# FlashMask: Reducing the Complexity of Attention Computation through Sparse Mask Representation

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Recent advancements in Larger-Scale Transformers have significantly benefited from sophisticated attention mechanisms, which are critical for modeling long-context sequences. However, the computational and memory demands of conventional attention mask computations, typically scaling with an $O(N^2)$ complexity where $N$ is the sequence length, pose significant challenges. This paper introduces FlashMask, a simple yet effective *Exact* attention algorithm designed to substantially reduce both the computational complexity and memory requirements of attention computations. By adopting a novel column-wise sparse representation of attention masks, FlashMask achieves a linear memory complexity of $O(N)$ and computational complexity of $O(N) \sim O(N^2)$. We assess the performance of FlashMask in a variety of masking scenarios, including causal and customized attention masks, demonstrating its versatility and robustness across a wide range of attention patterns and models. Our empirical analysis encompasses a variety of downstream training modalities, including Supervised Fine-Tuning (SFT), Direct Preference Optimization (DPO), and Reward Model (RM). We compare FlashMask against state-of-the-art techniques, including notably FlashAttention [1]. In kernel-level assessments, FlashMask achieves substantial computational speedups, up to 6.7x (SFT), 6.9x (DPO), and 8.3x (RM). Furthermore, in end-to-end training, FlashMask consistently enhances training speed significantly, with accelerations up to 2.4x (SFT), 4.2x (LoRA), 2.5x (DPO), and 2.6x (RM) across these varied scenarios without sacrificing model accuracy. Additionally, when implemented in the LoRA scenario, FlashMask enables the LLaMA2-7B to process sequence lengths of up to 544k, significantly enhancing its capability for long-context input.

## 1 Introduction

Transformers [2] , equipped with self-attention mechanisms, have revolutionized natural language processing (NLP) by efficiently modeling data dependencies without the limitations of sequential processing. This makes them ideal for handling long sequences. Large Language Models (LLMs), which utilize training paradigms such as Supervised Fine-Tuning (SFT) [3, 4] and Reinforcement Learning from Human Feedback (RLHF) [5, 6], critically rely on selective attention management through masks. Effective mask management is essential to focus selectively on pertinent data segments, optimizing both performance and computational efficiency.

However, the conventional attention mechanism in Transformers entails a quadratic increase in computational and memory demands $O(N^2)$, where $N$ denotes the sequence length. This exponential growth presents substantial challenges as models scale to sequence lengths ranging from 128K to 1M in advanced systems like GPT-4 [7], Claude [8], and Gemini [9], necessitating more efficient computational approaches. As sequence lengths extend, the memory load for masked attention computations also grows quadratically, adversely affecting computational speed and the ability to manage various mask configurations across different tasks. Current methodologies often resort to approximate sparse attention strategies [10, 11, 12], which unfortunately trade off precision for computational efficiency, underscoring an essential gap in achieving high precision with reduced computational costs.

This paper introduces FlashMask, a novel approach utilizing a sparse mask representation to accelerate attention computations in transformers, effectively addressing both computational and memory scalability issues. Unlike previous methods that compromise accuracy for efficiency, FlashMask provides precise computations without sacrificing accuracy, ensuring high fidelity in attention mechanisms. The contributions of this work include:

- **Exact Computation.** FlashMask uniquely ensures precise attention computations across varying sequence lengths and tasks. It employs a unique column-wise sparse mask representation, denoted by FlashMaskStart (FMS) and FlashMaskEnd (FME), to precisely mask specific rows within columns, ensuring computational efficiency and accuracy.
- **Long Context Modeling.** FlashMask significantly reduces computational and memory demands, enabling efficient processing of extended sequences critical for deploying LLMs in resource-limited settings.
- **Efficient Mask Computation.** FlashMask leverages strategic sparse masking to increase computational throughput, thereby improving processing speeds and broadening the practical utility of LLMs in diverse real-world scenarios.
- **Extensive Empirical Validation.** Empirical studies validate FlashMask's efficiency in computation and storage. Its practical application in real-world scenarios and integration with existing frameworks underscore its potential impact. Moreover, a comprehensive comparison with state-of-the-art methods like FlashAttention-DenseMask, FlashAttention-Varlen highlights FlashMask's efficiency and versatility.

# 2 Background

The attention mechanism has revolutionized data handling in NLP by mimicking human selective focus, allowing neural networks to prioritize parts of the input data. This addresses limitations of traditional sequence-to-sequence models, enhancing context awareness in long sequences. The Transformer model by Vaswani et al. [2] implements this mechanism centrally, using multiple parallel attention heads instead of recurrent layers, thus improving efficiency and performance.

## 2.1 Attention Computation

Central to the Transformer architecture is the attention mechanism, which computes relevance scores between elements in a sequence to focus more on important aspects and less on others. This mechanism can be expressed as:

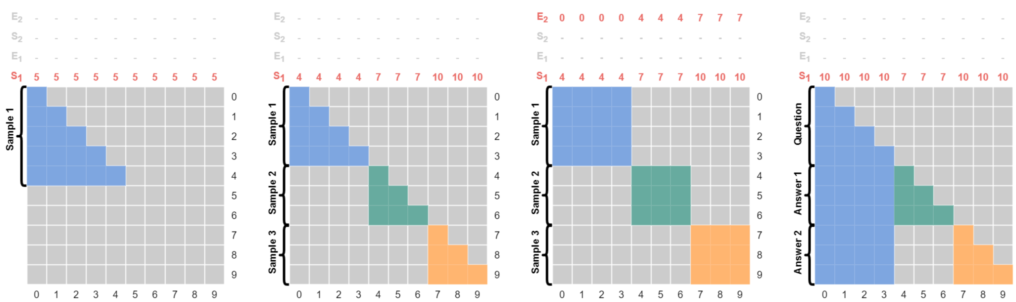
$$\text{Attention}_{mask}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V, \tag{1}$$

where $Q$, $K$, $V$, and $M$ represent the query, key, value, and mask matrices respectively, derived from the input data, and $d_k$ is the dimension of keys. The term $M$ incorporates constraints to selectively consider certain parts of the input sequence during attention computation, enabling functionality like masking future tokens in sequence-to-sequence modeling. One inherent challenge with attention is its computational and memory complexity, both of which scale quadratically with the length of the input sequence. Processing long sequences presents significant challenges, which are exacerbated in the downstream pipeline of training large language models (LLMs). Different training stages, such as Supervised Fine-Tuning (SFT/LoRA [3, 4, 13, 14, 15]), Direct Preference Optimization (DPO) [16, 17, 18, 19, 20], Reward Model (RM) [5, 21, 22, 23, 24], and Proximal Policy Optimization (PPO) [25, 6], place diverse demands on the attention mask.

## 2.2 Masking Variable-Length Sequences

The advent of large transformer-based models has marked substantial progression in handling increased sequence lengths in natural language processing. Previously, models like BERT [26] and

GPT-2 [27] were limited to sequences of approximately 512 tokens, whereas more recent adaptations such as the LLaMA [28, 29, 30], GPT-4 [7] and Claude series [8] stretched these limits to encompass 2K to 200K tokens, respectively. Innovations from Google's Gemini [9] have further shifted this boundary, managing up to 1M tokens. Enhanced sequence management within these models employs various masking techniques in the attention matrix, adapting to the length and diversity of input sequences. Techniques such as the use of padding operations are illustrated in Figure 1(a), which help maintain efficiency by allowing uniform processing of diverse input lengths through padding masks. However, conventional padding can lead to inefficiencies due to the diverse sequence lengths typically found in training data, often following a long-tail distribution. This issue is adeptly addressed by dynamic token allocation technologies like InToken [31, 3, 32, 33, 34], which optimize computational resources by adjusting the token count based on actual data needs, significantly improving the training efficiency for datasets with various sequence lengths in Figure 1(b)(c).




