# FlashMask-3: Efficient and Expressive Mask-Aware Distributed Attention

**Guoxia Wang**[*], **Qianyue He**[*], **Siming Wu**[*], **Haoyang Xie**[*]
**Qiwen Bao**, **Jinle Zeng**, **Jiabin Yang**, **Tian Wu**, **Dianhai Yu**[†]

Baidu Inc.
{wangguoxia, heqianyue, wusiming, xiehaoyang, yudianhai}@baidu.com

## Abstract

Long-context training of large language and multimodal models couples expressive structured attention masks (document, prefix-LM, share-question, sliding-window, block-sparse) with multi-node context parallelism, exposing the mask as a dominant bottleneck. The column-wise sparse mask of FlashMask compresses such masks from $O(n^2)$ to $O(n)$ via four query-axis boundary vectors and covers a broad family of structured patterns. Implemented for Ampere GPUs, FlashMask does not exploit FlashAttention-3's TMA/WGMMA warp-specialized pipeline on Hopper GPUs, and offers no distributed support. We observe that the column-wise sparse mask is indexed by key positions but stores query-axis interval boundaries. Under any multi-chunk context-parallel assignment, this property reduces per-rank mask localization to a provably zero-communication *clip–shift–max* on the boundary vectors, directly compatible with the single-GPU kernel. Building on this property, FlashMask-3 introduces a mask-aware Hopper kernel with a three-role warp specialization atop the FlashAttention-3 TMA/WGMMA pipeline, together with a mask-aware distributed extension: zero-communication mask localization, a sparsity-aware load balancer (HCC-IPO), and a mask-aware compute-communication overlap with mask-driven communication pruning, whose forward all-gather additionally adopts topology-aware hierarchical routing. On a single H100 GPU across 12 structured masks at context lengths from 4K to 128K, FlashMask-3 improves achieved TFLOPs over FlashMask, FlexAttention, and MagiAttention by 45% to 141%, 34% to 71%, and 2% to 44%, respectively. Under context-parallel degrees from 4 to 32 with 8K sequence per device, it attains 31% to 75% compute-communication overlap efficiency and improves throughput over Megatron-Core CP (ring), Megatron-Core CP (all-gather) and MagiAttention by $1.80\times$ to $3.71\times$, $1.71\times$ to $2.92\times$ and $1.09\times$ to $1.46\times$ respectively. In end-to-end training of a 118B-A14B MoE model at a 128K context on 128 H100 GPUs, it raises throughput by 7.3% to 11.0% over a Dual Chunk all-gather baseline, with attention forward+backward 24.5% to 44.9% faster. The source code is available at `https://github.com/PaddlePaddle/FlashMask`.

## 1 Introduction

Long-context training of large language and multimodal models increasingly relies on structured attention masks, including document masks [Dao et al., 2022], prefix-LM masks [Wang et al., 2026], share-question masks [Deng et al., 2025], sliding-window masks [Beltagy et al., 2020], and block-sparse patterns [Child et al., 2019, Yuan et al., 2025, Lu et al., 2025], to encode training-data semantics

[*]Equal contribution.
[†]Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

within a single packed batch. As context grows toward hundreds of thousands of tokens and training scales to multi-node context parallelism [Liu et al., 2023, Jacobs et al., 2023, Narayanan et al., 2021], masks become a first-class bottleneck: mask materialization, mask-aware kernel scheduling, and mask-induced load imbalance [Jiang et al., 2025, Tao and Huang, 2025] come to dominate step time. Programmable attention compilers [He et al., 2024] handle arbitrary masks but trail FlashAttention-3 (FA3) [Shah et al., 2024] in Hopper throughput; FA3 itself first-classes only causal, full, and varlen-document masks. The column-wise sparse representation of FlashMask [Wang et al., 2024] fills the expressiveness gap by encoding masks through four $O(n)$ boundary vectors, covering 12+ structured patterns within an exact tiled kernel [Dao et al., 2022, Dao, 2023]. However, FlashMask was implemented on Ampere GPUs and predates FA3: it exploits neither Hopper's TMA/WGMMA warp-specialized pipeline [Shah et al., 2024] nor any form of distributed context parallelism [Liu et al., 2023, Brandon et al., 2023, Narayanan et al., 2021].

We observe a structural property that resolves the distributed gap. Because the column-wise sparse mask is indexed by key positions but stores query-axis intervals [Wang et al., 2024], per-rank mask localization under any multi-chunk CP assignment reduces to an elementwise *clip–shift–max* on the boundary vectors, requiring no cross-rank communication and producing a rank-local tensor directly compatible with the single-GPU kernel (Theorem 1). Building on this property, we present FlashMask-3,[3] which organizes mask-awareness along two axes: (i) a mask-aware Hopper kernel with three-role warp specialization integrating column-wise sparse scanning into the FA3 TMA/WGMMA pipeline (§3.1); and (ii) a mask-aware distributed extension combining zero-communication mask localization (§3.2), a sparsity-aware hierarchical load balancer(§3.3), and a mask-aware compute-communication overlap [Chang et al., 2024, NVIDIA, 2023] with topology-aware hierarchical routing and mask-driven communication pruning (§3.4).

Our main contributions are as follows:

- **Mask-aware Hopper kernel.** A three-role warp specialization (KV Index Generator, KV Loader, MMA Consumer) integrates column-wise sparse mask processing into the asynchronous TMA/WGMMA pipeline at near-zero overhead, complemented by a preemptive persistent-tile scheduler that restores SM-level balance under irregular masks.
- **Mask-aware distributed extension.** A zero-communication *clip–shift–max* mask localization with provable correctness; a sparsity-aware hierarchical load balancer (HCC-IPO); and a mask-aware compute-communication overlap with mask-driven communication pruning, whose forward all-gather uses topology-aware hierarchical routing and whose backward reduce-scatter is overlapped with attention recomputation through bitmask-AMO (Atomic Memory Operations) synchronization.
- **Empirical evaluation.** On a single H100 GPU across 12 structured masks at context lengths from 4K to 128K, FlashMask-3 improves achieved TFLOPs over FlashMask, FlexAttention, and MagiAttention by 45% to 141%, 34% to 71%, and 2% to 44%, respectively. Under context-parallel degrees from 4 to 32 with 8K sequence per device, it attains 31% to 75% compute-communication overlap efficiency and improves throughput over Megatron-Core CP (ring), Megatron-Core CP (all-gather) and MagiAttention by $1.80\times$ to $3.71\times$, $1.71\times$ to $2.92\times$ and $1.09\times$ to $1.46\times$ respectively. In end-to-end training of a 118B-A14B MoE model at a 128K context on 128 H100 GPUs (CP degree 16), our distributed extension raises the training throughput by 7.3% to 11.0% over the same system using Dual Chunk balancing and all-gather communication.

