention techniques by ensuring no loss of information, as it

Algorithm 1 Double Sparsity Decode

Require: $Q \in \mathbb{R}^{d_h}, K \in \mathbb{R}^{S \times d_h}, V \in \mathbb{R}^{S \times d_h}, C \in \mathbb{N}^r,$
$K_{label} \in \mathbb{R}^{S \times r}, r = \alpha d_h, k = \beta S$

Ensure: y

$Q_{label} \leftarrow Q_{[C]}$

$\hat{s} \leftarrow Q_{label} \cdot K_{label}$

$i \leftarrow \text{argtopk}(\hat{s}, k)$

$s \leftarrow \text{softmax}\left(\frac{Q \cdot K^T_{[i,:]}}{\sqrt{d_h}}\right)$

$y \leftarrow s \cdot V_{[i,:]}$

return y

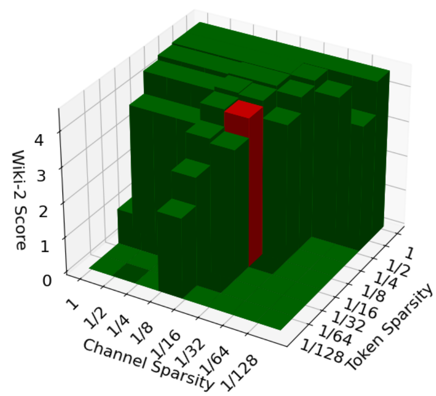


Figure 1: Perplexity of Llama at different token-sparsity and channel-sparsity levels.

Figure 2: Decoding process of Double Sparsity.

maintains the **entire KV cache**. To avoid the cache misses associated with runtime sorting, Double Sparsity utilizes **offline calibration** to pre-determine outlier channels for each transformer layer. A compact **label cache** is employed to store outlier channel values from the Key cache, optimizing memory access patterns to leverage GPU's preference for contiguous memory access. Algorithm 1 and Figure 2 illustrate the decoding process of Double Sparsity.

4.1 Offline Calibration

Offline calibration is a commonly used technique to identify channel sparsity, particularly effective for pinpointing outlier channels. For example, AWQ (Lin et al., 2023) utilizes offline calibration to identify salient weight channels that significantly impact model performance. Inspired by this approach, we employ offline calibration to pre-determine the channels that most influence attention scores. Attention computation can be expressed as $A = Q \cdot K^T$, which can be broken down into $A = \sum_i^{d_h} S_i$ where $S_i = Q_i * K_i$. Due to channel sparsity, only a few S_i have a significant impact on A. Therefore, by conducting offline calibration on a small validation set, we can efficiently identify these critical channels by computing the $\underset{i}{argmax}\ S_i$. Figure 7a in Appendix A illustrates the outlier channels identified by AWQ and Double Sparsity.

To validate the efficacy of outlier channels identified through offline calibration, we conducted a comparison in Appendix A between the outlier channel indices derived from offline calibration and those determined during the online decoding process. A significant overlap between the two sets underscores the reliability of offline-calibrated outliers. Figure 7b illustrates this relationship. An observation from the comparison is that when the ratio surpasses 0.25, the overlap reaches 0.95.

4.2 Forwarding with Label Cache

After identifying the outlier channel indices, it becomes crucial to access them efficiently. Reading these channels directly from the Key cache can lead to non-contiguous memory accesses, which significantly underutilized the bandwidth. To alleviate non-contiguous memory accesses, we leverage a label cache to store pre-determined heavy channel values. This label cache allows for continuous memory access when computing approximate attention, avoiding the need to retrieve non-contiguous segments from the Key cache. During the prefilling stage, all heavy channel values from the Key cache are stored in the label cache; in the decoding phase, only the heavy channel values of new tokens are added. Since approximate attention is not sensitive to precision, we can store the label cache in 4-bit. This approach enables us to maintain a label cache that is only 1/16 the size of the K cache, facilitating contiguous memory access and significantly improving the hit rate of L1/L2 caches, thereby optimizing inference speed and efficiency. In Appendix B.1, an ablation study was conducted to evaluate the impact of label caches. The results demonstrated that a label cache accelerates decoding speeds by 2 to 4 times compared to configurations without a label cache.

