SpargeAttn: Accurate Sparse Attention Accelerating Any Model Inference

Jintao Zhang [*1] **Chendong Xiang** [*1] **Haofeng Huang** [*1] **Jia Wei** [1] **Haocheng Xi** [2] **Jun Zhu** [1] **Jianfei Chen** [1]

Abstract

An efficient attention implementation is essential for large models due to its quadratic time complexity. Fortunately, attention commonly exhibits sparsity, i.e., many values in the attention map are near zero, allowing for the omission of corresponding computations. Many studies have utilized the sparse pattern to accelerate attention. However, most existing works focus on optimizing attention within specific models by exploiting certain sparse patterns of the attention map. **A universal sparse attention that guarantees both the speedup and end-to-end performance of diverse models remains elusive.** In this paper, we propose SpargeAttn, a universal sparse and quantized attention for any model. Our method uses a two-stage online filter: in the first stage, we rapidly and accurately predict the attention map, enabling the skip of some matrix multiplications in attention. In the second stage, we design an online softmax-aware filter that incurs no extra overhead and further skips some matrix multiplications. Experiments show that our method significantly accelerates diverse models, including language, image, and video generation, without sacrificing end-to-end metrics. The codes are available at https://github.com/thu-ml/SpargeAttn. [1]

1. Introduction

As sequence lengths in large models become longer, such as 45K-128K in video generation and language models (Yang et al., 2025; Bao et al., 2024; Dubey et al., 2024), the time consuming of attention occupies a significant portion of inference latency in large models (Zhang et al., 2025). Fortunately, the attention map $P = \mathrm{Softmax}(QK^\top/\sqrt{d})$ exhibits inherent sparsity, as the softmax operation often creates many values approaching zero (Deng et al., 2024). *Sparse attention methods* exploit such sparsity to accelerate attention by (1) constructing a "sparse mask", which indicates the important non-zero entries of the attention map P that should be computed, and (2) computing attention only for the parts corresponding to the *sparse mask*. There are three distinct categories of sparse attention methods based on how the sparse mask is generated. *pattern-based method* (Zhang et al., 2023; Xiao et al., 2024a; Fu et al., 2024; Zhu et al., 2024; Xiao et al., 2025; 2024b) relies on specific sparsity patterns based on empirical observations, *dynamic sparse attention* (Ribar et al., 2024; Singhania et al., 2024; Jiang et al., 2024; FlexPrefill, 2025; Gao et al., 2024) computes the mask on-the-fly based on the inputs, and *training-based method* (Kitaev et al., 2020; Pagliardini et al., 2023) directly train models with native sparse attention.

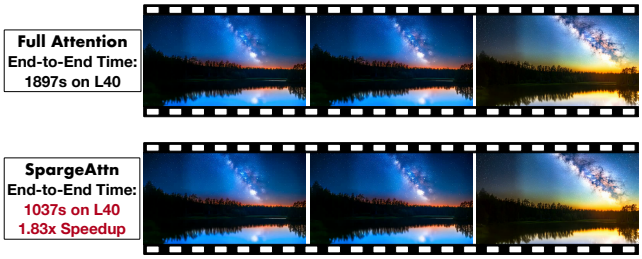

Figure 1. SpargeAttn can achieve 1.83x speedup on Mochi on L40 GPU, with no video quality loss.

Limitation. (L1. Universality) Though existing sparse attention methods already demonstrate promising speedup on some tasks, their universality is still limited. Existing works are typically developed for specific tasks, such as language modeling, utilizing task-specific patterns such as sliding windows or attention sinks. However, the attention pattern varies significantly across tasks (see examples in Fig. 2), making these patterns hard to generalize. (L2. Usability) Moreover, it is difficult to implement both *accurate* and *efficient* sparse attention for any input. This is because *accuracy* demands precise prediction of the sparse regions in the attention map, while *efficiency* requires the overhead of this prediction to be minimal. However, current methods are difficult to effectively satisfy both of the requirements si-

[*]Equal contribution [1]Department of Computer Science and Technology, Tsinghua University [2]University of California, Berkeley. Correspondence to: Jianfei Chen <jianfeic@tsinghua.edu.cn>. *Preprint*.

[1]All experiments in this paper used SpargeAttn based on SageAttention. An updated implementation based on SageAttention2, is available at https://github.com/thu-ml/SpargeAttn. It further offers a 30% speedup over the Attention in this paper.

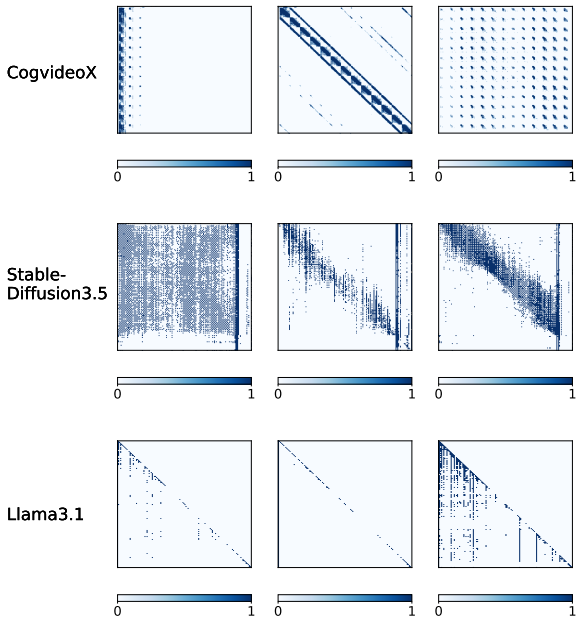

Figure 2. Some sampled patterns of attention map P in video, image, and language generation models.

multaneously. For example, MInference (Jiang et al., 2024) requires a large sequence length, such as 100K, to achieve a noticeable speedup.

Goal. We aim to design a training-free sparse attention operator that accelerates all models without metrics loss.

Our approach. In this work, we develop `SpargeAttn`, a *training-free* sparse attention that can be adopted *universally* on various tasks, including language modeling and text-to-image/video, and various sequence lengths. We propose three main techniques to improve the universality, accuracy, and efficiency. First, we propose a universal sparse mask prediction algorithm, which constructs the sparse mask by compressing each block of Q, K to a single token. Importantly, we compress *selectively* based on the *similarity* of tokens within the block, so the algorithm can accurately predict sparse masks universally across tasks. Second, we propose a sparse online softmax algorithm at the GPU warp level, which further omits some PV products by leveraging the difference between global maximum values and local maximum values in online softmax. Third, we integrate this sparse approach into the 8-bit quantized SageAttention framework for further acceleration.

Result. We evaluate `SpargeAttn` on a variety of generative tasks, including language modeling and text-to-image/video, with comprehensive performance metrics on the model quality. `SpargeAttn` can robustly retain model end-to-end performance while existing sparse attention baselines incur degradation. Moreover, `SpargeAttn` is 2.5x to 5x faster than existing dense and sparse attention models.

2. Related Work

Depending on how the sparsity mask is constructed, sparse attention methods can be divided into three types: **(1) Pattern required methods** rely on some fixed patterns of the attention map, such as sliding windows or attention sinks (Xiao et al., 2024b). H2O (Zhang et al., 2023), InfLLM (Xiao et al., 2024a), and DUOAttention (Xiao et al., 2025) rely on sliding window pattern. SampleAttention (Zhu et al., 2024), MOA (Fu et al., 2024), and StreamingLLM (Xiao et al., 2024b) rely on sliding window and attention sink pattern. DitFastAttn (Yuan et al., 2024) relies on sliding window patterns and similarities between different attention maps. Moreover, DitFastAttn is restricted to simple diffusion transformers, showing incompatibility with language models and MMDiT models like Flux (Black Forest Labs, 2023), Stable Diffusion3 and 3.5 (Stability AI, 2023), and CogVideoX (Yang et al., 2025). As the pattern varies across models, these methods may not universally work for different models. **(2) Dynamic sparse methods** dynamically construct the sparse mask based on the input without the need of preset patterns, and are thus potentially more universal. Existing works can be further categorized into channel compression and token compression. Channel compression methods include SparQAttn (Ribar et al., 2024) and LokiAttn (Singhania et al., 2024). They construct the mask by carrying full attention with reduced dimensionality. However, as the dimension is already small, e.g., 64, 128, in commonly used attention, the speedup potential might be limited. Token compression methods include MInference (Jiang et al., 2024) and FlexPrefill (FlexPrefill, 2025). They construct the mask by compressing each block of tokens to a single token and compute attention on this shorter sequence. However, this approximation is too aggressive: missing important blocks of P is possible if they do not have a large attention score on the compressed sequence. SeerAttention (Gao et al., 2024) requires training of additional parameters for attention, which is expensive to use. Moreover, they are all designed for language models, and their applicability to other model types, such as diffusion models, remains uncertain. **(3) Training-based methods** modify the attention computation logic, requiring retraining the entire model, such as Reformer (Kitaev et al., 2020) and FastAttention (Pagliardini et al., 2023). These methods are much more expensive to use than training-free methods.