## 2 Background

### 2.1 FlashAttention-3 on Hopper GPUs

The FlashAttention family [Dao et al., 2022, Rabe and Staats, 2021, Milakov and Gimelshein, 2018, Dao, 2023] computes exact attention $O = \mathrm{softmax}(QK^\top/\sqrt{d})V$ without materializing the $n \times n$ score matrix, through tiling and online softmax. FlashAttention-3 [Shah et al., 2024] targets Hopper GPUs by adding asynchronous TMA loads, WGMMA instructions, and warp-specialized producer–consumer pipelines: a producer warp group issues TMA loads while consumer warp groups execute WGMMA, coordinated through a small set of cross-warp barriers. FlashMask-3 builds on this

[3]The name aligns with FlashAttention-3, on which our Hopper kernel is built, mirroring how FlashMask [Wang et al., 2024] built on FlashAttention-2 [Dao, 2023].

asynchronous backbone, inserting mask processing as an additional warp-specialized stage rather than inline branches in the main compute loop (§3.1).

### 2.2 Column-wise sparse mask representation

FlashMask [Wang et al., 2024] encodes structured attention masks from the perspective of key tokens. The score matrix is split along the diagonal into lower-triangular and upper-triangular regions, and the mask at each key $j$ is encoded by four boundary vectors $\mathrm{LTS}, \mathrm{LTE}, \mathrm{UTS}, \mathrm{UTE} \in \mathbb{Z}^n_{\geq 0}$, with the masked query rows at key $j$ given by $[\mathrm{LTS}_j, \mathrm{LTE}_j) \cup [\mathrm{UTS}_j, \mathrm{UTE}_j)$. The tensor is *indexed by key positions* but *stores query-axis values*, a structural asymmetry that we exploit for zero-communication distributed mask localization in §3.2. Inside a tiled kernel, blockwise max/min along the key axis classifies each query–key tile as fully-masked, partially-masked, or unmasked, supporting tile skipping (§3.1) and the cost model in §3.3.

Two alternative routes target masks beyond the FlashAttention-native subset. FlexAttention [He et al., 2024] JIT-compiles an arbitrary user-defined `mask_mod` function via a Triton-based DSL [Tillet et al., 2019]; the compiled kernel handles arbitrary mask logic but follows a code path separate from FA3 and incurs per-mask compilation cost. MagiAttention [Tao and Huang, 2025] introduces an AttnSlice formulation paired with a dedicated Flex-Flash-Attention (FFA) kernel that natively supports arbitrary mask types; its distributed extension materializes per-rank mask structure via slice-cutting operations. In contrast, FlashMask-3 retains the $O(n)$ column-wise sparse representation and reduces per-rank mask localization to a *clip–shift–max* on the boundary vectors at zero cross-rank communication (§3.2), with no per-rank slice materialization or kernel recompilation.

### 2.3 Context parallelism and load balancing

Context parallelism (CP) shards the query sequence across $P$ ranks; each rank computes attention for its local queries against the global $K, V$ stream exchanged across ranks. Ring Attention [Liu et al., 2023] circulates $K, V$ around a device ring overlapped with blockwise attention; Striped Attention [Brandon et al., 2023] interleaves tokens for causal-load balance; DeepSpeed Ulysses [Jacobs et al., 2023] converts sequence parallelism into head parallelism via all-to-all; Megatron CP [Narayanan et al., 2021] pairs early and late query chunks (*dual chunking*) for balanced causal workload and obtains global $K, V$ through a forward all-gather and a backward reduce-scatter. Under expressive masks, the monotonic-load assumption underlying dual chunking breaks, coupling two further challenges: balancing mask-induced sparse compute across ranks, and overlapping forward all-gather and backward reduce-scatter with mask-aware computation, both addressed in §3.

### 2.4 Compute–communication overlap

Compute–communication overlap operates at two levels. At the kernel-stream level, FLUX [Chang et al., 2024] fuses all-gather and reduce-scatter into GEMM tiles via signal-based dependencies, achieving near-complete overlap for linear layers; Ring Attention [Liu et al., 2023] overlaps $K, V$ circulation with blockwise attention via dual CUDA streams. Extending tile-level fusion to attention is non-trivial because online softmax requires per-tile rescaling that degrades throughput upon splitting. At the topology level, NVSHMEM [NVIDIA, 2023] exposes GPU-initiated one-sided put/get over symmetric memory, enabling fine-grained overlap without CPU staging. These mechanisms form the substrate on which §3.4 builds a mask-aware overlap that prunes communication of fully-masked KV chunks and tracks per-segment readiness via bitmask-AMO synchronization.

A comprehensive discussion of additional related work is provided in Appendix B.

## 3 FlashMask-3

FlashMask-3 takes a column-wise sparse mask alongside $(Q, K, V)$ tensors, with the same mask schema and the same kernel binary serving both single-GPU and query-sharded context-parallel execution. Mask-awareness propagates through four layers of the execution stack via four cooperating mechanisms (§3.1–§3.4), all reusing the same full/partial/unmasked block classification off the column-wise sparse boundary vectors at kernel runtime, host-side preprocessing, and communication-side filtering. Figure 1 summarizes the cooperation.