5 Reducing GPU Memory Usage with Double Sparsity-Offload

Building upon Double Sparsity, we propose the Double Sparsity-Offload technique to further reduce the GPU memory overhead in large language models. This approach significantly diminishes the memory requirement to 1/16 of the original KV caches. By optimizing memory usage, Double Sparsity-Offload enables more efficient decoding, especially with limited GPU memory resources.

5.1 Prefetching Tokens with Double Buffer

The Double Sparsity-Offload algorithm introduces a double buffer prefetching system during for decoding process. The complete KV cache is stored on the CPU, while the GPU maintains only the label cache and a double buffer. During the decoding process, each layer processes its embeddings through the next layer's query projection to generate an approximate query for the subsequent layer. This approximate query is then used to compute the next layer's approximate attention. While the current layer's attention and feed-forward network computations are being performed, the tokens corresponding to the approximate attention results for the next layer are offloaded to the GPU. This use of double buffering allows for a smooth and efficient overlap of computation and memory transfer.

5.2 Empirical Analysis: Embedding Similarity Between Layers

The feasibility of the Double Sparsity-Offload algorithm is based on the high degree of similarity between embeddings across consecutive layers. To empirically validate this assumption, we conducted an analysis using the Pile validation dataset, applied to the Llama-2-7B model. We measured the cosine similarity of embeddings between every two consecutive layers throughout the model. The results show that apart from the first two layers, the second and third layers, and the very last layers (30 and 31), all other layer pairs exhibited a cosine similarity exceeding 90%, with the majority of layers showing similarities above 95%. These high similarity scores support the viability of utilizing prior layer embeddings to predict queries for subsequent layers in Double Sparsity-Offload.

5.3 Complexity Analysis

To understand the potential speedup of Double Sparsity, we need to analyze its Cache IO-Complexity since attention mechanisms are bandwidth-bounded. Double Sparsity can be simplified into two steps: calculating approximate attention and computing attention over k tokens. Memory-wise, the total access comprises $O(d)$ bytes for Q, $O(S \times r)$ for the label cache, $O(2 \times k \times d)$ for the KV cache, leading to a total of $O(S \times r + 2 \times k \times d) = O(\alpha \times S \times d + 2 \times \beta \times S \times d)$. Given that the approximate attention phase of Double Sparsity does not involve softmax operations, it allows for high parallelism compared to the following step. Therefore, the overall IO complexity of Double Sparsity primarily depends on the latter step, which can be approximated as $O(2 \times \beta \times S \times d)$. This analysis reveals that Double Sparsity's time complexity is linearly dependent on β, and the extra memory overhead is linearly proportional to α. Table 1 summarizes all the sparsity works discussed, specifying their overhead, complexity, and speedup.

Table 1: Comparison of sparsity-related techniques. 'SparQ (1xK)' denotes single-dimension storage of the Key cache, while 'SparQ (2xK)' refers to dual-dimension storage of the Key cache.

Method	On-device Cache Size	Cache IO-Complexity	Min β	Speedup
H2O	$S \times \beta$	$S \times \beta$	1/5	Yes
SparQ (1xK)	S	$S \times \beta$	1/8	No
SparQ (2xK)	$S \times 1.5$	$S \times \beta$	1/8	Yes
AWQ	S	S	1	Yes
Double Sparsity	$S \times (1 + \frac{\alpha}{2})$	$S \times \beta$	1/16	Yes
Double Sparsity-Offload	$S \times \frac{\alpha}{2}$	$S \times \beta$	1/16	Yes

Table 2: Perplexity of models at various sparsity levels. Note the minimal changes in perplexity from sparsity levels 1 to 1/16, with a significant performance gap emerging between levels 1/16 and 1/32.