There are other ways to accelerate attention, such as optimizing the kernel implementation (Dao et al., 2022; Dao, 2024; Shah et al., 2024), quantization (Zhang et al., 2025), distributing the workload (Liu et al., 2024a), and designing linear time attention (Wang et al., 2020; Choromanski et al., 2021; Yu et al., 2022; Katharopoulos et al., 2020). They are orthogonal to our approach.

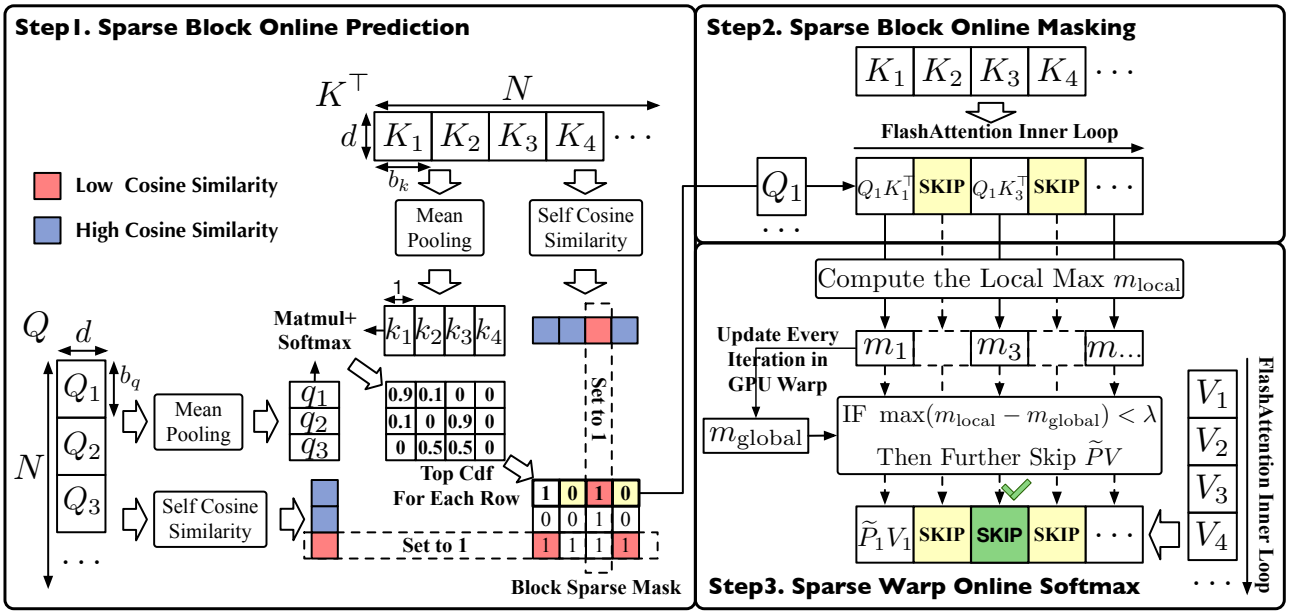

Figure 3. Workflow of SpargeAttn.

3. SpargeAttn

SpargeAttn contains a two-stage online filter to implement sparse FlashAttention. First, as shown in Step1 and Step2 in Fig. 3, we design a fast and accurate method to predict the sparse block in the attention map, thereby skipping the corresponding products of $Q_i K_j^\top$ and $\widetilde{P}_{ij} V_j$. Second, as shown in Step3 in Fig. 3, we design a sparse online softmax method to further skip the products of $\widetilde{P}_{ij} V_j$.

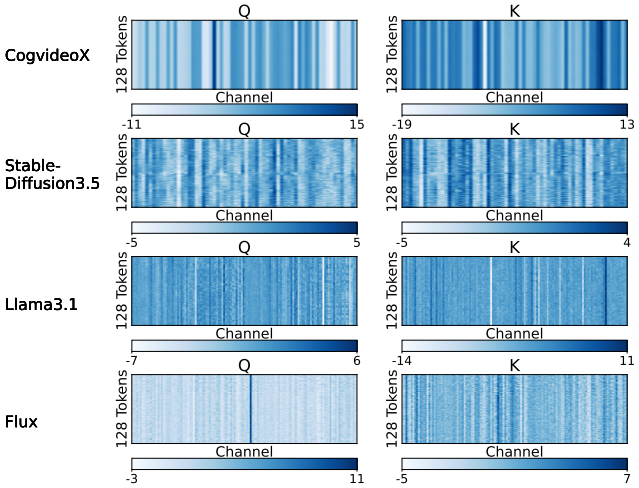

Figure 4. Exemplary patterns of the query and key in the attention of various models.

3.1. Sparse FlashAttention

SpargeAttn adopts the tiling strategy of FlashAttention (Dao, 2024), and skip computing the blocks that are filtered out. Consider an attention operation $S = QK^\top/\sqrt{d}$, $P = \sigma(S)$, $O = PV$, where $\sigma(S)_{ij} = \exp(S_{ij})/\sum_k \exp(S_{ik})$ is the softmax operation. Let N be the sequence length and d be the dimensionality of each head; the matrices Q, K, and V each have dimensions $N \times d$, while the matrix S and P is $N \times N$. FlashAttention proposes to tile Q, K, and V from the token dimension into blocks $\{Q_i\}, \{K_i\}, \{V_i\}$ with block sizes b_q, b_k, b_k, respectively. Then, it uses online softmax (Milakov & Gimelshein, 2018) to progressively compute each block of O, i.e., O_i:

$$\begin{aligned} S_{ij} &= Q_i K_j^\top/\sqrt{d}, \quad (m_{ij}, \widetilde{P}_{ij}) = \tilde{\sigma}(m_{i,j-1}, S_{ij}), \\ l_{ij} &= \exp(m_{i,j-1} - m_{ij}) l_{i,j-1} + \text{rowsum}(\widetilde{P}_{ij}), \\ O_{ij} &= \text{diag}\left(\exp(m_{i,j-1} - m_{ij})\right) O_{i,j-1} + \widetilde{P}_{ij} V_j \end{aligned} \tag{1}$$

where m_{ij} and l_{ij} are $b_q \times 1$ vectors, which are initialized to $-\infty$ and 0 respectively. The $\tilde{\sigma}()$ is an operator similar to softmax.: $m_{ij} = \max\{m_{i,j-1}, \text{rowmax}(S_{ij})\}$, $\widetilde{P}_{i,j} = \exp(S_{ij} - m_{ij})$. Finally, the output O_i can be computed by $O_i = \text{diag}(l_{ij})^{-1} O_{ij}$.

Implementing sparse FlashAttention is intuitive. By *skipping* certain block matrix multiplications of $Q_i K_j^\top$ and $\widetilde{P}_{ij} V_j$, we can accelerate the attention computation. We formulate sparse attention based on FlashAttention in the following definitions.