(a) (b) (c) (d)

Figure 1: Common patterns of attention masks. (a) Padded masks from single-sequence inputs in unidirectional (uni-) attention. (b) InToken masks from grouping several masks with different lengths in uni-attention. (c) InToken masks in bidirectional (bidi-) attention. (d) Question and Answering Masks in uni-attention.

Despite having extensive text-handling capabilities, the meticulous design of masking configurations remains crucial for specific training scenarios. The illustrated scenarios in Figure 1(d) and Figure 2 depict various specialized masking mechanisms employed to enhance model training efficiency and applicability. Figure 1(d) illustrates a scenario involving DPO/RM with two or more answers, where each answer's tokens have visibility to the tokens of the question, and tokens from different answers are not visible to each other. Multi-shot and in-context learning scenarios facilitated by extended attention spans in configurations like Figure 2(a) are becoming prevalent, which allows the final question in a series to receive comprehensive attention, enhancing contextual understanding [35, 36]. Furthermore, hybrid masking forms combining features from different methodologies are demonstrated in Figure 2(b). These incorporate sink tokens [37] and a sliding window mask from the Big Bird [38], facilitating a localized yet extensive context capture. Figure 2(c) is also derived from Big Bird, showing a bi-directional global attention mask, which allows for a comprehensive global context capture. Such innovative approaches in masking not only bolster the efficiency of training large transformer models but also pave the way for advanced explorations into the capabilities of attention mechanisms, such as simulating token eviction during inference as depicted in Figure 2(d). These advancements underscore the dynamic and adaptable nature of transformer technology in accommodating varying training needs and enhancing the overall performance of LLMs.

### 2.3 Attention Optimization Techniques

As aforementioned in Equation 1, the computational and memory demands of this mechanism, particularly the computation of $QK^T$, become significant as the sequence length $N$ increases. This is due to the size of the resultant attention scores matrix, which scales quadratically with the sequence length, leading to a complexity of $O(N^2)$. Several related works has been proposed to alleviate the issue. In the realm of model training optimizations, Memory Efficient Attention [39] (MEA) and FlashAttention [1] have been pivotal. MEA focuses on reducing the model's memory demands by altering the self-attention mechanisms. This allows either for the use of larger models or for the extension of maximum sequence lengths within existing hardware constraints. On the

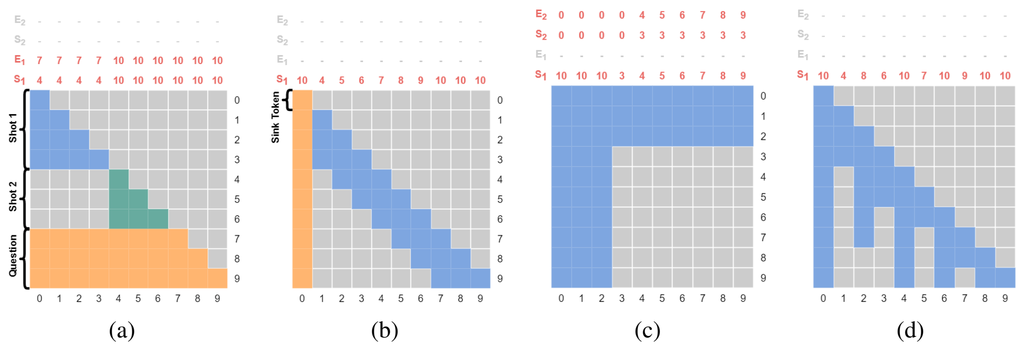



Figure 2: Extended patterns of attention masks. (a) In-context learning formatted multi-shot masks in uni-attention. (b) Sink + Slidewindow masks in uni-attention. (c) Global masks in bidi-attention. (d) Customized masks in uni-attention.

other hand, FlashAttention enhances the efficiency of attention mechanisms with IO-Awareness to better utilize contemporary GPU architectures, resulting in faster computations and reduced energy consumption. This method reduces memory overhead to $O(N)$ utilizing tiling techniques during the computation process, making it particularly effective in scenarios without the need for a custom mask. However, for specific training contexts requiring custom masking, the memory overhead with FlashAttention remains $O(N^2)$. Note that, in typical training setups like unidirectional causal attention or bidirectional full-context attention, the default mode of operation with FlashAttention does not involve passing a custom mask.

During the inference stage, optimizations such as FlashDecoding [40] and FlashDecoding++ [41] play crucial roles. FlashDecoding enhances the decoder in transformers to expedite the generation of sequences by optimizing state management and employing techniques that minimize computational waste. FlashDecoding++ further advances these improvements, incorporating sophisticated dynamic batching and more refined state management to significantly boost throughput and reduce latency. Concerning long sequence training, RingAttention [42] is notable for its efficiency in distributed training contexts, managing communication overhead and memory utilization effectively across multiple nodes.

Another class of study targets on the sparsity/low-rank of attention computation. The Sparse Transformer [10] revolutionizes sequence processing with log-linear complexity. Similarly, Reformer [43] optimizes memory via locality-sensitive hashing, while Big Bird [38] introduces a hybrid attention method to manage longer sequences efficiently. Linformer [44] reduces complexity using low-rank approximations, significantly economizing computation and storage requirements. Both of the previously discussed solutions either compromise precision or yield only marginal enhancements in efficiency. Conversely, our proposed FlashMask is capable of delivering an exact computations.

# 3 FlashMask: Algorithm and Analysis

In this section, we present the critical design of the column-wise sparse mask representation, implementation of the mask computation kernel, and a complexity analysis of the proposed FlashMask.

## 3.1 Column-wise Sparse Mask Representation

We introduce FlashMask, a column-wise sparse masking technique, represented using $\mathbf{FMS}, \mathbf{FME} \in \mathbb{R}^N$ (the row index of **F**lash **M**ask **S**tart and **F**lash **M**ask **E**nd), where $\mathbf{FMS}_c, \mathbf{FME}_c$ denote that elements in the $c$-th column of the attention score matrix $\mathbf{S} = \mathbf{QK}^T$ within the interval $[\mathbf{FMS}_c, \mathbf{FME}_c)$ are masked (set to $-\infty$). As shown in Fig. 2(a), $\mathbf{FMS} = [4, 4, 4, 4, 10, 10, 10, 10, 10, 10]$, $\mathbf{FME} = [7, 7, 7, 7, 10, 10, 10, 10, 10, 10]$ indicates that, for the first column, the 4-th to 6-th rows are masked.

## 3.2 Integration with FlashAttention

Unidirectional (causal) attention, commonly utilized in large language models, incorporates FlashMask within the FlashAttention-2 algorithm, as detailed in Algorithm 1. This paper elaborates the implementation of FlashMask using the lower triangular section of the mask for illustration, where the blue section represents the computation by the dense mask method (for comparison and not

**Algorithm 1** Optimized Forward Pass with FlashMask

**Require:** Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM, block sizes $B_c$, $B_r$, dense mask $\mathbf{D} \in \mathbb{R}^{N\times N}$, column-wise sparse mask starting rows $\mathbf{FMS} \in \mathbb{R}^N$, ending rows $\mathbf{FME} \in \mathbb{R}^N$.