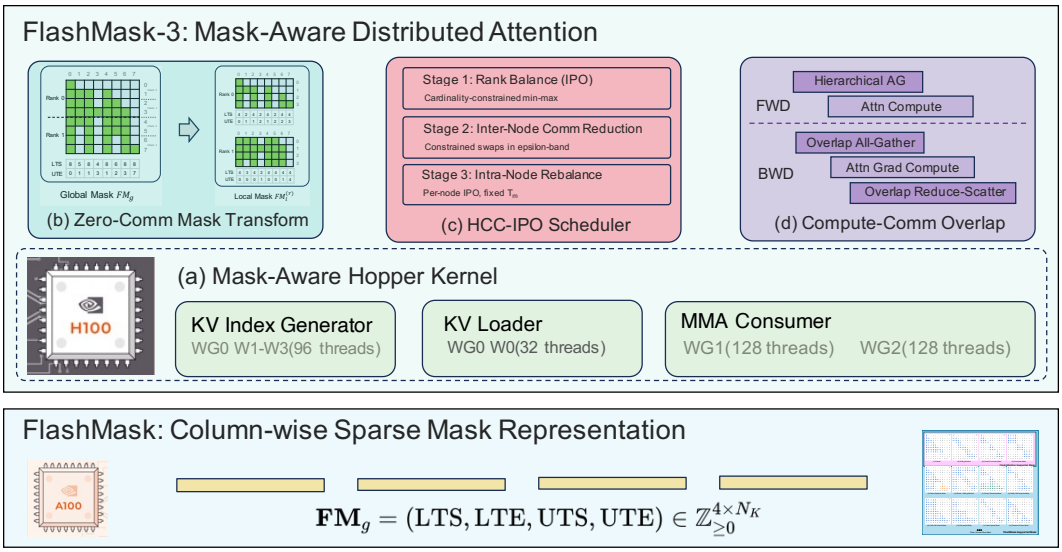



Figure 1: Overview of FlashMask-3 across four layers of the execution stack.

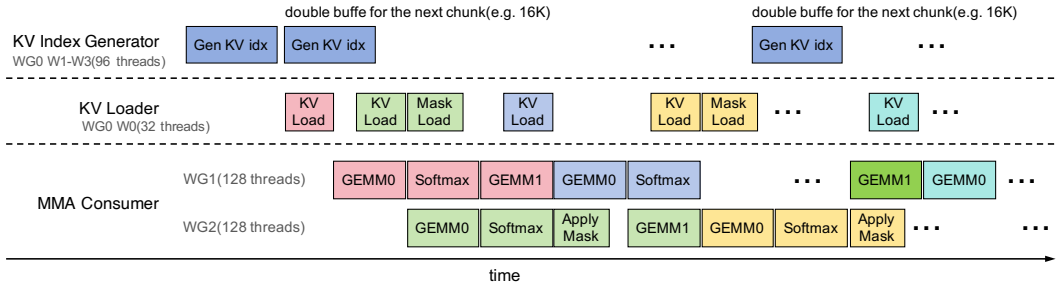



Figure 2: Three-role warp specialization in the FlashMask-3 Hopper kernel: KV Index Generator (mask scan), KV Loader (TMA load), and MMA Consumer (WGMMA score), connected by three asynchronous pipelines.

### 3.1 Hopper-optimized mask-aware kernel

On Hopper, FA3 overlaps TMA-issued KV loads with WGMMA score and online softmax via two warp-group roles [Shah et al., 2024]; inserting column-wise sparse mask scanning into either role would stall the TMA queue or introduce a barrier inside the WGMMA critical path. We observe that FA3's Producer warp group dedicates 128 threads to TMA issue while a single thread suffices to launch each op, leaving 96 threads idle every iteration; FlashMask-3 reclaims these threads as a third warp-specialized role, the *KV Index Generator*, which scans the column-wise sparse boundaries described in §2.2 and emits a queue of valid blocks classified as fully-masked, partially-masked, or unmasked via the blockwise max/min of $\{\mathrm{LTS}, \mathrm{LTE}, \mathrm{UTS}, \mathrm{UTE}\}$. The KV Loader consumes the queue and issues TMA loads only for listed blocks; the MMA Consumer runs the WGMMA score path, applying a row-interval mask only on partially-masked blocks (Figure 2). The same classification feeds host-side preprocessing in §3.3 and communication-side filtering in §3.4 on independent code paths.

Beyond warp specialization, irregular masks make per-$Q$-block valid-$K$-block counts vary by an order of magnitude, leaving SMs that own long-tail $Q$-blocks straggling under static tile-to-SM dispatch. We adopt a *Preemptive Persistent Tile (PPT) scheduler* in which each CTA atomically claims its next $Q$-block from a global counter, with the KV Index Generator issuing the atomic op so the claim overlaps with mask scanning and stays off the WGMMA critical path.

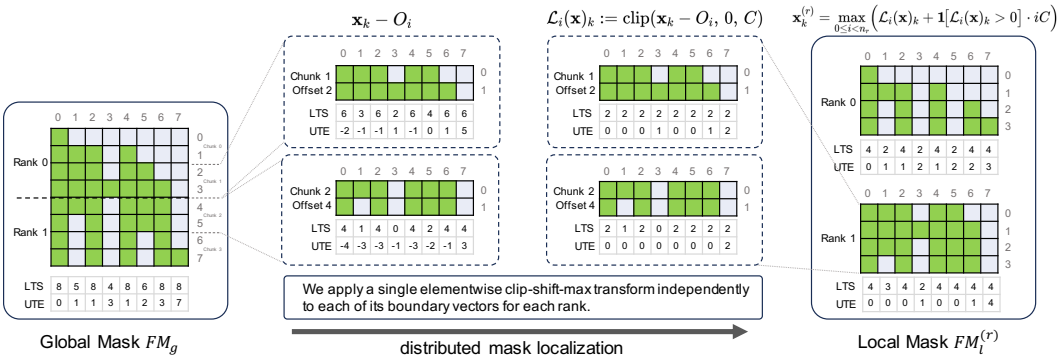



Figure 3: Zero-communication mapping of global mask boundaries (LTS, UTE) to rank-local coordinates using the clip–shift–max transform (Eq. 1).

### 3.2 Zero-communication distributed mask transform

We localize the global column-wise sparse mask $\mathbf{FM}_g = (\mathrm{LTS}, \mathrm{LTE}, \mathrm{UTS}, \mathrm{UTE}) \in \mathbb{Z}_{\ge 0}^{4 \times N_K}$ [Wang et al., 2024], where for each key $k$, the masked query intervals $[\mathrm{LTS}_k, \mathrm{LTE}_k) \cup [\mathrm{UTS}_k, \mathrm{UTE}_k)$ are defined by the lower- and upper-triangular pairs. To distribute this global mask to each rank without communication, we apply an elementwise *clip–shift–max* transform independently to each boundary vector. Suppose rank $r$ owns $n_r$ disjoint query chunks $\{[O_i, O_i + C)\}_{i=0}^{n_r - 1}$, sorted by global offsets $O_0 < O_1 < \cdots < O_{n_r - 1}$, with chunk size $C$ set by the upstream scheduler. For any global boundary vector $\mathbf{x} \in \mathbb{Z}_{\ge 0}^{N_K}$, we define the single-chunk clip as $\mathcal{L}_i(\mathbf{x})_k := \mathrm{clip}(\mathbf{x}_k - O_i,\, 0,\, C)$. The rank-local boundary vector $\mathbf{x}^{(r)}$ is then

$$\mathbf{x}_k^{(r)} = \max_{0 \le i < n_r} \Big( \mathcal{L}_i(\mathbf{x})_k + \mathbf{1}\big[\mathcal{L}_i(\mathbf{x})_k > 0\big] \cdot iC \Big), \qquad k \in [0, N_K). \tag{1}$$