Definition 1 (Block Masks)**.** Let M_g and M_{pv} be binary masks of dimensions $\lceil N/b_q \rceil \times \lceil N/b_k \rceil$, where each value is either 0 or 1. These masks determine which computations are skipped in the sparse attention mechanism.

Definition 2 (Sparse FlashAttention)**.** The computation rules for sparse FlashAttention based on the masks are defined as follows:

$$Q_i K_j^\top, \widetilde{P}_{ij} V_j \text{ are skipped if } M_g[i,j] = 0. \tag{2}$$

$$\widetilde{P}_{ij}V_j \text{ is skipped if } M_{pv}[i,j]=0. \tag{3}$$

3.2. Selective Token Compression for Sparse Prediction

Key idea. Although attention maps vary across models, we observe that various models exhibit a common trait: Most closer tokens in the query and key matrices of the attention show high similarity (See Fig. 4). Consequently, for blocks composed of highly similar tokens, we can consolidate these tokens into a single representative token for the block. Based on this observation, we propose a pattern-free online prediction method for identifying sparse blocks in P to skip some computation of $Q_iK_j^\top$ and $\widetilde{P}_{ij}V_j$ during the FlashAttention process. Specifically, we first compress blocks exhibiting high self-similarity within Q and K into tokens. Then, we swiftly compute a compressed attention map $\hat{P}$ using the compressed Q and K. Finally, we selectively compute $\{Q_iK_j^\top, \widetilde{P}_{ij}V_j\}$ for those pairs (i,j) where $\{\hat{P}[i,j]\}$ accumulates a high score in the compressed attention map. Importantly, compressing only the token blocks with high self-similarity is crucial, as omitting computations for non-self-similar blocks can result in the loss of critical information. This will be confirmed in Sec. 4 and A.2.

Prediction. As shown in `Step1` in Fig. 3, we first compute a mean cosine similarity across tokens for each block of Q and K. Next, we compress each block into a single token by calculating a mean across tokens. Then, we compute a compressed $QK^\top$ using the compressed Q and K. Finally, to prevent interference from non-self-similar blocks, i.e., the block similarity less than a hyper-parameter θ, we set the corresponding values in S to $-\infty$, and then obtain a compressed attention map through softmax. This algorithm can be expressed as:

$$\begin{aligned}
q=\{q_i\}&=\{\text{mean}(Q_i,\text{axis}=0)\}\\
k=\{k_j\}&=\{\text{mean}(K_j,\text{axis}=0)\}\\
s_{qi}&=\text{CosSim}(Q_i),\ s_{kj}=\text{CosSim}(K_j)\\
\hat{S}[i]&=q_ik^\top;\quad \hat{S}[:,j]=-\infty,\ \text{If } s_{kj}<\theta\\
\hat{P}[i]&=\text{Softmax}(\hat{S}[i])
\end{aligned}$$

where $Q_i\in\mathbb{R}^{b_q\times d}, q_i\in\mathbb{R}^{1\times d}, K_j\in\mathbb{R}^{b_k\times d}, k_j\in\mathbb{R}^{1\times d}$ and $\text{CosSim}(X)=\frac{XX^\top}{|\max(XX^\top)|}$ measures the cosine-similarity within a block.

For each row of $\hat{P}$, i.e., $\hat{P}[i]$, we select the positions of the top values whose cumulative sum reaches $\tau\cdot\sum\hat{P}[i]$, where τ is a hyper-parameter. These positions are set to 1 in $M_g[i,:]$, while all other positions are set to 0.

$$M_g[i,:]=\text{TopCdf}(\hat{P}[i],\tau) \tag{4}$$

where the $\text{TopCdf}(\hat{P}[i],\tau)$ can be formulated as follows.

```
def Top_Cdf(P[i], tau):
    sorted_P, idx = torch.sort(P[i], descending=True)
    cusum_P = torch.cumsum(sorted_P, dim=0)
    mask = cusum_P <= tau * P[i].sum()
    M_i = torch.zeros_like(mask)
    M_i[idx] = mask
    return M_i
```

Finally, we need to ensure that calculations involving non-self-similar blocks of Q or K are not omitted. Therefore, we set all values in the rows of M_g corresponding to not self-similar blocks of Q to 1, and all values in the columns of M_g corresponding to non-self-similar blocks of K to 1.

$$M_g[i,:]=1,\ \text{If } s_{qi}<\theta;\quad M_g[:,j]=1,\ \text{If } s_{kj}<\theta \tag{5}$$

3.3. Masking of the First Stage

Masking. The M_g can be applied in FlashAttention directly to saving some computation. In the inner loop of FlashAttention, i.e., during computing attention between a Q_i and $\{K_j\},\{V_j\}$, we can skip $\{Q_iK_j^\top, \widetilde{P}_{ij}V_j\}$ when $M_g[i,j]=0$.

$$\text{Skip } Q_iK_j^\top \text{ and } \widetilde{P}_{ij}V_j,\ \text{If } M_g[i,j]=0 \tag{6}$$

3.4. Sparse Warp Online Softmax

Key idea. We can further identify the small enough values in the attention map during the online softmax process. If all values in $\widetilde{P}_{ij}$ are close enough to zero, the $\widetilde{P}_{ij}V_j$ will be negligible and can be omitted.

To identify which $\widetilde{P}_{ij}=\exp(S_{ij}-m_{i,j})$ (See Sec. 3.1) contains values small enough to be omitted, we note that in every inner loop of FlashAttention, the O_{ij} will be scaled by $\exp(m_{i,j-1}-m_{ij})$ and then plus the $\widetilde{P}_{ij}V_j$:

$$\begin{aligned}
m_{\text{local}}=&\text{rowmax}(S_{ij}),\quad m_{ij}=\max\{m_{i,j-1},m_{\text{local}}\}\\
O_{ij}=&\text{diag}\left(\exp(m_{i,j-1}-m_{ij})\right)O_{i,j-1}+\widetilde{P}_{ij}V_j
\end{aligned}$$

If $\text{rowmax}(S_{ij})<m_{ij}$, then $m_{ij}=m_{i,j-1}$. Consequently, $O_{ij}=O_{i,j-1}+\widetilde{P}_{ij}V_j$. Furthermore, if $\text{rowmax}(S_{ij})\ll m_{ij}$ holds ture, then all values in $\widetilde{P}_{ij}=\exp(S_{ij}-m_{ij})$ are close to 0. This results in all values in $\widetilde{P}_{ij}V_j$ being close to 0. This condition implies that $\widetilde{P}_{ij}V_j$ is negligible when $\text{rowmax}(S_{ij})$ is significantly smaller than m_{ij}:

$$\begin{aligned}
O_{ij}\approx O_{i,j-1},&\quad \text{if } \max\left(\exp(S_{ij}-m_{ij})\right)\to 0\\
\max\left(\exp(S_{ij}-m_{ij})\right)\to 0 &\Leftrightarrow \max(m_{\text{local}}-m_{ij})<\lambda
\end{aligned}$$

The above equivalence is satisfied when λ is small enough.

Therefore, based on the analysis above, we propose a simple yet effective sparse method to further skip the $\widetilde{P}_{ij}V_j$ computation. Specifically, in the inner loop of FlashAttention, the S_{ij} will be split by c_w GPU warps to $\{S_{ij}[\frac{i_w*b_q}{c_w}:$

Algorithm 1 Implementation of SpargeAttn.