1: Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
2: Divide the output $\mathbf{O} \in \mathbb{R}^{N\times d}$ into $T_r$ blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, and divide the logsumexp $L$ into $T_r$ blocks $L_i, \ldots, L_{T_r}$ of size $B_r$ each.
3: Divide $\mathbf{D}$ into $T_r \times T_c$ blocks $\mathbf{D}_{1,1}, \ldots, \mathbf{D}_{T_r,T_c}$.
4: Divide $\mathbf{FMS}$ into $T_c$ blocks $\mathbf{FMS}_1, \ldots, \mathbf{FMS}_{T_c}$, and divide $\mathbf{FME}$ into $\mathbf{FME}_1, \ldots, \mathbf{FME}_{T_c}$.
5: Precompute the max value $\mathbf{maxFMS}_1, \ldots, \mathbf{maxFMS}_{T_c}$ for each $\mathbf{FMS}_1, \ldots, \mathbf{FMS}_{T_c}$, write to HBM.
6: Precompute the max value $\mathbf{maxFME}_1, \ldots, \mathbf{maxFME}_{T_c}$ for each $\mathbf{FME}_1, \ldots, \mathbf{FME}_{T_c}$, write to HBM.
7: Precompute the min value $\mathbf{minFMS}_1, \ldots, \mathbf{minFMS}_{T_c}$ for each $\mathbf{FMS}_1, \ldots, \mathbf{FMS}_{T_c}$, write to HBM.
8: Precompute the min value $\mathbf{minFME}_1, \ldots, \mathbf{minFME}_{T_c}$ for each $\mathbf{FME}_1, \ldots, \mathbf{FME}_{T_c}$, write to HBM.
9: **for** $1 \le i \le T_r$ **do**
10: Load $\mathbf{Q}_i$ from HBM to on-chip SRAM.
11: On chip, initialize $\mathbf{O}_i^{(0)} = (0)_{B_r\times d} \in \mathbb{R}^{B_r\times d}, \ell_i^{(0)} = (0)_{B_r} \in \mathbb{R}^{B_r}, m_i^{(0)} = (-\infty)_{B_r} \in \mathbb{R}^{B_r}$.
12: **for** $1 \le j \le T_c$ **do**
13: **if** $i \times B_r \ge \mathbf{maxFMS}_j$ **and** $(i+1) \times B_r \le \mathbf{minFME}_j$ **then**
14: Continue
15: **end if**
16: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
17: Load $\mathbf{FMS}_j$ from HBM to on-chip SRAM.
18: Load $\mathbf{FME}_j$ from HBM to on-chip SRAM.
19: On chip, compute $\mathbf{S}_i^{(j)} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r\times B_c}$.
20: On chip, set $\mathbf{S}_i^{(j)} = \mathbf{S}_i^{(j)} + \mathbf{D}_{i,j}$
21: **if** $(i+1) \times B_r \ge \mathbf{minFMS}_j$ **and** $i \times B_r \le \mathbf{maxFME}_j$ **then**
22: On chip, set $\mathbf{S}_i^{(j)}[x][y] = -\infty, \forall x, y$, such that $\mathbf{FMS}_j[y] \le i \times B_r + x \le \mathbf{FME}_j[y]$
23: **end if**
24: On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \mathrm{rowmax}(\mathbf{S}_i^{(j)})) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_i^{(j)} = \exp(\mathbf{S}_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r\times B_c}$ (pointwise), $\ell_i^{(j)} = e^{m_i^{j-1} - m_i^{(j)}} \ell_i^{(j-1)} + \mathrm{rowsum}(\tilde{\mathbf{P}}_i^{(j)}) \in \mathbb{R}^{B_r}$.
25: On chip, compute $\mathbf{O}_i^{(j)} = \mathrm{diag}(e^{m_i^{(j-1)} - m_i^{(j)}})^{-1} \mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_i^{(j)} \mathbf{V}_j$.
26: **end for**
27: On chip, compute $\mathbf{O}_i = \mathrm{diag}(\ell_i^{(T_c)})^{-1} \mathbf{O}_i^{(T_c)}$.
28: On chip, compute $L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$.
29: Write $\mathbf{O}_i$ to HBM as the $i$-th block of $\mathbf{O}$.
30: Write $L_i$ to HBM as the $i$-th block of $L$.
31: **end for**
32: Return the output $\mathbf{O}$ and the logsumexp $L$.

present in FlashMask) and the red section indicates the FlashMask computation. FlashAttention Forward involves two nested loops; the outer loop iterates over each block $\mathbf{Q}_i$ of $\mathbf{Q}$, and the inner loop iterates over all blocks $\mathbf{K}_j$ of $\mathbf{K}$ and $\mathbf{V}_j$ of $\mathbf{V}$. In the inner loop, $\mathbf{S}_i^{(j)} = \mathbf{Q}\mathbf{K}^T$ is computed on SRAM. Once $\mathbf{S}_i^{(j)}$ is generated, the corresponding dense mask is added as a bias (shown in line 20 of Algorithm 1), whereas FlashMask applies the column-wise sparse mask by setting elements beyond $\mathbf{FMS}_c$ but not exceeding $\mathbf{FME}_c$ to $-\infty$ (as shown in lines 21 to 23 of Algorithm 1).

FlashMask further exploits the block computation feature of FlashAttention-2 to reduce computation. If all elements within a block are masked, the block's computation, including matrix multiplication and softmax operations, can be skipped. A block defined by rows $[r_0, r_1)$ and columns $[c_0, c_1)$ is skipped if $r_0 \ge \mathbf{max}(\mathbf{FMS}_{c_0:c_1})$ and $r_1 \le \mathbf{min}(\mathbf{FME}_{c_0:c_1})$. Considering that mask regions often exhibit continuity, most blocks are either completely masked or not at all, with only boundary blocks requiring fine-grained masking. A block is completely unmasked if every coordinate $(r, c)$ satisfies $r < \mathbf{FMS}_c$ or $r \ge \mathbf{FME}_c$, thus skipping fine-grained masking and avoiding extra masking overhead.

To avoid redundant computations in the FlashAttention-2 compute loop, we precompute $\mathbf{max}(\mathbf{FME}_{c_0:c_1})$ and $\mathbf{min}(\mathbf{FME}_{c_0:c_1})$ for each block before the execution loop using a kernel. This computation has a complexity of $O(N)$ and can be easily distributed over $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ thread blocks. A parallel reduction operation within each thread block then computes the maximum and minimum values, yielding $T_c$ values. The additional space complexity introduced here is $O(T_c)$. Similar computations are made for $\mathbf{max}(\mathbf{FMS}_{c_0:c_1})$, $\mathbf{min}(\mathbf{FMS}_{c_0:c_1})$,.

The backward computation in FlashAttention-2, which is typically column-parallel, benefits more from the column sparse mask approach. Blocks for which $\left\lfloor \frac{\mathbf{max}(\mathbf{FMS}_{c_0:c_1})}{B_r} \right\rfloor < i < \left\lfloor \frac{\mathbf{min}(\mathbf{FME}_{c_0:c_1})}{B_r} \right\rfloor$ are fully masked, allowing skipping of these intervals directly. Only blocks satisfying $\left\lfloor \frac{\mathbf{min}(\mathbf{FMS}_{c_0:c_1})}{B_r} \right\rfloor \leq i \leq \left\lfloor \frac{\mathbf{max}(\mathbf{FME}_{c_0:c_1})}{B_r} \right\rfloor$ require fine-grained masking.

It is important to note that unlike various approximate attention algorithms, our method ensures that each effective element of the attention score matrix is computed identically to FlashAttention-2, with masked elements explicitly set to $-\infty$, thus maintaining the accuracy of the algorithm's results. Futhermore, FlashMask is easily extendable to bidirectional attention computations.