Breaking down (1): (1) $\mathcal{L}_i$ maps the global boundary to chunk $i$'s local $[0, C)$ coordinate and clips out-of-chunk values; (2) $\mathcal{L}_i + iC$ embeds chunk $i$'s local values into the rank-local concatenated space $\bigsqcup_{i=0}^{n_r - 1} [iC, (i{+}1)C)$, preserving the global chunk order; (3) the indicator ensures only non-empty clips are shifted, avoiding pollution from vacuous chunks; (4) $\max_i$ aggregates across all $n_r$ chunks, ensuring only the chunk containing the boundary value $\mathbf{x}_k$ contributes. Figure 3 illustrates this *clip–shift–max* process for representative LTS and UTE boundary vectors.

Theorem 1 in Appendix C.1 shows that the rank-local boundary intervals exactly match the global intervals at the corresponding global query positions. Thus, $\mathbf{FM}_l^{(r)}$ encodes the per-rank mask without information loss. The transform requires only the global mask $\mathbf{FM}_g$ and the rank-owned offsets $\{O_i\}_{i=0}^{n_r - 1}$, so there is no need for cross-rank communication. The output preserves the kernel interface described in §3.1, and any upstream chunk assignment (§3.3) is integrated through this single tensor-level interface.

### 3.3 Hierarchical Sparsity-Aware Compute-Communication Balancing

Achieving efficient distributed attention requires jointly optimizing both compute and communication loads, especially under structured sparsity. While balancing per-rank compute loads is critical to avoid stragglers, disregarding the communication topology can easily lead to excessive inter-node key-value (KV) traffic and increased overall latency. However, exact joint optimization is intractable due to the combinatorial search space and sparsity-induced irregularities.

To address this challenge, we propose a *Hierarchical Compute-Communication Iterative Pairwise Optimal (HCC-IPO)* scheduler, which alternates between: (i) inter-rank compute balancing (assigning chunks to ranks to minimize per-rank compute imbalance under strict cardinality constraints), (ii) inter-node communication reduction (iteratively swapping chunk assignments between nodes to minimize the maximum inter-node KV traffic while maintaining compute balance within an $\epsilon$-tolerance), and (iii) intra-node compute rebalancing (restoring compute load balance within each node, with node assignments fixed). Full algorithmic details and theoretical guarantees are provided in Appendix C.3.

Formally, let $N$ be the total number of chunks, $P$ the number of ranks, and $\kappa = N/P$ the per-rank chunk cardinality. For each chunk $i$, let $w_i$ denote its compute load(see Appendix C.2). A chunk-to-rank assignment is a mapping $\pi : [N] \to [P]$, where each rank $j$ is assigned chunk set $S_j(\pi) = \pi^{-1}(j)$ with total compute load $L_j(\pi) = \sum_{i \in S_j(\pi)} w_i$. The cardinality-constrained min-max partitioning problem is:

$$\min_{\pi} \max_{j \in [P]} L_j(\pi), \qquad \text{s.t.} \quad |S_j(\pi)| = \kappa \ \ \forall j, \tag{2}$$

A standard lower bound is $LB = \max\left(\left\lceil \frac{1}{P} \sum_i w_i \right\rceil, \max_i w_i\right)$. We employ the IPO method for this partitioning (Appendix C.3).

For communication balancing, we group ranks by node: let $\mathcal{R}_m$ denote the ranks on node $m$, and $T_m = \bigcup_{j \in \mathcal{R}_m} S_j$ the chunk set assigned to node $m$. Let $K_i$ denote the set of KV blocks accessed by chunk $i$, and $\mathrm{Local}(m)$ the set of KV blocks whose queries reside on node $m$. The inter-node KV traffic and the global communication bottleneck are defined as

$$\mathrm{Comm}(m, \pi) \ = \ \Big| \bigcup_{i \in T_m} K_i \ \setminus \ \mathrm{Local}(m) \Big|, \qquad \mathrm{Comm}_{\max}(\pi) \ = \ \max_{m} \mathrm{Comm}(m, \pi), \tag{3}$$

To reduce $\mathrm{Comm}_{\max}$, we identify $m_{\max}$ and $m_{\min}$ by their communication loads, enumerate feasible one-for-one chunk swaps ($i_a \in T_{m_{\max}}, i_b \in T_{m_{\min}}$) that preserve compute balance within $\epsilon$-tolerance (i.e., $|L_j(\pi') - \bar{L}| \le \epsilon \bar{L}$ for all affected ranks, with $\bar{L} = \frac{1}{P}\sum_i w_i$), and commit the swap that yields the largest reduction in $\mathrm{Comm}_{\max}$. After communication balancing, intra-node compute balance is restored by applying IPO within each node.

The resulting assignment $\pi$ achieves balanced compute loads, bounds the worst-case inter-node KV traffic, and is fully compatible with the distributed mask transformation described in Section 3.2. This hierarchical, sparsity-aware balancing enables scalable and efficient mask-aware distributed attention under realistic hardware constraints.