Input: Matrices Q(FP16), K(FP16), V(FP16) $\in \mathbb{R}^{N\times d}$, block size b_q, b_{kv}, count of GPU Warps c_w, hyper-parameters τ, θ, and λ.
Divide Q to $T_m = N/b_q$ blocks $\{Q_i\}$; divide K, V to $T_n = N/b_{kv}$ blocks $\{K_i\}$ and $\{V_i\}$.
$\hat{Q}_i, \hat{K}_j, \delta_Q, \delta_K = \text{Quant}(Q_i, K_j)$; // per-block quantization in SageAttention.
$q = \{q_i\} = \{\text{mean}(Q_i, \text{axis} = 0)\}$; $k = \{k_j\} = \{\text{mean}(K_j, \text{axis} = 0)\}$;
$\hat{S} = qk^\top$; $s_{qi} = \text{CosSim}(Q_i)$; $s_{kj} = \text{CosSim}(K_j)$; $\hat{S}[:, j] = -\infty$, If $s_{kj} < \theta$;
$\hat{P}[i] = \text{Softmax}(\hat{S}[i])$; $M[i, :] = \text{TopCdf}(\hat{P}[i], \tau)$; $M[i, :] = 1$, If $s_{qi} < \theta$; $M[:, j] = 1$, If $s_{kj} < \theta$;
for $i = 1$ **to** T_m **do**
  Load $\hat{Q}_i$ and $\delta_Q[i]$ into a SM ;
  for j in $[1, T_n]$ **do**
    if $M[i, j]! = 0$ **then**
      Load $\hat{K}_j$, $\hat{V}_j$, and $\delta_K[j]$ into the SM ;
      $S_{ij} = \text{Matmul}(\hat{Q}_i, \hat{K}_j^T) \times \delta_Q \times \delta_K$; // dequantization of SageAttention.
      $m_{\text{local}} = \text{rowmax}(S_{ij})$; $m_{ij} = \max(m_{i,j-1}, m_{\text{local}})$; $\widetilde{P}_{ij} = \exp(S_{ij} - m_{ij})$; $l_{ij} = e^{m_{i,j-1} - m_{ij}} + \text{rowsum}(\widetilde{P}_{ij})$;
      $i_w = \text{range}(c_w)$; $I_w = [\frac{i_w * b_q}{c_w} : \frac{(i_w+1) * b_q}{c_w}]$;
      if $\max(m_{\text{local}}[I_w] - m_{ij}[I_w]) < \lambda$ **then**
        $O_{ij}[I_w] = \text{diag}(e^{m_{i,j-1}[I_w] - m_{ij}[I_w]})^{-1} O_{i,j-1}[I_w] + \text{Matmul}(\widetilde{P}_{ij}[I_w], V_j)$; // Paralleled by c_w warps.
      end if
    end if
  end for
  $O_i = \text{diag}(l_{i,T_n})^{-1} O_{i,T_n}$; Write O_i ;
end for
return $O = \{O_i\}$;

$\frac{(i_w+1)*b_q}{c_w}, :]\}$, where i_w is the index of the GPU warp. Let $I_w = [\frac{i_w*b_q}{c_w} : \frac{(i_w+1)*b_q}{c_w}]$. If $\max(m_{\text{local}}[I_w] - m_{ij}[I_w]) < \lambda$, where λ is small enough, then $O_{ij}[I_w] \approx O_{i,j-1}[I_w]$, and we will skip the computation of $\widetilde{P}_{ij}[I_w]V_j$ which is used to update $O_{ij}[I_w]$.

3.5. Combined with SageAttention

To further accelerate our implementation of sparse attention, we integrate our method into SageAttention (Zhang et al., 2025), which proposes a quantized method for accelerating attention. Since quantization operations and sparse operations are orthogonal, sparse computation can be directly applied to SageAttention. The complete algorithm is shown in Algorithm 1. Specifically, first, we need to add one judgment at the beginning of the inner loop of SageAttention (Line 10 in Algorithm 1) to decide whether to skip the whole inner loop once. Second, we add another judgment before the updating of O_{ij} in the inner loop of SageAttention (Line 15 in Algorithm 1) to decide whether to skip the computation of $\widetilde{P}_{ij}V_j$. Moreover, to minimize the attention map prediction overhead, we implement the prediction using CUDA and adopt some kernel fusion techniques.

3.6. Hyper-parameters Determination for Model Layer

Based on the method description in Sec. 3.2 and 3.4, our method incorporates three hyper-parameters: $\tau \in (0, 1)$, $\theta \in (-1, 1)$, and $\lambda < 0$. The parameter determination process for each attention layer in any model is straightforward. We aim to identify a set of hyperparameters that not only maximize attention sparsity but also constrain the attention error across five different model inputs. To evaluate attention accuracy, we employ a strict error metric, the Relative L1 distance, defined as $L1 = \sum |O - O'| / \sum |O|$. The process begins by setting two L1 error thresholds l_1 and l_2, e.g., $l_1 = 0.05, l_2 = 0.06$. We first conduct a grid search for τ and θ to identify the optimal pair that maximizes sparsity while ensuring $L1 < l_1$. Subsequently, we perform another grid search for λ to find the optimal value that further maximizes sparsity while maintaining $L1 < l_2$.

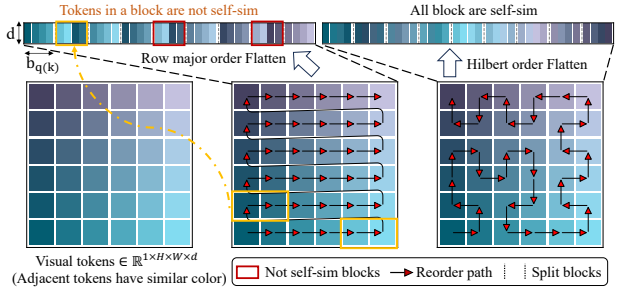

Figure 5. Illustration of different token permutation methods in $1 \times 6 \times 6$ space, with block size of 4.

3.7. HilbertCurve Permutation

Key idea. Improving sparsity while maintaining accuracy is a key challenge in enhancing the performance of sparse attention. In our algorithm, increasing the self-similarity of key and query blocks can reduce the number of non-self-similar blocks. This allows more blocks to participate in TopCdf selection, thereby improving sparsity. Since attention is computationally invariant to token permutations,

the problem reduces to finding a permutation that enhances the similarity of adjacent tokens.

Image and video models benefit from strong priors: adjacent pixels are likely to be similar. To better leverage this prior, we propose the HilbertCurve permutation, given 3D visual tokens $Q, K, V \in \mathbb{R}^{T\times H\times W\times d}$, We use the Hilbert Curve to fill the 3D space and then flatten tokens along the curve into shape $\mathbb{R}^{L\times d}, L = T \times H \times W$. Fig. 5 illustrates an example of $1 \times 6 \times 6$ visual tokens flatten by row-major order and HilbertCurve. The Hilbert Curve preserves locality effectively, traversing the entire 3D space without crossing rows or columns, thereby increasing the similarity of adjacent tokens and the sparsity of attention.

4. Experiment

4.1. Setup

Models. We validate the effectiveness of `SpargeAttn` across diverse representative models from language, image, and video generation. Specifically, we conduct experiments on `Llama3.1` (8B) (Dubey et al., 2024) for text-to-text, `CogvideoX` (2B) and `Mochi` (Team, 2024) for text-to-video, `Flux` (Black Forest Labs, 2023)(.1-dev) and `Stable-Diffusion3.5` (large) (Stability AI, 2023) for text-to-image.

Datasets. The Text-to-text model is evaluated on four zero-shot tasks: WikiText (Merity et al., 2017) to assess the model's prediction confidence, Longbench (Bai et al., 2024) and En.MC of InfiniteBench (Zhang et al., 2024b) for a comprehensive assessment of long context understanding capabilities, and the Needle-in-A-Haystack task (Kamradt, 2023) to assess the model's retrieval ability. Text-to-video models are evaluated using the open-sora (Zheng et al., 2024) prompt sets. Text-to-image models are assessed on COCO annotations (Lin et al., 2014).

End-to-end metrics. For `Llama3.1`, we use perplexity (ppl.) (Jelinek et al., 1977) for WikiText, Longbench score (Bai et al., 2024), and retrival accuracy for the Needle-in-A-Haystack task (Kamradt, 2023). For text-to-video models, following Zhao et al. (2025), we evaluate the quality of generated videos on five metrics: CLIPSIM and CLIP-Temp (CLIP-T) (Liu et al., 2024b) to measure the text-video alignment; VQA-a and VQA-t to assess the video aesthetic and technical quality, and Flow-score (FScore) for temporal consistency (Wu et al., 2023). For text-to-image models, generated images are compared with the images in the COCO dataset in three aspects: FID (Heusel et al., 2017) for fidelity evaluation, *Clipscore* (CLIP) (Hessel et al., 2021) for text-image alignment, and *ImageReward* (IR) (Xu et al., 2024) for human preference.