### 3.3 Complexity Analysis

We define sparsity as $\rho = \frac{p}{N^2}$, where $p$ is the number of masked elements in the attention score matrix, and $N$ is the maximum sequence length of Q and K, $N^2$ being the total number of elements in the attention score matrix. For a causal mask, $\rho = \frac{2\times p}{N^2}$ since half of the elements in the attention score matrix are already masked by the causal mask. The block sparsity $\alpha$ is defined as $\alpha = \frac{a}{\left\lceil \frac{N}{B_r} \right\rceil \times \left\lceil \frac{N}{B_c} \right\rceil}$, where $B_r, B_c$ are block sizes, and $a$ is the number of completely masked blocks. For a causal mask, $\alpha = \frac{2\times a}{\left\lceil \frac{N}{B_r} \right\rceil \times \left\lceil \frac{N}{B_c} \right\rceil}$.

**Space complexity.** The dense mask is represented as $\mathbf{D} \in \mathbb{R}^{N\times N}$, with a space complexity of $O(N^2)$. FlashMask denotes as $\mathbf{FMS}, \mathbf{FME} \in \mathbb{R}^N$, occupying $O(N)$ space, along with four precomputed arrays $\mathbf{maxFMS}, \mathbf{minFMS}, \mathbf{maxFME}, \mathbf{minFME} \in \mathbb{R}^{\left\lceil \frac{N}{B_c} \right\rceil}$, also occupying $O(N)$ space. Thus, the total space complexity for FlashMask is $O(N)$, significantly reducing memory usage and supporting training on longer sequences.

**Memory access complexity.** The dense mask accesses the entire $\mathbf{D} \in \mathbb{R}^{N\times N}$ matrix in line 20 of Algorithm 1, totaling $N^2$ memory accesses on HBM. FlashMask reads the $\mathbf{FMS}, \mathbf{FME} \in \mathbb{R}^N$ vectors from HBM as shown in lines 17 and 18 of Algorithm 1, with each $\mathbf{Q}_i$ reading the entire $\mathbf{FMS}, \mathbf{FME}$, totaling $2\times T_r \times N$ memory accesses. This reduces the memory access to approximately $\frac{2\times T_r \times N}{N^2} \approx \frac{2}{B_r}$, significantly boosting performance. Due to FlashMask's smaller space usage, it is possible to preload $\mathbf{FMS}, \mathbf{FME}$ into SRAM using only $2 \times B_c$ SRAM, enhancing memory access efficiency. For the backward process, which uses a column-parallel approach, SRAM-stored $\mathbf{FMS}, \mathbf{FME}$ can be well reused, further reducing the total memory access on HBM to $2 \times N$.

**Computational complexity.** The attention computation process normally iterates over the entire attention score matrix, with a computational complexity of $O(N^2)$. By skipping entirely masked blocks, FlashMask leverages block sparsity to reduce computational complexity to $O((1-\alpha)N^2)$.

## 4 Experiments

### 4.1 Setup

Experiments were conducted using GPU A800-SXM 80G, Intel(R) Xeon(R) Platinum 8350C CPUs, CUDA 12.0, and driver version 525.125.06. We evaluated FlashMask against various methods including Vanilla Attention, FlashAttention with dense mask (FA-DenseMask), variable length (FA-Varlen), and sliding window (FA-Window) across different scenarios and sequence lengths. Both kernel-level and end-to-end performance demonstrated the effectiveness of our method.

### 4.2 Data Construction

As mentioned in the Background section, commercial large models now support sequences up to 128K in length. FlashMask, with its lower memory overhead, can facilitate training with even longer

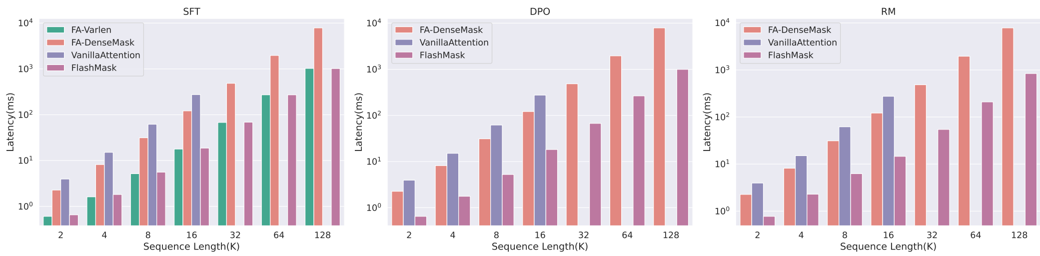


Figure 3: Comparison of Kernel Latency Based on Varying Sequence Lengths. FlashMask achieves substantial computational speedups, up to 6.7x (SFT), 6.9x (DPO), and 8.3x (RM).

contexts. However, currently available public datasets do not contain training data for scenarios exceeding 128K. For comprehensive testing of FlashMask, we constructed synthetic data to simulate long-sequence training.

For a given sequence length $L$, sequences were generated by mimicking InToken method with several sub-sequences. Randomly selecting $s \in [1, 10]$ split points uniformly within the range $(0, L)$, the sequence was divided into $s$ sub-sequences. The segment from the last split point to the end of the sequence was considered as Padding. For the RM scenario, shorter sequence lengths used a smaller upper limit on the number of splits: $s \in [1, 3]$ for $L \in (0, 4096]$ and $s \in [1, 4]$ for $L \in (4096, 8192]$. By discarding samples not meeting size requirements, we ensure each sub-sequence length was at least 128 (SFT, LoRA, DPO) or 512 (RM) and padding not exceeding 128 (SFT, LoRA, DPO) or 512 (RM). Suppose one sub-sequence with length $L'$ was further divided into a query and $k$ answers based on the scenario. The length of each answer was randomly determined from the range $[\frac{0.1L'}{1+0.1\times k}, \frac{0.2L'}{1+0.2\times k}]$, making the answer lengths approximately $[0.1, 0.2]$ of the query length. Therefore, the query length was equal to $L'$ minus the total answer lengths. A total of 240 valid samples per given sequence length $L$ were collected and binned into 10 categories by sparsity $\rho$, as shown in Appendix A.2.