	Sparsity Level					
Model	1	1/2	1/4	1/8	1/16	1/32
Llama-7B	5.68	5.69	5.69	5.72	5.80	7.66
Llama-2-7B	5.47	5.48	5.53	5.56	5.76	12.01
Llama-2-7B (offloading)	5.47	5.48	5.54	5.57	5.86	15.29
Llama-2-7B-chat	6.94	6.94	6.96	6.92	7.14	14.93
Llama-2-7B-chat (offloading)	6.94	6.94	6.97	6.92	7.33	20.08
Mistral-7B	5.25	5.25	5.26	5.27	5.37	14.55

6 Experiment

In Section 6.1, we demonstrate that both Double Sparsity and Double Sparsity-Offload maintain robust performance with a sparsity setting of 1/16 across various benchmarks, including Wiki-2 perplexity (Merity et al., 2016), MultifieldQA (Bai et al., 2023), GovReport (Huang et al., 2021), TriviaQA (Joshi et al., 2017), and MMLU (Hendrycks et al., 2021). In key-value retrieval tasks, Double Sparsity significantly outperforms other post-training sparse attention techniques. In Section 6.2, we compare Double Sparsity against state-of-the-art attention and end-to-end implementations. Results show that Double Sparsity achieves up to a 16-fold acceleration in attention mechanisms and up to a twofold increase in overall end-to-end processing speed. Additionally, Double Sparsity-Offload achieves a 16-fold acceleration compared to FlexGen Offload.

6.1 Accuracy Evaluation

6.1.1 Wiki-2 Perplexity

Wiki-2 perplexity is a benchmark derived from Wikipedia articles, offering a comprehensive test with its broad vocabulary and authentic text features. A lower score indicates better model performance. Table 2 illustrates the changes in perplexity across different sparsity levels for each model.

To demonstrate the model's performance at different sparsity levels and justify our selection of a sparsity level of 1/16, we constructed a 3D bar chart. According to Figure 9 in Appendix C, a noticeable shift in perplexity is observed as the sparsity level goes beyond 1/16.

To validate the robustness of Double Sparsity and the effectiveness of the 1/16 sparsity level, we conducted a series of ablation studies across various model configurations and conditions. Table 3 demonstrates the effectiveness of Double Sparsity at a sparsity level of 1/16 across various model sizes, attention mechanisms, and MoE configurations.

6.1.2 Long Context Benchmarks

We used Llama-2-7B to evaluate the performance of Double Sparsity across multiple long context benchmarks at various levels of sparsity, comparing its effectiveness with that of StreamingLLM and H2O. As illustrated in Figure 3, Double Sparsity maintains its performance with nearly no drop in accuracy at a sparsity level of 1/16, outperforming other techniques.

Table 3: Ablation study on different architectural models with different outlier types at 1/16 sparsity level. Note that GQA models are incompatible with K outlier channel.

Model	Architecture	Original	Double Sparsity			
			random channel	q outlier	k outlier	qk outlier
Llama-2-7B	Single/MHA	5.47	8.62	6.45	6.61	5.76
Llama-2-7B-chat	Single/MHA	6.94	10.1	7.8	9.44	7.14
Mistral-7B	Single/GQA	5.25	6.06	5.79	N/A	5.37
Llama-2-70B	Single/GQA	3.32	5.15	3.69	N/A	5.17
Mixtral-8x7B	MoE/GQA	3.84	N/A	3.84	N/A	17.3

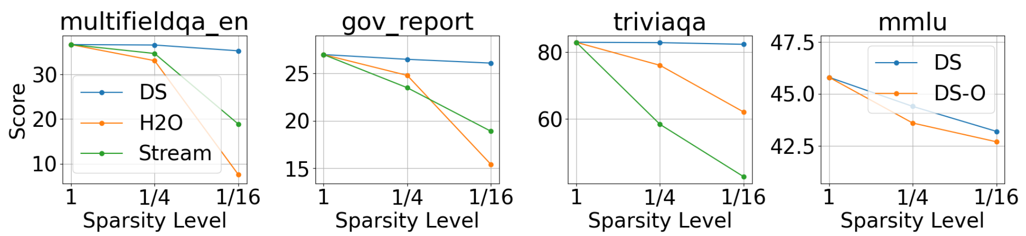


Figure 3: Performance of different techniques across various sparsity levels for long context benchmarks. 'DS' and 'DS-O' refer to Double Sparsity and Double Sparsity-Offloading. 'Stream' refers to Streaming-LLM.