Speed and sparsity metric. We use TOPS (tera operations per second) to evaluate the speed of sparse attention methods. Specifically, **TOPS** = $O(attn)/t$, where $O(attn)$ represents the total number of operations in a standard attention computation, and t is the latency from a given (Q, K, V) to the output of attention. **Note** that this speed metric is completely fair. This is because the $O(attn)$ is fixed for a set of inputs, and then the speed is determined by t, which includes the time spent predicting the sparse region of the attention map. We define **Sparsity** as the proportion of the Matmul of Q_iK_j plus $P_i^jV_j$ that are skipped relative to the total number of Q_iK_j plus $P_i^jV_j$ in a full attention required.

Implementation and Hyper-parameters. We implement our method using CUDA. As discussed in Sec. 3.6, we need to determine $l1, l2$ for models. We use $(l_1 = 0.08, l_2 = 0.09)$ for `Llama3.1`, $(l_1 = 0.05, l_2 = 0.06)$ for `CogvideoX` and `Mochi`, and $(l_1 = 0.07, l_2 = 0.08)$ for `Stable-Diffusion3.5` and `Flux`.

Baselines. Currently, sparse attention methods applicable across different model types are limited. We choose block-sparse MInference (Jiang et al., 2024) and FlexPrefill (FlexPrefill, 2025) as our baselines. To vary the *sparsity* of these baselines, we use 30% and 70% for MInference, and use $\gamma = 0.95$ and 0.99 for FlexPrefill according to their paper.

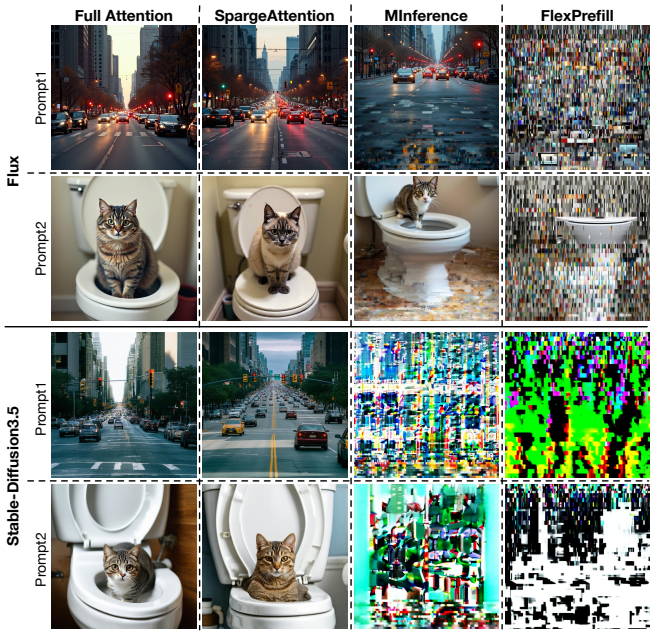

Figure 6. Comparison examples on `Flux` and `Stable-Diffusion3.5`. The sparsity of `SpargeAttn`, MInference and FlexPrefill is 0.38, 0.3, and 0.4 on `Flux` and 0.31, 0.3, and 0.35 on `Stable-Diffusion3.5`.

4.2. Quality and Efficiency Evaluation

End-to-end metrics. We assess the end-to-end metrics of various models using `SpargeAttn` compared to using full attention and baselines. Table 1 shows the results. We can

Table 1. End-to-end metrics across text, image, and video generation models. ✗ indicates an inability to generate results for evaluation. The speed and sparsity are the average for each layer in the model in real generation tasks described in Sec. 4.1. The speed and sparsity of `Llama3.1` are measured in the NeedleInAHaystack task with a 128K sequence length.

Model (seq_len)	**Attention** (Sparsity)	**Speed (TOPS)**↑	**WikiText (Ppl.)** ↓	**Longbench** ↑	**InfiniteBench** ↑	**NIAH** ↑
`Llama3.1` (128K)	Full-Attention	156.9	6.013	38.682	0.6594	0.907
	Minference (0.5)	140.1	10.631	28.860	0.5152	0.832
	FlexPrefill (0.5)	240.6	6.476	38.334	0.6460	0.858
	Minference (0.3)	115.7	6.705	34.074	0.6532	0.870
	FlexPrefill (0.42)	206.9	6.067	38.334	0.6581	0.878
	`SpargeAttn` (0.54)	**708.1**	**6.020**	**39.058**	**0.6638**	**0.909**

Model (seq_len)	**Attention** (Sparsity)	**Speed (TOPS)**↑	**CLIPSIM** ↑	**CLIP-T** ↑	**VQA-a** ↑	**VQA-t** ↑	**FScore** ↑
`CogvideoX` (17K)	Full-Attention	166.0	0.1819	0.9976	80.384	75.946	5.342
	Minference (0.5)	264.6	0.1728	0.9959	70.486	62.410	2.808
	FlexPrefill (0.6)	175.3	0.1523	0.9926	1.5171	4.5034	1.652
	Minference (0.3)	196.9	0.1754	0.9964	77.326	63.525	3.742
	FlexPrefill (0.45)	142.0	0.1564	0.9917	7.7259	8.8426	2.089
	`SpargeAttn` (0.46)	**507.9**	**0.1798**	**0.9974**	**78.276**	**74.846**	**5.030**
`Mochi` (22K)	Full-Attention	164.2	0.1725	0.9990	56.472	67.663	1.681
	Minference (0.5)	202.4	0.1629	0.9891	6.668	50.839	0.653
	FlexPrefill (0.48)	191.3	0.1667	0.9898	0.582	0.0043	✗
	Minference (0.3)	147.7	0.1682	0.9889	14.541	42.956	0.833
	FlexPrefill (0.4)	171.7	0.1677	0.9909	2.941	0.7413	✗
	`SpargeAttn` (0.47)	**582.4**	**0.1720**	**0.9990**	**54.179**	**67.219**	**1.807**

Model (seq_len)	**Attention** (Sparsity)	**Speed (TOPS)**↑	**FID** ↓	**CLIP** ↑	**IR** ↑
`Flux` (4.5K)	Full-Attention	158.2	166.103	31.217	0.8701
	Minference (0.5)	151.8	180.650	30.235	0.4084
	FlexPrefill (0.48)	47.7	443.928	18.3377	-2.2657
	Minference (0.3)	118.9	170.221	31.001	0.7701
	FlexPrefill (0.41)	40.9	405.043	19.5591	-2.2362
	`SpargeAttn` (0.38)	**280.3**	**163.982**	**31.448**	**0.9207**
`Stable-Diffusion3.5` (4.5K)	Full-Attention	164.2	166.101	32.007	0.9699
	Minference (0.5)	186.4	348.930	18.3024	-2.2678
	FlexPrefill (0.37)	23.1	350.497	18.447	-2.2774
	Minference (0.3)	150.3	337.530	18.099	-2.2647
	FlexPrefill (0.35)	22.7	348.612	18.147	-2.2756
	`SpargeAttn` (0.31)	**293.0**	**166.193**	**32.114**	**0.9727**

observe that our method incurs almost no end-to-end metric loss across various models compared to Full-Attention and surpasses baselines with various sparsity levels in terms of end-to-end accuracy. Fig. 6 and 7 show some visible comparison examples on `Flux`, `Stable-Diffusion3.5`, and `Mochi`, showing that `SpargeAttn` incurs no performance loss and outperforms baselines.

Attention speed. Table 1 shows that our method achieves faster speeds compared to Full-Attention and surpasses baselines with various sparsity levels in terms of attention speed. Fig. 9 illustrates the kernel speeds of various methods across different sparsity, highlighting the efficiency of our approach and its significant advantage over other methods.