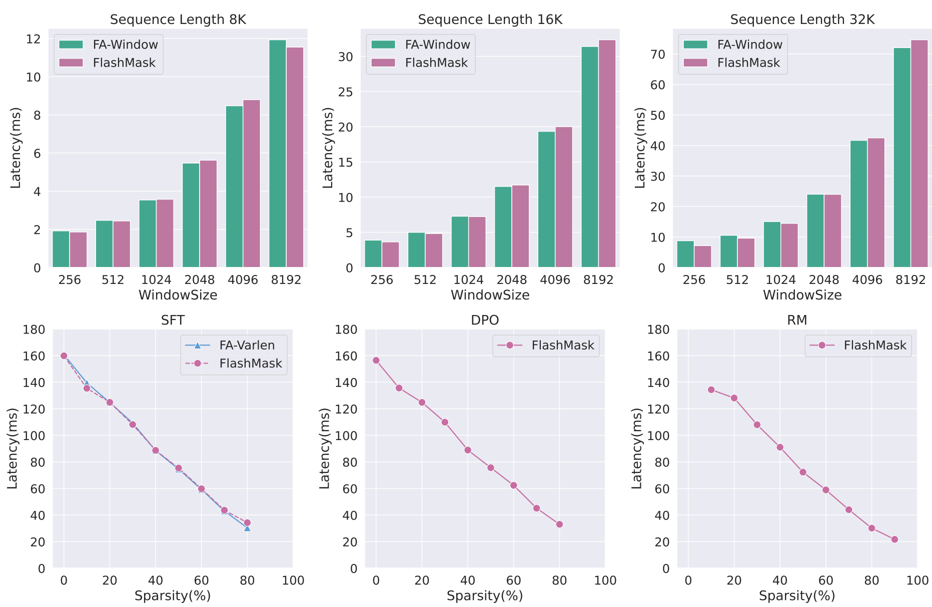
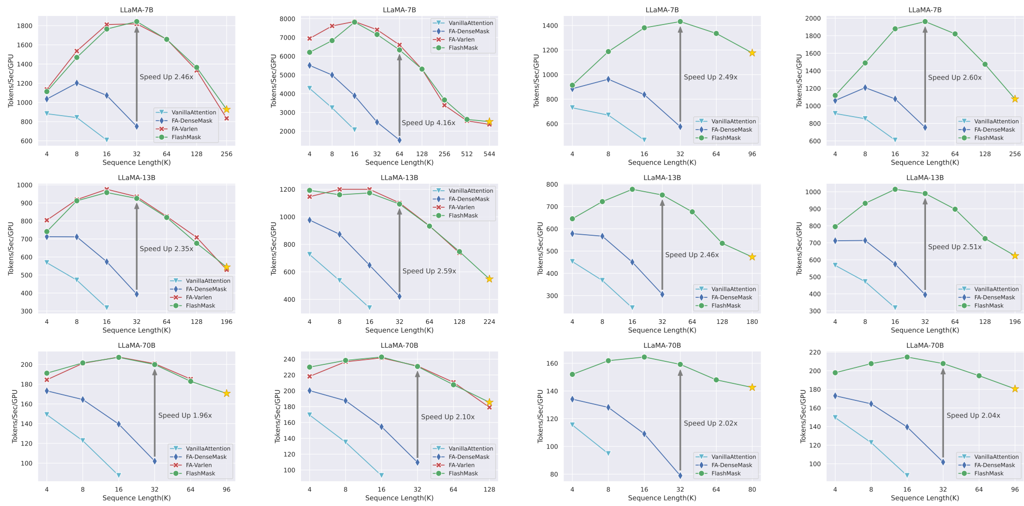


Figure 4: Top: Comparison of Kernel Latency while Varying Window Size. Bottom: Comparison of Kernel Latency while Varying Input Sparsity.

## 4.3 Kernel Experiments

We conducted tests with batch sizes of 1, 2, and 4 using Vanilla Attention, FA-DenseMask, and FlashMask. Each experiment began with 5 warm-up runs followed by 50 measurements, totaling 55 runs with kernel latency as the performance metric. Additional comparisons were made with FA-Varlen in the SFT scenario. Results for batch size 1 are shown in Figure 3 (results for batch sizes 2 and




(a) SFT (b) LoRA (c) DPO (d) RM

Figure 5: Comparison of End-to-End Training Throughput on Synthetic Dataset.

4 can be found in Appendix A.3). FlashMask demonstrated significant latency advantages across all lengths, up to 8.3-fold time saving compared to FA-DenseMask. Vanilla Attention was significantly more time-consuming and exceeded memory limits at lengths greater than 32K. The closest competitor to FlashMask, FA-Varlen, exhibited higher latencies as sequence lengths increased. Similar trends were observed in the DPO and RM scenarios, with FlashMask significantly outperforming FA-DenseMask and Vanilla Attention, especially in the RM scenario where higher sparsity levels further enhanced FlashMask's effectiveness. Performance benefits from varying sparsity levels were also quantified, with FlashMask showing linear negative correlation with increasing sparsity, demonstrating efficient utilization of sample sparsity for acceleration. FlashMask's capability to perform sliding window attention was further tested against FA-Window with window sizes of 256, 512, 1024, 2048, 4096, and 8192, as shown in Figure 4 Top. FlashMask matched FA-Window in latency across sequence lengths of 8K, 16K, and 32K, showing comparable delay performances at increasing window sizes.

### 4.4 End-to-End Experiments

The end-to-end performance[1] of the model was tested using synthetic datasets across three scales of the LLaMA2 model and four downstream scenarios (SFT, LoRA, DPO, RM) at various sequence lengths, measuring throughput in average Tokens/Sec/GPU. Each sequence length of 240 valid samples was trained for one epoch, with results presented in Figure 5. In the SFT scenario, FlashMask showed a clear throughput advantage over FA-DenseMask and Vanilla Attention, performing comparably to FA-Varlen. As sequence lengths increased, the throughput advantage of FlashMask over FA-DenseMask and Vanilla Attention also enhanced, even enabling the completion of longer sequence tasks within the same computational resources. In LoRA, DPO, and RM scenarios, FlashMask consistently showed significant advantages. Notably, in the LoRA scenario at the LLaMA2-7B, FlashMask achieved a 4.16x throughput improvement over FA-DenseMask, supporting sequence lengths up to 544K. It's important to note that FA-Varlen was unable to support the DPO and RM scenarios with the answers sharing one question, whereas FlashMask was capable of handling various scenarios including DPO and RM.

Additional experiments were conducted on the open-source dataset LongBench [45], comparing the end-to-end performance of FA-DenseMask, FA-Varlen, and FlashMask at sequence lengths of 16K, 32K, and 64K. The performance improvements were consistent with those observed in the synthetic dataset. The detailed results are presented in Appendix A.3. Memory usage during the experiments was also recorded, showing significant reductions for FlashMask compared to FA-DenseMask, with detailed results presented in Appendix A.3.

[1]To simplify the tuning of hyperparameters, we standardize the global batch size to 16, with a batch size of 1 per device. Additional training hyperparameters are detailed in Table 1

# 5 Discussion

Several key topics emerge that are crucial for comprehending the full scope and implications of FlashMask. These include the rationale behind the design choices, adaptations for supporting bidirectional and other custom masks, and the necessity as well as limits of the current approach.

**Necessity and Scope of the Study.** The substantial advancement rendered by FlashMask in improving attention mask computation is a significant evolution over the current FlashAttention framework. Notably, FlashMask addresses and significantly mitigates the limitations observed with FlashAttention in handling conventional and custom mask computations. This enhancement not only broadens the applicative reach of FlashAttention but also signifies a key shift in efficiency metrics critical for Transformer architectures. More importantly, the flexibility of FlashMask extends beyond the proprietary boundaries of FlashAttention, offering potential benefits to a wider range of Transformer-based models. By facilitating more efficient computation of the attention mechanism, FlashMask enables innovations in processing vast datasets and complex models, thereby improving performances across varied applications in the LLM field. This cross-model adaptability confirms the robustness and utility of FlashMask as a universally applicable enhancement tool within and potentially outside the Transformer architecture spectrum, promising substantial gains in computational efficiency and model scalability.

**Bidirectional and Custom Masks.** In the exploration of attention mechanisms, the introduction of FlashMask as discussed in this study offers a significant leap in computational efficiency, particularly for masking processes in unidirectional attention mechanisms. By extending this approach to bidirectional networks through the simple addition of vectors indicating the start and end indices of the mask, FlashMask transcends conventional computational bounds, casting itself not just as a sparse attention methodology, but as a versatile computational paradigm. Its adaptability across various custom masking tasks and ability to effectively manage diverse types of mask combinations underscores its potential to greatly enhance the efficiency of attention computations. Moreover, the inherent sparsity of the attention mask during inference provides a robust justification for employing FlashMask, indicating its utility and effectiveness in practical applications. This paradigm shift highlights the importance of developing scalable and efficient computational strategies in the evolving landscape of transformer architectures, suggesting that future research should continue to leverage these innovations to tackle increasing computational demands.