### 3.4 Mask-aware hierarchical compute-communication overlap

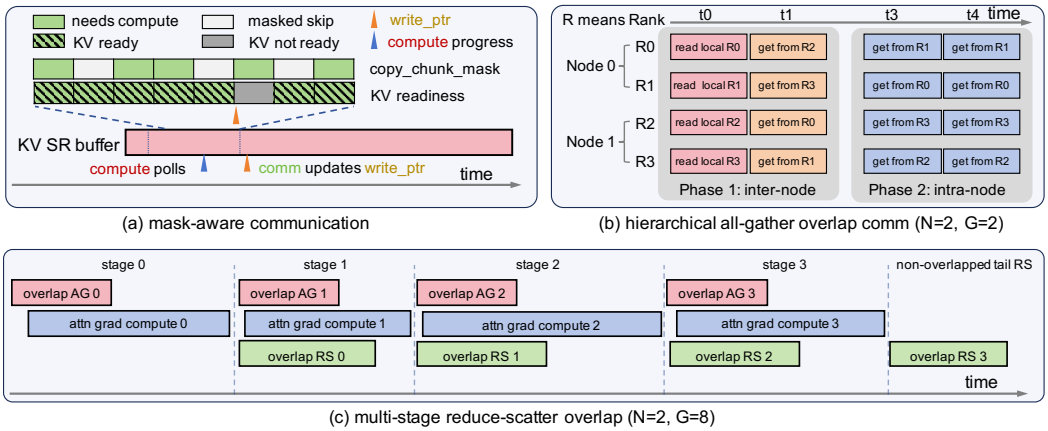
CP introduces overlap in two aspects: (i) the all-gather (AG) of $K, V$ in the forward pass (and backward recomputation), and (ii) the reduce-scatter (RS) of $\mathrm{d}K, \mathrm{d}V$ in the backward pass. FlashMask-3 hides both costs behind sparse attention computation via three mechanisms: (1) a sparse-mask-aware NVSHMEM symmetric substrate; (2) a hierarchical (intra-/inter-node) AG scheme *decoupled* from the attention kernel; and (3) a multi-stage backward pipeline interleaving AG, attention, and RS. Communication kernels occupy only 4–8 SMs on high-priority non-blocking streams. The overlap requires fewer than 20 lines of minimally invasive modification to the attention kernel. Detailed protocols are in Appendix C.4 and C.5; Figure 4 provides a timeline summary.

**Sparse-mask-aware NVSHMEM symmetric substrate.** A single *Send/Recv buffer* (SR buffer), sized for the complete post-AG $K, V$ tensors, is allocated in NVSHMEM symmetric memory. Any device can access any remote SR-buffer slot via one-sided RDMA `get`/`put`, and the attention kernel reads $K, V$ directly from this buffer without intermediate copies. The only unavoidable data movement is a one-time copy of the local $K, V$ shard into the SR buffer; the same buffer is reused for backward recomputation at no extra cost. Before each communication launch, a lightweight sparsity-check kernel scans the column-partitioned sparse boundaries (§3.2) to produce a binary `copy_chunk_mask` marking entirely masked-out KV chunks. The communication kernel skips these chunks, eliminating $\approx 50\%$ of AG/RS traffic in typical causal document and prefix-LM document masks in our experiments.

**Decoupled hierarchical all-gather.** The AG kernel runs on a high-priority stream while the attention kernel performs a single pass over the gathered $K, V$, without partitioning $K, V$ across communication steps, thereby avoiding the inter-partition online-softmax LSE correction overhead. The two kernels synchronize via a monotonically increasing *write-pointer frontier*; the KV Loader (§3.1) polls this pointer before each TMA load. To exploit the bandwidth asymmetry between RoCE and NVLink (5–10$\times$ higher per-GPU), the AG is routed via *congruence groups*:

$$\mathcal{C}(r) \ = \ \big\{\, r \bmod G + k \cdot G \ \big|\ k = 0, \ldots, N{-}1 \,\big\}, \tag{4}$$

where $r$ is the global device rank, $N$ the number of nodes, and $G{=}8$ GPUs per node. In **Phase 1**, each device issues $N{-}1$ RoCE `get` operations to fetch KV chunks from cross-node congruence-group members. In **Phase 2**, each device retrieves the remaining $(G{-}1){\cdot}N$ chunks from intra-node




(a) mask-aware communication

(b) hierarchical all-gather overlap comm (N=2, G=2)

(c) multi-stage reduce-scatter overlap (N=2, G=8)

Figure 4: Mask-aware hierarchical compute-communication overlap. **(a)** AG skips fully-masked chunks; compute polls a `write_ptr` updated by the comm kernel. **(b)** Hierarchical AG ($N$=2, $G$=2): inter-node congruence-group fetch (Phase 1), then intra-node NVLink fetch (Phase 2). **(c)** Multi-stage backward overlap ($N$=2, $G$=8): RS of segment $t$ overlaps AG and grad compute of segment $t$+1; the tail RS runs non-overlapped.

neighbors over NVLink, whose congruence-group data are already available. Letting $P = N \cdot G$, at $P$=32 inter-node pulls drop from $(N-1) \cdot G = 24$ to $N-1 = 3$, a $\approx$90% reduction.

**Multi-stage backward overlap.** The backward pass reuses the same hierarchical AG for $K, V$ recomputation, adopting KV column parallelism: each CTA processes one KV chunk and iterates over all $Q$ blocks, eliminating sequential dependencies across chunks. The contiguous *write-pointer frontier* is replaced by per-chunk `work_done` flags. The RS is realized as a sparse all-to-all via NVSHMEM one-sided `put`, synchronized by a bitmask-AMO mechanism for `put` producers and gradient reduction consumers, with both SM-level and rank-level load balanced. We partition the KV sequence into $T$ segments of $\mathcal{C} = 2^{\lfloor \frac{1}{2} \log_2 P \rfloor}$ chunks each (e.g., $T$=$\mathcal{C}$=4 for $P$=16), so the RS of segment $t$ overlaps with the AG and backward attention of segment $t$+1. Since forward-saved LSE eliminates cross-segment online-softmax rescaling, this partition incurs virtually no overhead beyond fp32 d$Q$ accumulation across segments.

## 4 Experiments

In this section, we evaluate FlashMask-3 through a comprehensive set of experiments designed to assess single-GPU kernel efficiency (§4.1), distributed scalability (§4.2), and end-to-end training throughput on a production-scale MoE workload (§4.3), followed by component-level ablations (§4.4). All experiments are conducted on machines equipped with NVIDIA H100 80GB SXM5 GPUs interconnected via NVLink within each node and RoCE across nodes, with 8 GPUs per node and Intel Xeon Platinum 8468V CPUs (48 cores). We use CUDA 12.4, CUTLASS (commit `afa177`) [Thakkar et al., 2023] and NVSHMEM 3.2.5 throughout. Baseline configurations are as follows: MagiAttention (commit `d97b72`), FlexAttention (PyTorch 2.8.0), and Megatron 2.8.0. Unless otherwise stated, all experiments use a head dimension of $d = 128$ and the BF16 data type. Detailed experimental configurations are provided in Appendix D.1.1. Speed measurements are repeated; error bars in the figures and ± values in the tables denote ±1 standard deviation.