End-to-end speedup. Table 2 shows the end-to-end latency on `CogvideoX`, `Mochi`, and `Llama3.1` using `SpargeAttn`. Notably, `SpargeAttn` achieves 1.83x speedup on `Mochi`.

Table 2. End-to-end generation latency using `SpargeAttn`.

Model	**GPU**	**Original**	`SageAttn`	`SpargeAttn`
`CogvideoX`	RTX4090	87 s	68 s	**53 s**
`Mochi`	L40	1897 s	1544 s	**1037 s**
`Llama3.1` (24K)	RTX4090	4.01 s	3.53 s	**2.6 s**
`Llama3.1` (128K)	L40	52 s	42s	**29.98 s**

Table 3. Overhead of sparse block prediction in `SpargeAttn`.

Sequence Len	**Prediction (ms)**	**Full Attention (ms)**	Overhead
8k	**0.251**	6.649	3.78%
16k	**0.487**	26.83	1.82%
32k	**0.972**	106.68	0.911%
64k	**2.599**	424.24	0.612%
128k	**8.764**	1696.2	0.516%

4.3. Ablation Study and key Insights

Overhead of sparse block prediction. Table 3 compares the overhead of dynamic sparse block prediction in

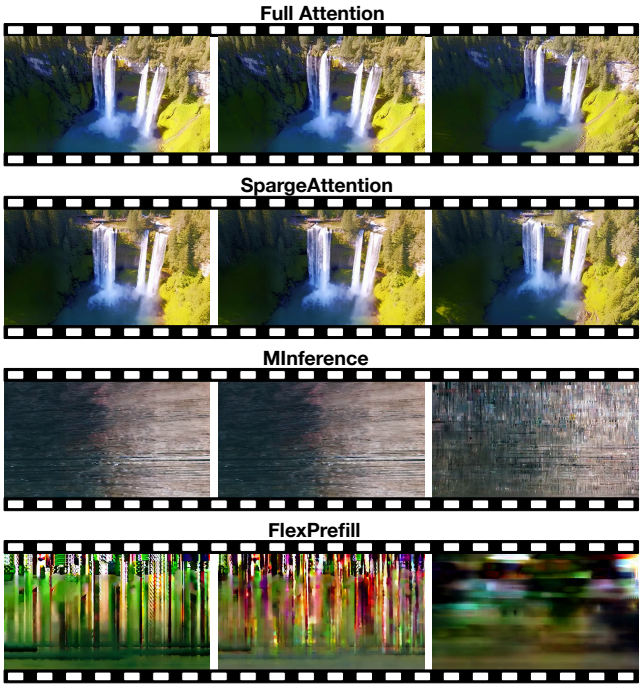

Figure 7. Comparison examples on `Mochi`. The sparsity of `SpargeAttn`, MInference and FlexPrefill is 0.47, 0.3, and 0.4.

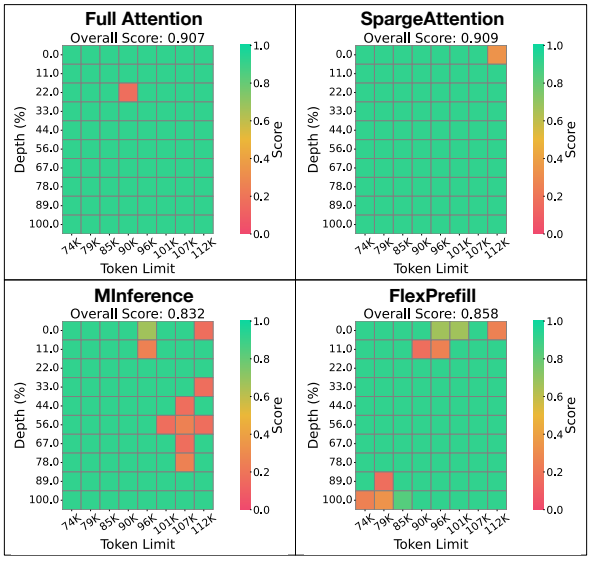

Figure 8. A NeedleInAHaystack comparison example on `Llama3.1`. The sparsity of `SpargeAttn`, MInference, and FlexPrefill is 0.5, 0.5, and 0.54.

`SpargeAttn` compared with attention execution latency. The results indicate that the prediction overhead is minimal compared to attention, particularly for longer sequences.

Effect of Hilbert Curve permutation. We evaluate the impact of Hilbert Curve permutation on `Mochi` by comparing three metrics: average block similarity across blocks of query or key, L1 error defined in Sec. 3.6, and *sparsity*. Table 4 shows that the HilbertCurve permutation consistently achieves superior block self-similarity and sparsity,

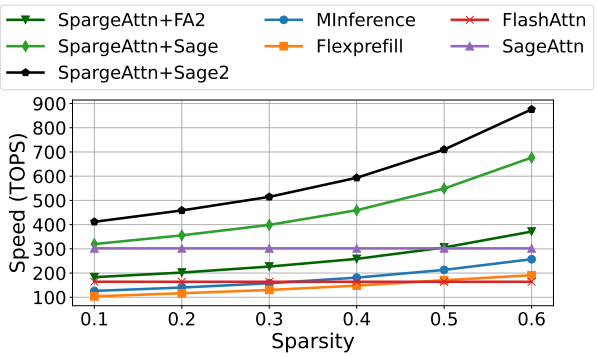

Figure 9. Kernel speed comparison under varying sparsity on RTX4090. Input tensors have a sequence length of 22K and a head dimension of 128. *SpargeAttn+FA2/Sage/Sage2* means deploying our method on FlashAttention2, SageAttention or SageAttention2 (Zhang et al., 2024a).

with only a marginal difference in accuracy. Please see Appendix A.1 for more analysis and details.

Table 4. Effect of permutation on sparsity and accuracy. Sim-q and Sim-k are the average block self-similarity of the query and key.

Method	Sim-q ↑	Sim-k ↑	L1 ↓	Sparsity ↑
Random	0.321	0.019	0.0414	0.048
Rowmajor	0.551	0.390	**0.0307**	0.363
Timemajor	0.514	0.367	0.0342	0.338
HilbertCurve	**0.572**	**0.479**	0.0389	**0.392**

Table 5. Abalation of self-similarity judge.

Method	VQA-a ↑	VQA-t ↑	FScore ↑
W/o. self-sim Judge	34.664	44.722	1.138
With self-sim Judge	54.179	67.219	1.807

Table 6. Analysis of sparsity from M_g and M_{pv}.

Strategy	only M_g	only M_{pv}	M_g +M_{pv}
Sparsity	51.2%	27.7%	54%

Ablation of self-similarity judge We ablate the effect of the self-similarity judge on `Mochi`. As shown in Table 5, we find that self-similarity judge can guarantee end-to-end accuracy. Please see Appendix A.2 for more analysis.

Analysis of sparsity from M_g and M_{pv}. Table 6 shows the sparsity when only using M_g, only using M_{pv}, and using M_g+M_{pv} on `Llama3.1` in NeedleInAHaystack task with 128K sequence length.

`SpargeAttn` enhance the LLM performance. From Table 1, Fig. 8 and 10, we observe that `SpargeAttn` enhances LLM performance in long-context tasks. This improvement may result from the fact that sparse attention helps the LLM focus on more relevant information.

Sparsity increases with sequence length. As shown in Table 7, we find that on `Llama3.1`, sparsity increases with

Table 7. Sparsity increases with sequence length under a constant accuracy bound on `Llama3.1`.

Sequence Len	8K	16K	24K	48K	128K
Sparsity	6.8%	26.4%	35.7%	49.8%	54%

sequence length. This suggests that the longer contexts, the higher speedup of `SpargeAttn` can achieve.

5. Conclusion

In this paper, we propose `SpargeAttn`, a universal sparse and quantized attention that executes attention efficiently and accurately for any input. Our method uses a two-stage online filter: in the first stage, we rapidly and accurately predict the attention map, enabling the skip of some matrix multiplications in attention. In the second stage, we design an online softmax-aware filter that incurs no extra overhead and further skips some matrix multiplications. Experiments show that `SpargeAttn` accelerates diverse models, including language, image, and video generation models, without sacrificing end-to-end metrics.