**Limitations and Future Directions.** While FlashMask demonstrates impressive performance in handling long-context sequences, it is observed that the computational cost of training Transformers increases more than linearly as the sequence length grows—not only due to the computation of masked attention but also because of the extensive use of other operators. This scenario highlights the inevitable need for leveraging or integrating distributed computing strategies or further algorithmic enhancements to elevate training efficiency. Such advancements could be practical in managing the computationally intensive tasks involved in processing extended contexts efficiently. As a part of future research directions, exploring synergistic solutions that combine the strengths of both algorithmic innovation (like FlashMask) and distributed system designs stands as a promising venture. This approach is anticipated to address scalability challenges and could set the stage for breakthroughs in handling unprecedentedly large data sets and complex model architectures.

# 6 Conclusion

In this paper, we introduced FlashMask, a groundbreaking attention computation paradigm designed to tackle the high computational and memory demands inherent in conventional attention mechanisms in large-scale transformers. By implementing a novel column-wise sparse representation of attention masks, FlashMask substantially reduces the memory and computational complexity from quadratic to linear with the sequence length, thereby enhancing processing speeds and efficiency. Our algorithm demonstrates versatility across various masking scenarios and retains robust performance in different training pipelines. Extensive empirical analysis confirms that FlashMask accelerates computational speed significantly, achieving up to 8.3x speedup in common modalities comparable to state-of-the-art methods like FlashAttention. This advancement marks a significant leap forward in the design of attention computation, offering the potential for broader applications and setting a new benchmark in the efficiency of processing long-context sequences.

# References


[1] 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.

[2] 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.

[3] Srinivasan Iyer, Xi Victoria Lin, Ramakanth Pasunuru, Todor Mihaylov, Daniel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, et al. Opt-iml: Scaling language model instruction meta learning through the lens of generalization. *arXiv preprint arXiv:2212.12017*, 2022.

[4] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

[5] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[6] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Y Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[7] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[8] Anthropic. Introducing claude. `https://www.anthropic.com/news/introducing-claude`, 2024. Accessed: May 20, 2024.

[9] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

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

[11] 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.

[12] Zhiqing Sun, Yiming Yang, and Shinjae Yoo. Sparse attention with learning to hash. In *International Conference on Learning Representations*, 2021.

[13] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[14] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

[15] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023.

[16] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

[17] Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*, 2023.

[18] Tianqi Liu, Yao Zhao, Rishabh Joshi, Misha Khalman, Mohammad Saleh, Peter J Liu, and Jialu Liu. Statistical rejection sampling improves preference optimization. *arXiv preprint arXiv:2309.06657*, 2023.

[19] Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

[20] Tianqi Liu, Zhen Qin, Junru Wu, Jiaming Shen, Misha Khalman, Rishabh Joshi, Yao Zhao, Mohammad Saleh, Simon Baumgartner, Jialu Liu, et al. Lipo: Listwise preference optimization through learning-to-rank. *arXiv preprint arXiv:2402.01878*, 2024.

[21] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. *arXiv preprint arXiv:2303.00001*, 2023.

[22] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[23] Binghai Wang, Rui Zheng, Lu Chen, Yan Liu, Shihan Dou, Caishuang Huang, Wei Shen, Senjie Jin, Enyu Zhou, Chenyu Shi, et al. Secrets of rlhf in large language models part ii: Reward modeling. *arXiv preprint arXiv:2401.06080*, 2024.

[24] Alexandre Rame, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. *Advances in Neural Information Processing Systems*, 36, 2024.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

[27] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

[29] 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.

[30] Wei Huang, Xudong Ma, Haotong Qin, Xingyu Zheng, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. How good are low-bit quantized llama3 models? an empirical study. *arXiv preprint arXiv:2404.14047*, 2024.

[31] Mario Michael Krell, Matej Kosec, Sergio P Perez, and Andrew Fitzgibbon. Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance. *arXiv preprint arXiv:2107.02027*, 2021.

[32] Mostafa Dehghani, Basil Mustafa, Josip Djolonga, Jonathan Heek, Matthias Minderer, Mathilde Caron, Andreas Steiner, Joan Puigcerver, Robert Geirhos, Ibrahim M Alabdulmohsin, et al. Patch n'pack: Navit, a vision transformer for any aspect ratio and resolution. *Advances in Neural Information Processing Systems*, 36, 2024.

[33] PaddleNLP Contributors. Paddlenlp: An easy-to-use and high performance nlp library. `https://github.com/PaddlePaddle/PaddleNLP`, 2021.

[34] BYTEDANCE INC. Effective transformer. `https://github.com/bytedance/effective_transformer`, 2021.

[35] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[36] Amanda Bertsch, Maor Ivgi, Uri Alon, Jonathan Berant, Matthew R Gormley, and Graham Neubig. In-context learning with long-context models: An in-depth exploration. *arXiv preprint arXiv:2405.00200*, 2024.

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

[38] 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.

[39] Markus N Rabe and Charles Staats. Self-attention does not need o (n2) memory. *arXiv preprint arXiv:2112.05682*, 2021.

[40] Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference, 2023.

[41] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Hanyu Dong, and Yu Wang. Flashdecoding++: Faster large language model inference on gpus. *arXiv preprint arXiv:2311.01282*, 2023.

[42] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.

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

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

[45] 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. *arXiv preprint arXiv:2308.14508*, 2023.

# A Appendix / supplemental material

## A.1 Algorithm Details

The detail implementation of FlashMask Backward Pass is presented in Algorithm 2. We do precomputations of max and min values of **FMS** and **FME** similar to the Forward Pass. Then the $\mathbf{FMS}_j$ and $\mathbf{FME}_j$ can be loaded to SRAM outside the inner loop (line 14-15), reducing the HBM accesses to $2 \times N$. Then, we do inner loop on $Q_i$ (line 16), computing the two valid parts and bypassing the masked part $i \in (\left\lfloor \frac{\mathbf{maxFMS}_j}{B_r} \right\rfloor, \left\lfloor \frac{\mathbf{minFME}_j}{B_r} \right\rfloor)$.

---

**Algorithm 2** Optimized Backward Pass with FlashMask

---

**Require:** Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dO} \in \mathbb{R}^{N \times d}$ in HBM, vector $L \in \mathbb{R}^N$ in HBM, block sizes $B_c$, $B_r$, dense bias mask $D \in \mathbb{R}^{N \times N}$, column-wise sparse mask starting rows $\mathbf{FMS} \in \mathbb{R}^N$, ending rows $\mathbf{FME} \in \mathbb{R}^N$.