### 4.1 Single-GPU Kernel Performance

We first isolate the kernel-level performance of FlashMask-3 by benchmarking on a single H100 GPU against FlashMask, FlexAttention, and MagiAttention. We sweep sequence lengths from 4K to 128K across 12 representative sparse mask patterns (see Equation 5 and Figure 5). The data construction method and experiment settings follows that described in Section A.5.2 [Wang et al., 2024]. Across this entire grid, FlashMask-3 improves achieved TFLOPs/s over FlashMask, FlexAttention, and MagiAttention by 45%–141%, 34%–71%, and 2%–44%, respectively, outperforming all three

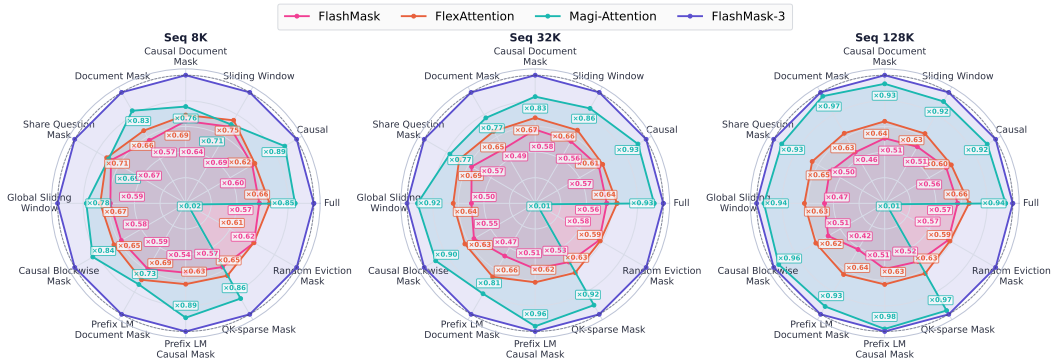



Figure 5: Kernel FWD & BWD throughput (TFLOPs/s) on a single H100 GPU. Error bars are omitted for legibility; the underlying per-mask measurements with standard deviations are reported in Figure 9 (Appendix D.1.3).

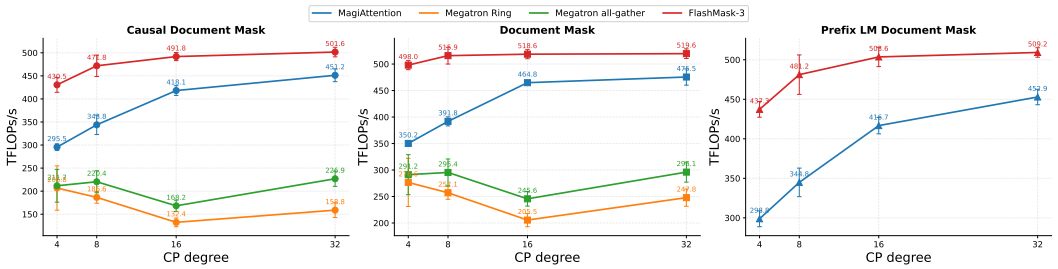



Figure 6: Distributed scaling. Per-GPU TFLOPs/s versus CP degree $P$ for FlashMask-3, Megatron all-gather, Megatron Ring Attention, and Magi-Attention on three practically relevant mask patterns.

baselines on every mask pattern and sequence length. To summarize these results into one figure per baseline, we report the geometric mean of speedups across all mask patterns and sequence lengths.

$$\text{Speedup} = \frac{V_{\text{TFLOPs/s, FlashMask-3}} - V_{\text{TFLOPs/s, baseline}}}{V_{\text{TFLOPs/s, baseline}}} \times 100\%. \tag{5}$$

The **82.6%** mean speedup over FlashMask validates the combined effect of our three-role warp-specialized pipeline, asynchronous TMA prefetching, Ping-Pong scheduling, and the PPT scheduler on Hopper. Against FlexAttention, the **55.3%** mean gain reflects the overhead of runtime mask evaluation via `torch.compile` and the absence of a Hopper-native execution pipeline. FlashMask-3 also surpasses MagiAttention by **15.0%** in geometric mean (excluding the Random Eviction mask, on which MagiAttention's block-sparse representation is ill-suited, resulting in over $50\times$ slowdown). Even on patterns favorable to block-sparse encoding, FlashMask-3 remains competitive or superior, demonstrating that the column-wise sparse representation achieves broad expressiveness without sacrificing efficiency. Across all comparisons, the speedup increases monotonically with sequence length (e.g., $+66.2\% \rightarrow +97.3\%$ over FlashMask from 8K to 128K), confirming that our architectural optimizations yield compounding benefits as attention becomes increasingly memory-bound. Detailed per-mask TFLOPs/s results are provided in Appendix D.1.3.

### 4.2 Context Parallelism Scalability

To evaluate the scalability of FlashMask-3 under context parallelism (CP), we compare against two representative baselines, namely Megatron-Core CP (both its *ring* and *all-gather* variants) and MagiAttention, on three practically relevant mask patterns: *Causal Document Mask*, *Document Mask*, and *Prefix LM Document Mask*. Since neither the ring nor the all-gather variant of Megatron-Core CP supports Prefix LM Document Mask, we exclude them from the corresponding comparison.

We adopt a weak-scaling protocol in which the per-device local sequence length is fixed at $S_{\text{local}}$=8K tokens and the CP degree is scaled from 4 to 32, yielding total sequence lengths ranging from 32K to 256K. All experiments use a batch size of $B$=$32/P$, $H_q$=64 query heads, and $H_{kv}$=1 key-value head; since $S$=$S_{\text{local}} \cdot P$ grows with $P$ while $B$ shrinks as $1/P$, the dense per-device attention

Table 1: End-to-end training throughput of a 118B-A14B MoE model (128K context, CP degree 16, $S_{\text{local}}$=8K tokens per device, $H_q$=64, $H_{kv}$=8, document mask) on 128 H100 GPUs, for three datasets of differing average mask sparsity. Each run covers 70 steps; we discard the 10 highest and 10 lowest throughput steps and report the mean ± one standard deviation over the remaining 50. $\Delta_{\text{attn}}$ is the corresponding attention forward+backward speedup over the baseline.