References

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., Dong, Y., Tang, J., and Li, J. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024.

Bao, F., Xiang, C., Yue, G., He, G., Zhu, H., Zheng, K., Zhao, M., Liu, S., Wang, Y., and Zhu, J. Vidu: a highly consistent, dynamic and skilled text-to-video generator with diffusion models. *arXiv preprint arXiv:2405.04233*, 2024.

Black Forest Labs. Flux. `https://github.com/black-forest-labs/flux`, 2023.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Deng, Y., Song, Z., and Yang, C. Attention is naturally sparse with gaussian distributed input. *arXiv preprint arXiv:2404.02690*, 2024.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

FlexPrefill. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *International Conference on Learning Representations*, 2025.

Fu, T., Huang, H., Ning, X., Zhang, G., Chen, B., Wu, T., Wang, H., Huang, Z., Li, S., Yan, S., et al. Moa: Mixture of sparse attention for automatic large language model compression. *arXiv preprint arXiv:2406.14909*, 2024.

Gao, Y., Zeng, Z., Du, D., Cao, S., So, H. K.-H., Cao, T., Yang, F., and Yang, M. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

Hessel, J., Holtzman, A., Forbes, M., Le Bras, R., and Choi, Y. CLIPScore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7514–7528, 2021.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Jelinek, F., Mercer, R. L., Bahl, L. R., and Baker, J. K. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.

Jiang, H., LI, Y., Zhang, C., Wu, Q., Luo, X., Ahn, S., Han, Z., Abdi, A. H., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Kamradt, G. Llmtest needle in a haystack-pressure testing llms. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`, 2023.

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

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014.

Liu, H., Zaharia, M., and Abbeel, P. Ringattention with blockwise transformers for near-infinite context. In *The Twelfth International Conference on Learning Representations*, 2024a.

Liu, Y., Cun, X., Liu, X., Wang, X., Zhang, Y., Chen, H., Liu, Y., Zeng, T., Chan, R., and Shan, Y. Evalcrafter: Benchmarking and evaluating large video generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22139–22149, 2024b.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Milakov, M. and Gimelshein, N. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

Pagliardini, M., Paliotta, D., Jaggi, M., and Fleuret, F. Fast attention over long sequences with dynamic sparse flash attention. *Advances in Neural Information Processing Systems*, 36:59808–59831, 2023.

Ribar, L., Chelombiev, I., Hudlass-Galley, L., Blake, C., Luschi, C., and Orr, D. Sparq attention: Bandwidth-efficient LLM inference. In *Forty-first International Conference on Machine Learning*, 2024.

Shah, J., Bikshandi, G., Zhang, Y., Thakkar, V., Ramani, P., and Dao, T. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Singhania, P., Singh, S., He, S., Feizi, S., and Bhatele, A. Loki: Low-rank keys for efficient sparse attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Stability AI. Introducing stable diffusion 3.5. `https://stability.ai/news/introducing-stable-diffusion-3-5`, 2023.

Team, G. Mochi 1. `https://github.com/genmoai/models`, 2024.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Wu, H., Zhang, E., Liao, L., Chen, C., Hou, J., Wang, A., Sun, W., Yan, Q., and Lin, W. Exploring video quality assessment on user generated contents from aesthetic and technical perspectives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 20144–20154, 2023.

Xiao, C., Zhang, P., Han, X., Xiao, G., Lin, Y., Zhang, Z., Liu, Z., and Sun, M. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024b.

Xiao, G., Tang, J., Zuo, J., Guo, J., Yang, S., Tang, H., Fu, Y., and Han, S. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *The International Conference on Learning Representations*, 2025.

Xu, J., Liu, X., Wu, Y., Tong, Y., Li, Q., Ding, M., Tang, J., and Dong, Y. Imagereward: Learning and evaluating human preferences for text-to-image generation. *Advances in Neural Information Processing Systems*, 36, 2024.

Yang, Z., Teng, J., Zheng, W., Ding, M., Huang, S., Xu, J., Yang, Y., Hong, W., Zhang, X., Feng, G., et al. Cogvideox: Text-to-video diffusion models with an expert transformer. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yu, W., Luo, M., Zhou, P., Si, C., Zhou, Y., Wang, X., Feng, J., and Yan, S. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10819–10829, 2022.

Yuan, Z., Zhang, H., Pu, L., Ning, X., Zhang, L., Zhao, T., Yan, S., Dai, G., and Wang, Y. DiTFastattn: Attention compression for diffusion transformer models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Zhang, J., Huang, H., Zhang, P., Wei, J., Zhu, J., and Chen, J. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization, 2024a. URL `https://arxiv.org/abs/2411.10958`.

Zhang, J., Wei, J., Zhang, P., Chen, J., and Zhu, J. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *International Conference on Learning Representations*, 2025.

Zhang, X., Chen, Y., Hu, S., Xu, Z., Chen, J., Hao, M., Han, X., Thai, Z., Wang, S., Liu, Z., and Sun, M. ∞Bench: Extending long context evaluation beyond 100K tokens. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, 2024b.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Zhao, T., Fang, T., Huang, H., Liu, E., Wan, R., Soedarmadji, W., Li, S., Lin, Z., Dai, G., Yan, S., Yang, H., et al. Vidit-q: Efficient and accurate quantization of diffusion transformers for image and video generation. In *International Conference on Learning Representations*, 2025.

Zheng, Z., Peng, X., Yang, T., Shen, C., Li, S., Liu, H., Zhou, Y., Li, T., and You, Y. Open-sora: Democratizing efficient video production for all. *arXiv preprint arXiv:2412.20404*, 2024.

Zhu, Q., Duan, J., Chen, C., Liu, S., Li, X., Feng, G., Lv, X., Cao, H., Chuanfu, X., Zhang, X., et al. Sampleattention: Near-lossless acceleration of long context llm inference with adaptive structured sparse attention. *arXiv preprint arXiv:2406.15486*, 2024.

A. Appendix

A.1. Detailed Explain and results of permutation ablation

We use five distinct prompts and pre-searched hyperparameters with $l_1 = 0.05, l_2 = 0.06$ on both `CogvideoX` and `Mochi` models. The permutation are performed separately in attention operation for Q, K, V after position embedding. To retain the original order of the input sequence, an inverse permutation is performed on the output of attention; for models using visual-language joint self-attention(e.g., `CogvideoX`), we only permute the visual tokens. When evaluating block self-similarity, we choose a block size of 128 for query and 64 for key, which aligns with our kernel implementation. The precision metric(L1) is evaluated using FlashAttention2 output as ground truth.

We choose different permutation methods to compare their impact on the performance of attention operations. Given a 3D visual token tensor with shape $T \times H \times W \times d$, the permutation finally results in a tensor with shape $L \times d$, where $L = T \times H \times W$. The permutation methods and their detailed descriptions are shown in Table 8.

Method	Detailed Description
Random	Random permutation of tokens, the order is recorded to perform inverse permutation.
Rowmajor	Permutation following row-major order. Tokens are continuous along the W dimension.
Columnmajor	Permutation following column-major order. Tokens are continuous along the H dimension.
Timemajor	Permutation following time-major order. Tokens are continuous along the T dimension.
HilbertCurve	Permutation following a Hilbert curve.

Table 8. The detailed description of different permutation methods.

Detailed results of permutation ablation for the `CogvideoX` and `Mochi` models are presented in Table 9. The HilbertCurve permutation consistently achieves superior block self-similarity and sparsity, with only a marginal loss in precision. This suggests that the HilbertCurve permutation effectively enhances block self-similarity and sparsity. It is worth noting that the random permutation retains the precision metrics but sacrifices sparsity. This indicates that our algorithm has the property of dynamically adjusting and robust to complex token sequences.