6.1.3 Key-Value Retrieval

The key-value retrieval benchmark is designed to assess a model's in-context retrieval capabilities. Our experiments compared Double Sparsity against other post-training sparsity techniques, including H2O, StreamingLLM, and RTN quantization (Nagel et al., 2020). We also tested the performance of Double Sparsity with the Vicuna-7B-16K model to observe how accuracy changes as context length increases. As shown in Figure 4, we demonstrate that Double Sparsity significantly surpasses the other techniques in key-value retrieval tasks. Notably, Double Sparsity and Double Sparsity-Offload show equivalent performance, highlighting that the offloading mechanism exhibits almost no decay.

6.2 Speedup Evaluation

6.2.1 Setups

Hardware. Our experiments were conducted on two types of GPUs: the A10G and the A100-SXM.

Implementation. For the Double Sparsity Attention, we utilized PyTorch to compute approximate attention and select the top-k tokens. The kernel for attention over top-k tokens was designed using OpenAI Triton. For end-to-end testing, we replaced the full attention mechanism in gpt-fast (PyTorch, 2023) with our Double Sparsity Attention. For Double Sparsity-Offload, we implemented asynchronous CPU to GPU memory copying using CUDA streams and DGL (Wang et al., 2019)'s gathering kernel.

Workload. We focused on high-workload scenarios to push the limits of Double Sparsity. This included a range of batch sizes from 4 to 32 and sequence lengths from 1024 to 16384. For Double Sparsity-Offload, we extended testing to extreme conditions on the A100 GPU, exploring sequence lengths from 64K to 256K. Given that gpt-fast's KV cache is pre-allocated, the tokens-per-second throughput depends solely on the batch size and sequence length.

Baseline. For attention acceleration evaluations, we use the **'scaled_dot_product_attention'** as our baseline. This implementation ranks among the fastest attention mechanisms, dynamically allocating computation among the most efficient options including **FlashAttention-2** (Dao, 2023), **Memory-Efficient Attention** (Lefaudeux et al., 2022), and the top-performing kernels from the PyTorch team. In the end-to-end speed evaluations of Double Sparsity, **gpt-fast** serves as the baseline, distinguished as the state-of-the-art for Llama models on the A100 GPU. It offers exceptionally low

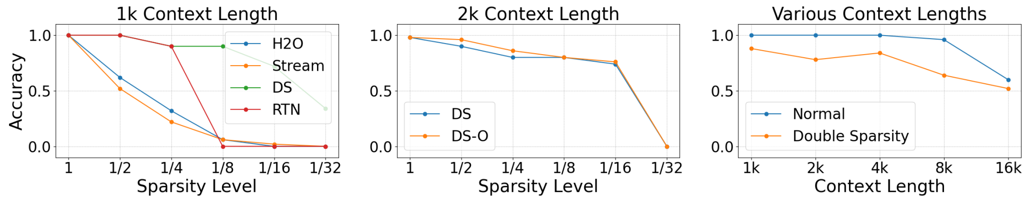


Figure 4: Retrieval accuracy across various sparsity levels and context lengths. 'DS' and 'DS-O' refer to Double Sparsity and Double Sparsity-Offloading. 'Stream' refers to Streaming-LLM. 'RTN' refers to RTN Quantization.