| | 70% sparsity | | 80% sparsity | | 86% sparsity | |
|---|---|---|---|---|---|---|
| Configuration | tokens/s/GPU | $\Delta_{\text{attn}}$ | tokens/s/GPU | $\Delta_{\text{attn}}$ | tokens/s/GPU | $\Delta_{\text{attn}}$ |
| Baseline (Dual Chunk + all-gather) | 712.6 ± 8.7 | – | 753.3 ± 34.7 | – | 812.4 ± 57.1 | – |
| + HCC-IPO balancing | 732.2 ± 24.5 | +10.4% | 786.2 ± 65.2 | +19.1% | 858.5 ± 62.1 | +10.5% |
| + mask-aware overlap | 750.8 ± 6.4 | +15.9% | 785.0 ± 33.6 | +13.9% | 860.0 ± 55.0 | +20.5% |
| + both, hierarchical routing off | 700.6 ± 28.8 | +8.4% | 742.0 ± 72.6 | +21.5% | 828.0 ± 60.6 | +23.2% |
| + both, w/o chunk-skip pruning | 750.0 ± 20.6 | +20.5% | 788.4 ± 55.0 | +25.4% | 888.6 ± 69.6 | +27.9% |
| **FlashMask-3 (balance + overlap)** | **764.7 ± 28.8** | **+24.5%** | **811.4 ± 75.9** | **+42.1%** | **902.1 ± 72.6** | **+44.9%** |

workload $BS^2/P$ is constant across CP degrees. The results are summarized in Figure 6. Additional configurations and complete forward/backward results are provided in Appendix D.1.2 and D.1.4.

At CP degrees 4 and 8, all communication is confined to intra-node NVLink interconnects, whereas scaling to CP 16 and 32 introduces inter-node traffic. Despite this transition, FlashMask-3 sustains strong scaling efficiency. On the Causal Document Mask, its throughput increases from 430.5 TFLOPs/s at CP = 4 to 471.8, 491.8, and 501.6 TFLOPs/s at CP = 8, 16, and 32, respectively. Across all supported mask patterns and CP degrees, FlashMask-3 achieves $1.09\times$ to $1.46\times$ higher throughput than MagiAttention, outperforms Megatron-Core CP (all-gather) by $1.71\times$ to $2.92\times$, and surpasses Megatron-Core CP (ring) by $1.80\times$ to $3.71\times$.

### 4.3 End-to-End Training Throughput

The preceding sections measure attention in isolation; we now report throughput for full training steps. We train a 118B-A14B MoE model at a 128K context on 128 H100 GPUs (16 nodes) with CP degree 16, a per-device local sequence length of $S_{\text{local}}$=8K tokens, $H_q$=64 query heads, $H_{kv}$=8 KV heads, and a document mask. The baseline is FlashMask-3 with Dual Chunk balancing and all-gather communication, i.e. with the balancer of §3.3 and the overlap of §3.4 disabled. Because the share of step time spent in attention depends on mask sparsity, we repeat the comparison on three datasets with average mask sparsity 70%, 80%, and 86%.

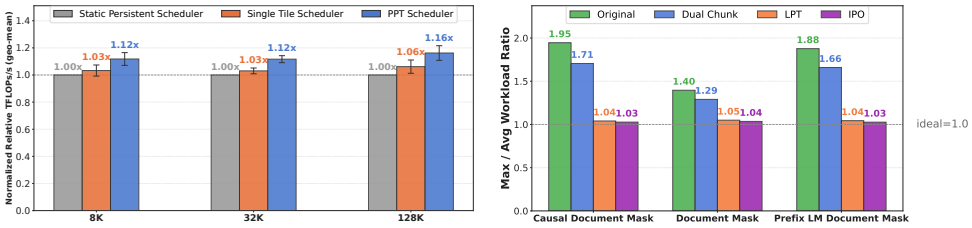
Table 1 reports gains of 7.3% / 7.7% / 11.0% in end-to-end throughput with both components enabled, at 70% / 80% / 86% sparsity, against attention forward+backward speedups of 24.5% / 42.1% / 44.9% (at 70% sparsity, 10900.5 ms per step down to 8757.6 ms). The end-to-end gain is smaller because attention accounts for only 43.5% / 25.8% / 24.4% of forward+backward compute time in the full configuration, and the end-to-end measurement also covers gradient synchronization and the optimizer update, which our method does not modify.

Each component helps on its own, contributing 2.8% / 4.4% / 5.7% (balancing) and 5.4% / 4.2% / 5.9% (overlap), and chunk-skip pruning is worth a further 2.0% / 2.9% / 1.5%. Hierarchical routing is required for the two to compose at this cross-node scale: with it disabled, balancing plus overlap falls to $-1.7\%$ / $-1.5\%$ / $+1.9\%$, below overlap alone, because the balancer spreads chunks across nodes and that traffic is no longer confined to congruence groups (§3.4). The default of 4 backward pipeline stages follows the rule of §3.4 at $P$=16: halving it to 2 costs up to 3.6%, while raising it to 8 yields no significant further gain.

FlashMask-3 computes exact attention: mask localization matches the global mask (Theorem 1), and the balancer, overlap, and pruning change only how work is placed, not what is computed. Convergence and downstream quality are therefore unaffected; Appendix D.3 compares the loss of an otherwise identical run without context parallelism.

### 4.4 Ablation Study

Complementing the end-to-end breakdown of §4.3, we conduct three groups of ablation experiments that isolate each key component proposed in FlashMask-3 across mask patterns, sequence lengths,




(a) (b)

Figure 7: (a) Speedup of the Preemptive Persistent Tile (PPT) scheduler over the static scheduler across different mask patterns and sequence lengths on a single H100 GPU. (b) Peak-to-average ratio of per-device computation time under different workload partitioning strategies with Context Parallelism.

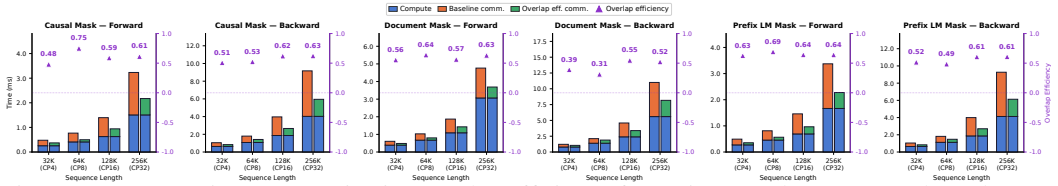



Figure 8: Computation-communication overlap efficiency for various mask patterns and CP degrees.

and CP degrees: *Preemptive Persistent Tile* (PPT), *load balancing*, and *computation-communication overlap*.