1: Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
2: Divide $\mathbf{O}$ into $T_r$ blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide $\mathbf{dO}$ into $T_r$ blocks $\mathbf{dO}_i, \dots, \mathbf{dO}_{T_r}$ of size $B_r \times d$ each, and divide $L$ into $T_r$ blocks $L_i, \dots, L_{T_r}$ of size $B_r$ each.
3: Initialize $\mathbf{dQ} = (0)_{N \times d}$ in HBM and divide it into $T_r$ blocks $\mathbf{dQ}_1, \dots, \mathbf{dQ}_{T_r}$ of size $B_r \times d$ each. Divide $\mathbf{dK}, \mathbf{dV} \in \mathbb{R}^{N \times d}$ in to $T_c$ blocks $\mathbf{dK}_1, \dots, \mathbf{dK}_{T_c}$ and $\mathbf{dV}_1, \dots, \mathbf{dV}_{T_c}$, of size $B_c \times d$ each.
4: Divide the dense mask $\mathbf{D}$ into $T_r \times T_c$ blocks $\mathbf{D}_{1,1}, \dots, \mathbf{D}_{T_r, T_c}$
5: Divide **FMS** into $T_c$ blocks $\mathbf{FMS}_1, \dots, \mathbf{FMS}_{T_c}$, and divide **FME** into $\mathbf{FME}_1, \dots, \mathbf{FME}_{T_c}$.
6: Precompute the max value $\mathbf{maxFMS}_1, \dots, \mathbf{maxFMS}_{T_c}$ for each $\mathbf{FMS}_1, \dots, \mathbf{FMS}_{T_c}$, write to HBM.
7: Precompute the max value $\mathbf{maxFME}_1, \dots, \mathbf{maxFME}_{T_c}$ for each $\mathbf{FME}_1, \dots, \mathbf{FME}_{T_c}$, write to HBM.
8: Precompute the min value $\mathbf{minFMS}_1, \dots, \mathbf{minFMS}_{T_c}$ for each $\mathbf{FMS}_1, \dots, \mathbf{FMS}_{T_c}$, write to HBM.
9: Precompute the min value $\mathbf{minFME}_1, \dots, \mathbf{minFME}_{T_c}$ for each $\mathbf{FME}_1, \dots, \mathbf{FME}_{T_c}$, write to HBM.
10: Compute $D = \mathrm{rowsum}(\mathbf{dO} \circ \mathbf{O}) \in \mathbb{R}^d$ (pointwise multiply), write $D$ to HBM and divide it into $T_r$ blocks $D_1, \dots, D_{T_r}$ of size $B_r$ each.
11: **for** $1 \le j \le T_c$ **do**
12: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
13: Initialize $\mathbf{dK}_j = (0)_{B_c \times d}, \mathbf{dV}_j = (0)_{B_c \times d}$ on SRAM.
14: Load $\mathbf{FMS}_j$ from HBM to on-chip SRAM.
15: Load $\mathbf{FME}_j$ from HBM to on-chip SRAM.
16: **for** $1 \le i \le \left\lfloor \frac{\mathbf{maxFMS}_j}{B_r} \right\rfloor$ *and* $\left\lfloor \frac{\mathbf{minFME}_j}{B_r} \right\rfloor \le i \le T_r$ **do**
17: Load $\mathbf{Q}_i, \mathbf{O}_i, \mathbf{dO}_i, \mathbf{dQ}_i, L_i, D_i$ from HBM to on-chip SRAM.
18: On chip, compute $\mathbf{S}_i^{(j)} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
19: On chip, set $\mathbf{S}_i^{(j)} = \mathbf{S}_i^{(j)} + D_{i,j}$
20: **if** $\left\lfloor \frac{\mathbf{maxFME}_j}{B_r} \right\rfloor \le i \le \left\lfloor \frac{\mathbf{minFMS}_j}{B_r} \right\rfloor$ **then**
21: On chip, set $\mathbf{S}_i^{(j)}[x][y] = -\infty$, for every $i * B_r + x \ge M_j[y]$.
22: **end if**
23: On chip, compute $\mathbf{P}_i^{(j)} = \exp(\mathbf{S}_{ij} - L_i) \in \mathbb{R}^{B_r \times B_c}$.
24: On chip, compute $\mathbf{dV}_j \leftarrow \mathbf{dV}_j + (\mathbf{P}_i^{(j)})^\top \mathbf{dO}_i \in \mathbb{R}^{B_c \times d}$.
25: On chip, compute $\mathbf{dP}_i^{(j)} = \mathbf{dO}_i \mathbf{V}_j^\top \in \mathbb{R}^{B_r \times B_c}$.
26: On chip, compute $\mathbf{dS}_i^{(j)} = \mathbf{P}_i^{(j)} \circ (\mathbf{dP}_i^{(j)} - D_i) \in \mathbb{R}^{B_r \times B_c}$.
27: Load $\mathbf{dQ}_i$ from HBM to SRAM, then on chip, update $\mathbf{dQ}_i \leftarrow \mathbf{dQ}_i + \mathbf{dS}_i^{(j)} \mathbf{K}_j \in \mathbb{R}^{B_r \times d}$, and write back to HBM.
28: On chip, compute $\mathbf{dK}_j \leftarrow \mathbf{dK}_j + {\mathbf{dS}_i^{(j)}}^\top \mathbf{Q}_i \in \mathbb{R}^{B_c \times d}$.
29: **end for**
30: Write $\mathbf{dK}_j, \mathbf{dV}_j$ to HBM.
31: **end for**
32: Return $\mathbf{dQ}, \mathbf{dK}, \mathbf{dV}$.

---

## A.2 Supplementary Experimental Details

All end-to-end training and testing in this paper were conducted on 4 servers, each equipped with 32 NVIDIA A800-SXM 80G GPUs. We comprehensively evaluated the performance of the LLaMA2 model across three different parameter scales, four downstream task scenarios, and various sequence lengths. Given the diversity of experimental combinations and the specific distributed parallel strategies required by models, in varying parameter scales, the primary goal of the experiments is not to achieve optimal end-to-end training performance but to demonstrate the effectiveness of the FlashMask method. Therefore, to ensure consistency, we set the following hyperparameters in Table 1 with the same hardware configuration.

Table 1: Training Hyperparameters for Various Scales of LLaMA2 Models.

| **Model** | **LLaMA2-7B** | **LLaMA2-13B** | **LLaMA2-70B** |
|---|---|---|---|
| **Global Batch Size** | 16 | 16 | 16 |
| **Gradient Accumulation Step** | 2 | 4 | 16 |
| **Sharding Stage1 Degree** | 8 | 4 | 1 |
| **Tensor Parallel Degree** | 4 | 4 | 8 |
| **PipeLine Parallel Degree** | 1 | 2 | 4 |
| **Sequence Parallel Degree** | ✓ | ✓ | ✓ |

To verify the representativeness of our synthetic dataset, sparsity distribution histograms of synthetic dataset are presented in Figure 6. Then we use InToken method with max sequence length of 16K, 32K, 64K, and 128K on the open-source dataset LongBench, and compute the distribution histograms, presented in Figure 7. Note that many long sentences are truncated for max sequence length 16K, and 32K. Results indicate that the sparsity distributions of LongBench dataset and synthetic dataset are similar.

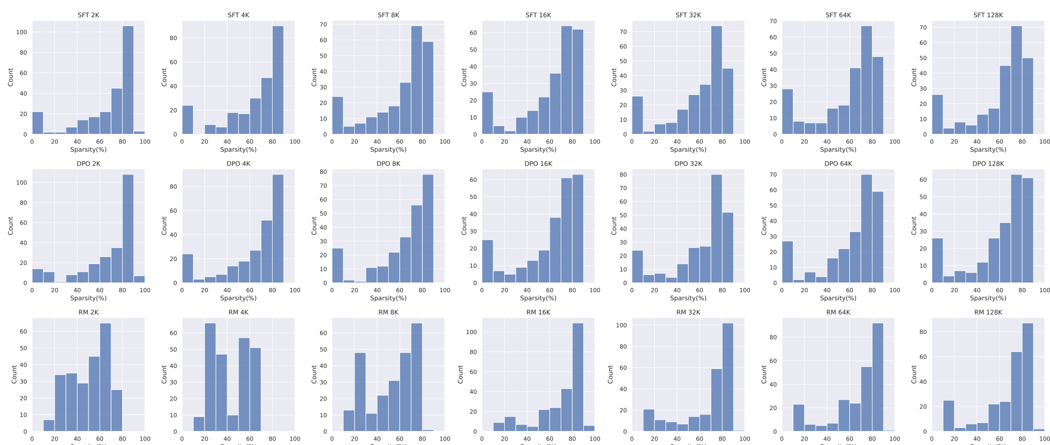



Figure 6: Sparsity Distribution of Synthetic Dataset.