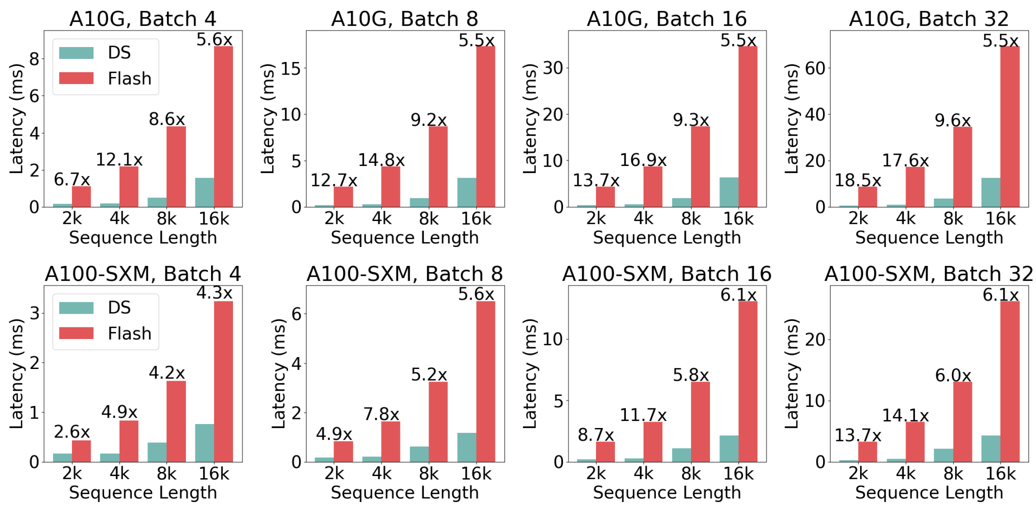


Figure 5: Latency and speedup of Double Sparsity Attention at various batch sizes and sequence lengths. 'DS' indicates double sparsity attention. 'Flash' indicates the 'scaled_dot_product_attention', which is the fastest of FlashAttention-2 and Memory-Efficient Attention.

latency and throughput that surpasses that of the huggingface transformers by tenfold. For evaluating Double Sparsity-Offload, we compare it against FlexGen Offloading, which shares the same gpt-fast codebase and memory footprint.

Other Settings. Given Double Sparsity's focus on the attention mechanism, both weights and activations were set to FP16 precision. Furthermore, considering the limitations imposed by Triton kernels on Torch compile options, neither Double Sparsity nor gpt-fast employed the Torch compiler.

6.2.2 Attention Operator Speedup

Figure 5 provides a comprehensive view of the latency and speedup of Double Sparsity compared to 'scaled_dot_product_attention' across different batch sizes and sequence lengths. On the A10G GPU, every case achieves at least a fivefold speedup, with more than half exceeding ninefold. Notably, Double Sparsity achieves a linear speedup at a sequence length of 4096 with large batch sizes. On the A100 GPU, nearly all cases see at least fourfold faster processing, with larger batches reaching up to tenfold speedup. The greater speedup for smaller batches on the A10G might be due to the launch time of Triton kernels, which becomes significant when the kernel execution time on the A100 is short.

6.2.3 End-to-End Inference Speedup

Figure 6 (a)(b) presents the throughput comparison between Double Sparsity and gpt-fast, measured in tokens per second across various batch sizes and sequence lengths. We deployed the Llama-2-7B model and maximized memory usage to achieve high workload conditions. The results indicate that

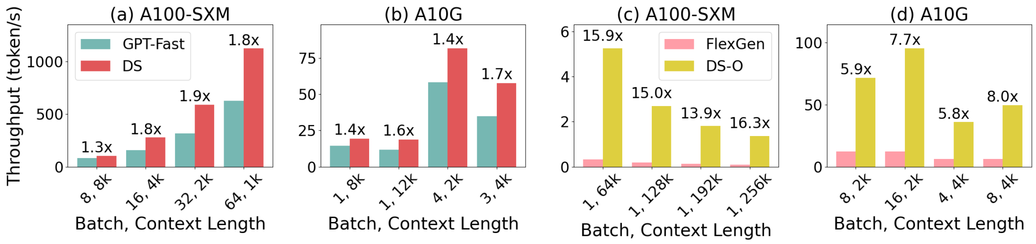


Figure 6: Throughput (token/s) and speedup of Double Sparsity (Offloading) in end-to-end scenarios.

Double Sparsity yields a minimum speedup of 1.3x across all tested conditions. In certain scenarios, the speedup approached twofold, showcasing Double Sparsity's overall efficiency.