Method	Sim-q↑		Sim-k↑		Precision(L1)↓		Sparsity↑	
	CogvideoX	Mochi	CogvideoX	Mochi	CogvideoX	Mochi	CogvideoX	Mochi
Random	0.502	0.321	0.025	0.019	0.0348	0.0414	0.027	0.048
Rowmajor	0.676	0.551	0.435	0.390	**0.0265**	**0.0307**	0.242	0.363
Columnmajor	0.633	0.547	0.335	0.394	0.0274	0.0342	0.198	0.366
Timemajor	0.692	0.514	0.479	0.367	0.0294	0.0342	0.238	0.338
HilbertCurve	**0.709**	**0.572**	**0.523**	**0.479**	0.0323	0.0389	**0.265**	**0.392**

Table 9. The impact of permutation on `CogvideoX` and `Mochi` models. Sim-q is the block self-similarity of the query, and Sim-k is the block self-similarity of the key.

A.2. Ablation Study of Self-Similarity Judge

To investigate the impact of the self-similarity judge on attention performance, we follow the experimental setting outlined in Sec. A.1 and conduct an ablation study by removing the self-similarity judge. In most cases, the presence of highly localized patterns results in a minimal number of non-self-similar blocks, leading to only minor differences in precision and sparsity when averaging across all tensor cases. To obtain more meaningful and interpretable insights, we specifically analyze cases where the precision difference is statistically significant.

To this end, we apply a threshold-based selection criterion, retaining only those cases where the absolute difference between $L1^{sim-judge}$ (precision error with the self-similarity judge) and $L1^{no-judge}$ (precision error without the self-similarity judge) exceeds 0.05. This criterion results in approximately 2% of the tensor cases being retained for further analysis. We employ precision (L1 error) and sparsity as evaluation metrics to assess the influence of the self-similarity judge on the attention output. The results are summarized in Table 10.

The findings demonstrate that the self-similarity judge effectively mitigates extreme precision loss while introducing only a marginal reduction in sparsity. Furthermore, we observe that a significant proportion of cases exhibiting notable differences

Table 10. Impact of the self-similarity judge on the accuracy and sparsity of attention.

Method	w/ judge		w/o judge		filter w/ judge		filter w/o judge	
	CogvideoX	Mochi	CogvideoX	Mochi	CogvideoX	Mochi	CogvideoX	Mochi
L1 error↓	0.0316	0.0343	0.0325	0.0365	0.0843	0.0555	0.214	0.154
Sparsity ↑	0.199	0.301	0.203	0.305	0.242	0.371	0.275	0.392

originate from the Random permutation category in the CogvideoX model. This observation further highlights the role of the self-similarity judge in enhancing the model's robustness to complex token sequences while maintaining high precision.

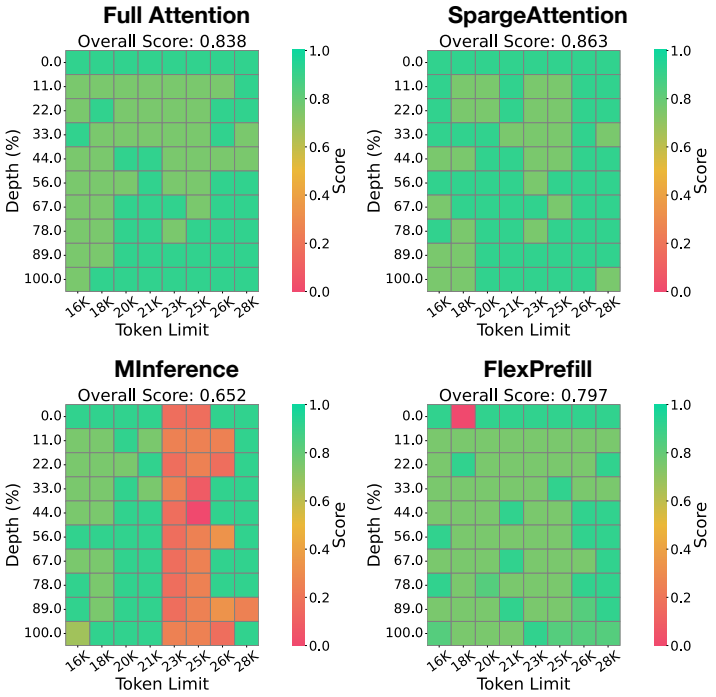

Figure 10. A NeedleInAHaystack comparison example on Llama3.1. The sparsity of SpargeAttn, MInference, and FlexPrefill is 0.36, 0.3, and 0.3.

Table 11. End-to-end metrics on Llama3.1 in the NeedleInAHaystack task with 16-28K sequence lengths.

Model (seq_len)	**Attention** (Sparsity)	**Speed (TOPS)**↑	**NIAH** ↑
Llama3.1 (24K)	Full-Attention	156.9	0.838
	Minference (0.5)	122.5	0.635
	FlexPrefill (0.6)	179.6	0.776
	Minference (0.3)	102.3	0.652
	FlexPrefill (0.3)	117.6	0.797
	SpargeAttn (0.36)	**443.6**	**0.863**

A.3. Additional Experiments

In this section, we present additional experimental results further to evaluate the performance of SpargeAttn compared to baselines. Fig. 10 and 11 show the results on Llama3.1 in the NeedleInAHaystack task with 16-28K sequence length. Fig 11 shows a visible comparison example on Mochi.

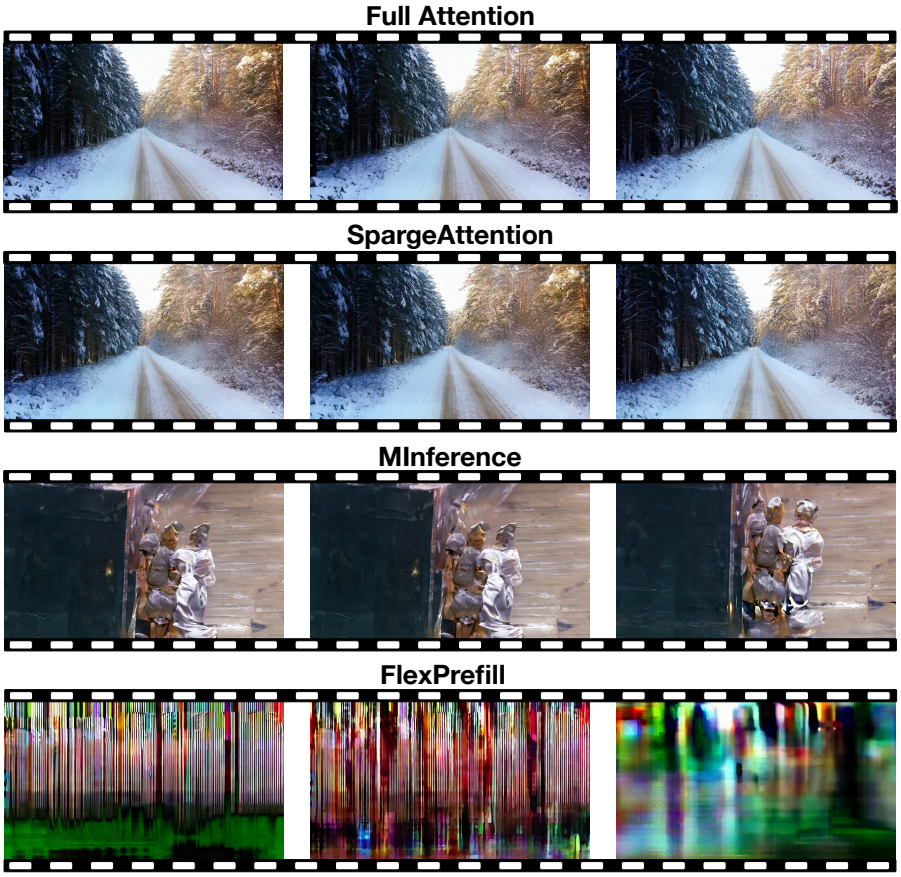

Figure 11. Comparison examples on Mochi. The sparsity of SpargeAttn, MInference and FlexPrefill is 0.47, 0.3, and 0.4.