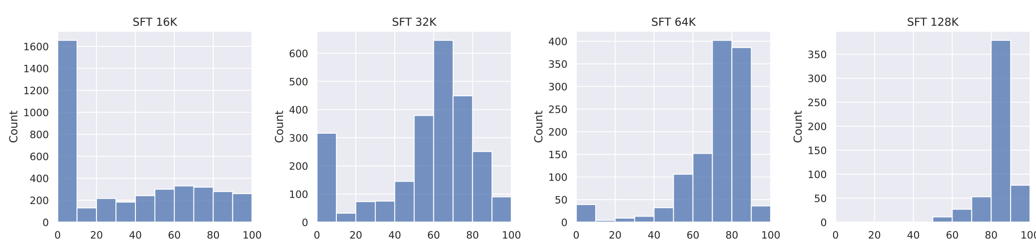



Figure 7: Sparsity Distribution of LongBench Dataset.

## A.3 Full Experiment Results

Kernel experiments are also conducted on batch sizes 4, and 8. FA-Varlen is excluded by default. Results are presented in Figure 8 and 9. The trends are identical to Figure 3 in Section 4.3, except memory exhaustion occurred with less sequence length, especially for FA-DenseMask and Vanilla Attention which require $O(N^2)$ memory to launch.

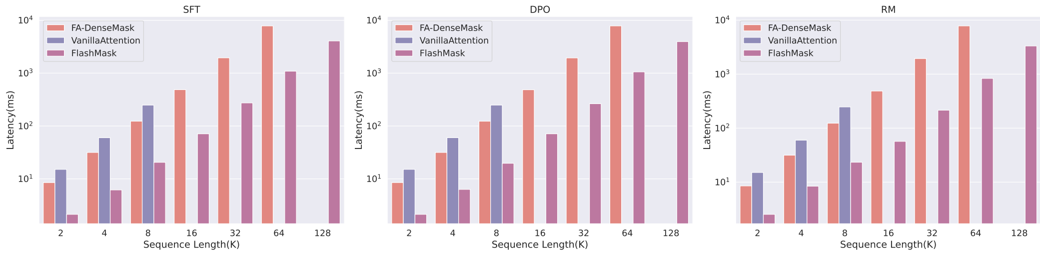



Figure 8: Kernel Latency Comparison with Varying the Length of Sequence.(Batch Size = 4)

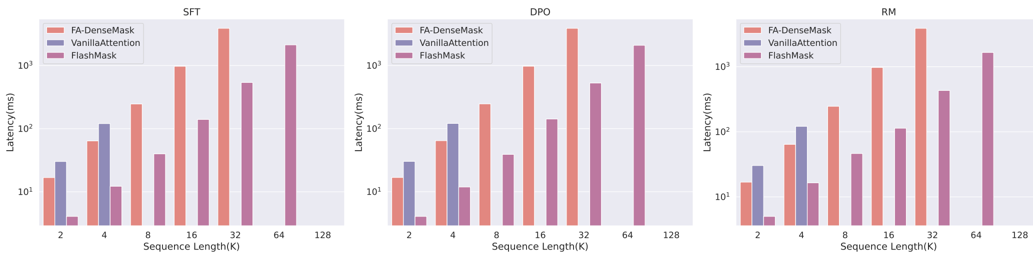



Figure 9: Kernel Latency Comparison with Varying the Length of Sequence. (Batch Size = 8)

We evaluate the effectiveness of FlashMask on the open-source dataset LongBench. The throughput of LoRA fine-tuning for LLaMA2-7B are shown in Figure 10. FlashMask performed close to FA-Varlen, showcasing 4.12x faster than FA-DenseMask, proving that FlashMask can deliver significant training accelerations in generalized real-world scenarios.

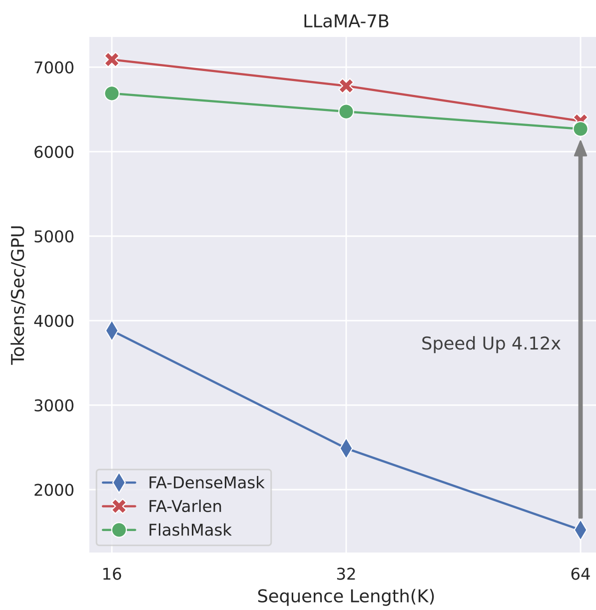
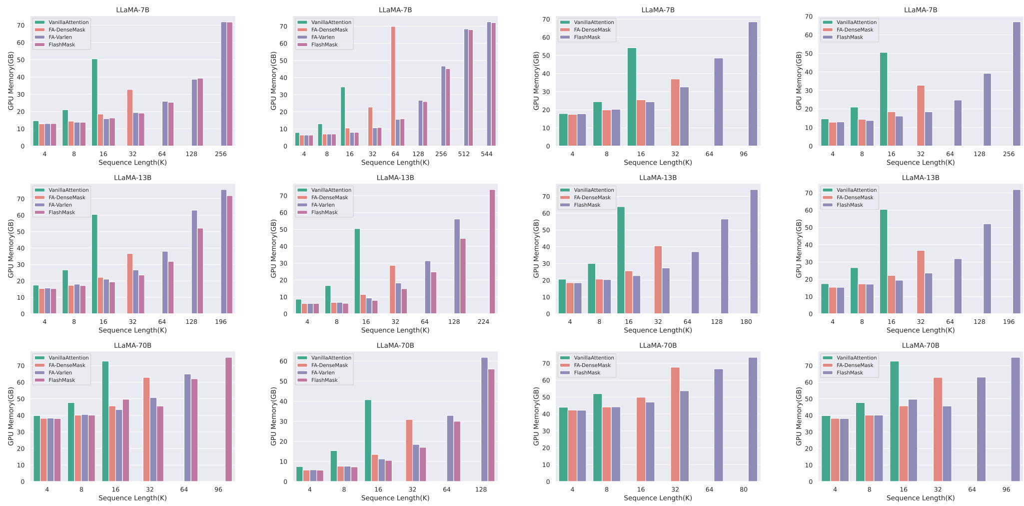



Figure 10: Comparison of End-to-End Training Throughput on LongBench Dataset.

Figure 11 presents the GPU memory consumption in End-to-End training. FlashMask showed linear memory consumption with increasing sequence length, far less than FA-DenseMask. Therefore, FlashMask supports training with much longer sequences in memory limits of 80G.




(a) SFT (b) LoRA (c) DPO (d) RM

Figure 11: Comparison of End-to-End Training GPU Memory.

# 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: This paper's contributions and scope are described in Abstract and Introduction.

   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 paper includes a discussion section about limitations.

   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: [Yes]

Justification: In sec 3.3 Complexity Analysis

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 source code will be public available and can reproduce the results according README.

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 source code will be public 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, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Refer to Experiments section.

   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: All our experimental results are run multiple times and then averaged.

   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: Referring to the Experiments section, we provide a running environment.

   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: We follow the NeurIPS Code of Ethics properly.

   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: There is no societal impact of the work performed.

    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: The paper poses no such risks.

    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: CC BY-NC-ND 4.0

    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: The paper does not 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: The 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: The 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.