In Figure 6 (c)(d), we compare the throughput of Double Sparsity-Offload to FlexGen under a constrained memory footprint, set at 1/16 of a full KV cache for both methods. Both techniques utilize a double buffer for asynchronous data copying. The results show that Double Sparsity-Offload achieves a 4-8$\times$ speedup over FlexGen under regular workloads, and a 16$\times$ speedup in scenarios with long texts ranging from 64K to 256K in sequence length.

7 Related Work

Sparse Attention Inference Due to the significant overhead of attention mechanisms, many studies have focused on exploiting attention sparsity to accelerate inference. These efforts can be categorized under three main criteria: 1) static or dynamic sparse patterns; 2) the presence of token eviction; 3) accelerating pre-filling or decoding. StreamingLLM (Xiao et al., 2024) and LM-Infinite (Han et al., 2023) utilize static sparse patterns with token eviction to accelerate decoding. These approaches achieve inference acceleration by preserving only a small number of initial tokens along with local tokens. H2O (Zhang et al., 2024) and Scissorhands (Liu et al., 2024a) employ dynamic sparse patterns with token eviction for decoding, preserving only a small fraction of the KV cache called heavy hitters according to accumulated attention scores, while FastGen (Ge et al., 2024) uses adaptive sparse attention patterns for different attention heads. MInference (Jiang et al., 2024) serves as a prefilling acceleration method that retains all tokens. It first identifies sparse patterns within the model, and then leverages these identified patterns to compute the pre-filling stage. SparQ (Ribar et al., 2023) and Quest (Tang et al., 2024) implement dynamic sparse decoding while also preserving all tokens. SparQ filters the top-k tokens using heavy channels of queries. Quest segments tokens into multiple pages and computes attention on the top-k pages to facilitate the decoding process.

Sparse Attention Training There are also many efforts to reduce attention complexity through training (Qiu et al., 2020; Ding et al., 2023; Tay et al., 2020; Chen et al., 2021). For example, Sparse transformer (Child et al., 2019) reduces the complexity to $O(n\sqrt{n})$ by introducing sparse factorization of the attention matrix. Reformer (Kitaev et al., 2019) achieves $O(n \log n)$ complexity via locality-sensitive hashing. Longformer (Beltagy et al., 2020), BigBard (Zaheer et al., 2020), and Linformer (Wang et al., 2020) further reduce the complexity to linear. Linear attention architectures have also been proposed in Katharopoulos et al. (2020).

Other Attention and Inference Optimizations Despite efforts to sparsify the attention computation, there are many other optimizations for attention efficiency. Common techniques include quantization and compression (Hooper et al., 2024; Liu et al., 2024b; Kang et al., 2024; Nawrot et al., 2024), efficient attention architecture like multi-query attention (Shazeer, 2019) and group-query attention (Ainslie et al., 2023), and memory-efficient attention algorithms (Rabe & Staats, 2021; Dao et al., 2022). Alternatives to transformers include using the state space model to remove the attention mechanism (Gu et al., 2021). Other common inference optimizations for LLMs include batching Yu et al. (2022), memory optimizations Sheng et al. (2023b); Kwon et al. (2023); Aminabadi et al. (2022), parameter sharing Sheng et al. (2023a); Chen et al. (2023), speculative decoding Stern et al. (2018); Leviathan et al. (2023); Miao et al. (2023), scheduling Han et al. (2022); Agrawal et al. (2023); Patel et al. (2023); Zhong et al. (2024), quantization Xiao et al. (2023); Lin et al. (2023); Dettmers et al. (2022); Frantar et al. (2022), and sparsification Frantar & Alistarh (2023).

8 Future Directions and Conclusion

Future Directions. Despite the progress made with Double Sparsity, several limitations remain that reveal promising directions for future research. It is challenging to perfectly overlap communication with computation. Enhancing asynchronous capabilities to mask communication overheads presents a promising direction that allows for significant acceleration with a minimal memory footprint.

Conclusion. In this work, we introduced Double Sparsity and Double Sparsity-Offload, innovative post-training sparse attention techniques. Double Sparsity leverages offline calibration and label cache to achieve nearly lossless performance across various benchmarks at a 1/16 sparsity level. Performance tests showed that Double Sparsity could accelerate attention computations by up to $16\times$ and achieve an end-to-end speedup of $1.9\times$. Double Sparsity-Offload significantly reduced KV Cache memory usage to 1/16, outperforming the throughput of previous SOTA offloading techniques by 16 times.

References

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.

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

Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, et al. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale. *arXiv preprint arXiv:2207.00032*, 2022.

Sotiris Anagnostidis, Dario Pavllo, Luca Biggio, Lorenzo Noci, Aurelien Lucchi, and Thomas Hofmann. Dynamic context pruning for efficient and interpretable autoregressive transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

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. Longbench: A bilingual, multitask benchmark for long context understanding, 2023.

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

Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34: 17413–17426, 2021.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. *arXiv preprint arXiv:2310.18547*, 2023.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *URL https://openai.com/blog/sparse-transformers*, 2019.

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

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35: 30318–30332, 2022.

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

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Optq: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2022.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uNrFpDPMyo.

Google. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2021.

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

Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. Microsecond-scale preemption for concurrent {GPU-accelerated}{DNN} inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 539–558, 2022.

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

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

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

Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention, 2024. URL https://arxiv.org/abs/2407.02490.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. triviaqa: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. *arXiv e-prints*, art. arXiv:1705.03551, 2017.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2019.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. xformers: A modular and hackable transformer modelling library. `https://github.com/facebookresearch/xformers`, 2022.

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.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

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

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*, 2024b.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 2023.

Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization, 2020.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*, 2024.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Aashaka Shah, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. *arXiv preprint arXiv:2311.18677*, 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, 2023.

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

Jiezhong Qiu, Hao Ma, Omer Levy, Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 2555–2565, 2020.

Markus N Rabe and Charles Staats. Self-attention does not need $o(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

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.

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

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*, 2023a.

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

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

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

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pp. 9438–9447. PMLR, 2020.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=NG7sS51zVF`.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022.

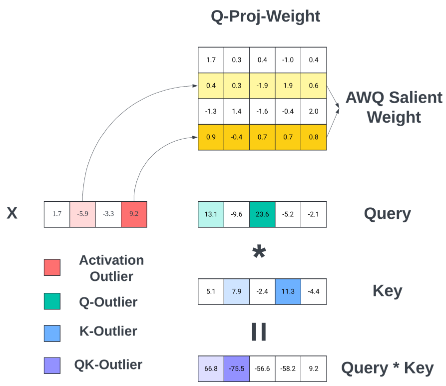
Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

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, 2024.

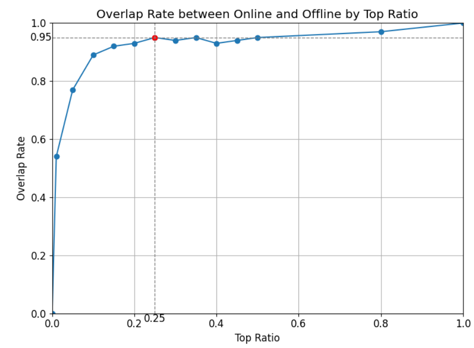
Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. *arXiv preprint arXiv:2401.09670*, 2024.

A Offline Calibration Illustration

The x-axis of the Figure 7b denotes the ratio of the selected top-k channels to the total number of channels, while the y-axis quantifies the degree of overlap between the offline and online outliers.



(a) Outlier channels of AWQ and Double Sparsity.



(b) Outlier channel overlap rate between offline calibration and online decoding.

Figure 7: Analysis of Double Sparsity calibration in identifying outlier channels

B Ablation Study

B.1 Forward without Label Cache

To investigate the significance of the label cache in the forward pass of Double Sparsity, we conducted experiments comparing performance with and without the label cache. As depicted in Table 4, label caches significantly enhance the forward pass, yielding a speedup ranging from 2 to 4 times.

Table 4: Latency comparing performance With and Without Label Cache.

Batch	Seq Len	With Label Cache (ms)	Without Label Cache (ms)	Speedup
4	2048	0.165	0.279	1.7
4	4096	0.181	0.559	3.1
4	8192	0.504	1.250	2.5
4	16384	1.550	3.000	1.9
32	2048	0.467	1.960	4.2
32	4096	0.983	3.950	4.0
32	8192	3.600	9.540	2.6
32	16384	12.600	24.000	1.9

B.2 Embedding Similarity Across Layers

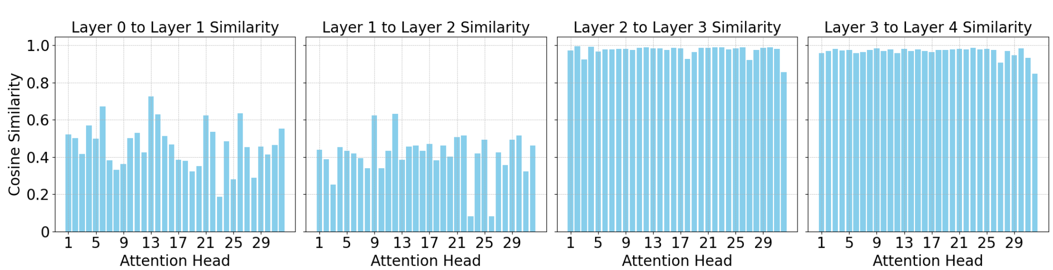


Figure 8: Average cosine similarity of embeddings across all attention heads between layers 0-1, 1-2, 2-3, and 3-4 on the Pile dataset for Llama-2-7B model.

C Perplexity Selection Illustration

Figure 9 uses token-level sparsity as the x-axis, channel-level sparsity as the y-axis, and 10-perplexity values as the z-axis, where higher bars indicate better performance. A sudden shift in perplexity is observed as the sparsity level goes beyond 1/16.

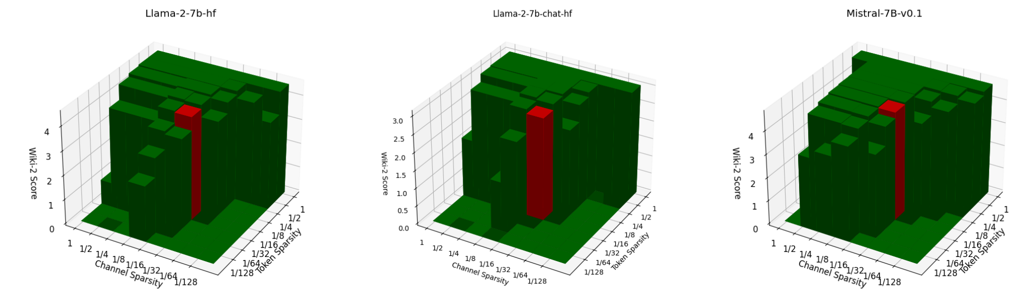


Figure 9: Perplexity of models at different token-sparsity and channel-sparsity levels. Notably, the red bars, representing a sparsity level of 1/16 for both token and channel, show that the model's performance remains largely consistent with the original model at this level.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims in the abstract and introductions are empirically verified by the experimental section 6.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations and future directions to address them are discussed in 8.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: We don't include theoretical results. All the results are derived from experiments.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The experimental setups are described in Section 6.1 and Section 6.2. The code is available.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code is available.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental setups are described in Section 6.1 and Section 6.2. The code is available. The code is available.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We measure the throughput and latency by running a sufficiently large number of tests and reporting the average. These results are highly deterministic, so we do not include error bars in the figures to maintain clarity. The improvements are statistically significant.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experimental setups are described in Section 6.2. We use common NVIDIA GPUs to conduct experiments. The code is available.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research respects NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: The techniques introduced in this paper make the use of large language models easier and more efficient. They do not have additional direct societal impact beyond the existing societal impact of large language models. However, because they accelerate the execution of LLMs, they may amplify the existing societal impact of these models, whether positive or negative.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper does not release new datasets or models, so this is not applicable.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: This paper uses open-weight models and public datasets for experiments. The usage respects all the original licenses.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper doesn't release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.