**Effect of Preemptive Persistent Tile (PPT).** We compare three tile-scheduling strategies on a single H100 GPU: (i) a *static scheduler* that assigns a fixed tile grid, (ii) a *single-tile scheduler* that dispatches one work tile at a time, and (iii) the proposed *PPT* scheduler. As shown in Figure 7(a), across 12 mask configurations at sequence lengths of 8K, 32K, and 128K, PPT consistently outperforms the static scheduler by up to 12%, demonstrating that preemptive, persistent tile execution effectively amortizes scheduling overhead and improves streaming multiprocessor (SM) utilization.

**Effect of load balancing.** We compare four workload partitioning strategies under CP: (i) a naïve *baseline* that evenly splits the KV sequence, (ii) *Dual Chunk*, (iii) *Longest-Processing-Time first* (LPT) [Graham, 1969], and (iv) our proposed *Iterative Pairwise Optimization* (IPO). We report the *peak-to-average ratio* of per-device computation time, where a value closer to 1.0 indicates a more balanced distribution. As illustrated in Figure 7(b), IPO achieves the lowest peak-to-average ratio across all three mask patterns, confirming that it further refines the LPT solution toward a more balanced workload assignment. A detailed visualization of per-rank compute load distributions under each strategy is provided in Figure 11 (Appendix).

**Effect of computation-communication overlap.** Following the methodology of Flux [Chang et al., 2024], we report the overlap efficiency defined as $E_{\text{overlap}} = 1 - T_{\text{exposed}}/T_{\text{comm}}$, where $T_{\text{exposed}}$ denotes the portion of communication time that is not hidden behind computation and $T_{\text{comm}}$ is the total communication time. A value closer to 1.0 indicates that computation more fully hides communication latency. As shown in Figure 8, FlashMask-3 attains an overlap efficiency ranging from 0.31 to 0.75, validating the effectiveness of our mask-aware hierarchical compute-communication.

## 5 Conclusion

FlashMask-3 establishes that the column-wise sparse mask representation, originally a single-GPU memory optimization, also serves as the right interface for distributing expressive masks at zero communication on Hopper-class hardware. The method preserves one column-wise sparse mask abstraction and one FA3-derived kernel across single-GPU and multi-node settings, with mask-awareness propagated through Hopper kernel scheduling, hierarchical load balancing, and compute-communication overlap. Empirical evaluation across 12 structured masks at context lengths up to 128K on a single H100 GPU and context parallelism up to $P$=32 yields consistent improvements in single-GPU TFLOPs, compute-communication overlap, and distributed throughput over representative single-GPU and distributed mask baselines. On end-to-end training of a 118B-A14B MoE model at a 128K context across 128 H100 GPUs, these mechanisms yield a 7.3%–11.0% throughput gain.

## Acknowledgments

We thank the anonymous reviewers, area chairs, and senior area chairs for their insightful comments and constructive suggestions.

This work was supported by the National Engineering Research Center of Deep Learning Technology and Application.

## References

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

William Brandon, Aniruddha Nrusimha, Kevin Qian, Zachary Ankner, Tian Jin, Zhiye Song, and Jonathan Ragan-Kelley. Striped attention: Faster ring attention for causal transformers. *arXiv preprint arXiv:2311.09431*, 2023.

Li-Wen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuan Zhang, Zuquan Song, Ziheng Jiang, Haibin Lin, Xin Jin, and Xin Liu. Flux: Fast software-based communication overlap on gpus through kernel fusion. *arXiv preprint arXiv:2406.06858*, 2024.

Sirui Chen, Jingji Chen, Siqi Zhu, Ziheng Jiang, Yanghua Peng, and Xuehai Qian. Mesh-attention: A new communication-efficient distributed attention with improved data locality. *arXiv preprint arXiv:2512.20968*, 2025.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. 2019.

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

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

Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, et al. Emerging properties in unified multimodal pretraining. *arXiv preprint arXiv:2505.14683*, 2025.

Jiarui Fang and Shangchun Zhao. Usp: A unified sequence parallelism approach for long context generative ai. *arXiv preprint arXiv:2405.07719*, 2024.

Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

Horace He, Driss Guessous, Yanbo Liang, and Joy Dong. Flexattention: The flexibility of pytorch with the performance of flashattention. *PyTorch Blog*, 8, 2024.

Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

Chenyu Jiang, Zhenkun Cai, Ye Tian, Zhen Jia, Yida Wang, and Chuan Wu. Dcp: Addressing input dynamism in long-context training via dynamic context parallelism. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pages 221–236, 2025.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E. Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. Distflashattn: Distributed memory-efficient attention for long-context llms training. *arXiv preprint arXiv:2310.03294*, 2023.

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

Tianyu Lu et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

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

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using megatron-lm. In Bronis R. de Supinski, Mary W. Hall, and Todd Gamblin, editors, *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, page 58. ACM, 2021. doi: 10.1145/3458817.3476209. URL `https://doi.org/10.1145/3458817.3476209`.

NVIDIA. Nvshmem: Nvidia openshmem library for nvidia gpus. Software documentation, 2023. URL `https://developer.nvidia.com/nvshmem`.

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

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37:68658–68685, 2024.

Zewei Tao and Yunpeng Huang. Magiattention: A distributed attention towards linear scalability for ultra-long context, heterogeneous mask training. `https://github.com/SandAI-org/MagiAttention`, 2025.

Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, Matt Nicely, Duane Merrill, Dustyn Blasig, Aditya Atluri, Fengqi Qiao, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Manish Gupta. CUTLASS, January 2023. URL `https://github.com/NVIDIA/cutlass`.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

Guoxia Wang, Jinle Zeng, Xiyuan Xiao, Siming Wu, Jiabin Yang, Lujing Zheng, Zeyu Chen, Jiang Bian, Dianhai Yu, and Haifeng Wang. Flashmask: Efficient and rich mask extension of flashattention. *arXiv preprint arXiv:2410.01359*, 2024.

Haifeng Wang, Hua Wu, Tian Wu, Yu Sun, Jing Liu, Dianhai Yu, Yanjun Ma, et al. Ernie 5.0 technical report. *arXiv preprint arXiv:2602.04705*, 2026.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, 2025.

Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. Deepep: an efficient expert-parallel communication library. `https://github.com/deepseek-ai/DeepEP`, 2025.

## A Limitations and Broader Impact

**Limitations.** The method covers the column-wise sparse mask family of Wang et al. [2024]; masks whose per-column masked rows do not decompose into two intervals require representation extensions. The Hopper kernel exploits TMA/WGMMA and assumes Hopper-class architectures; porting to other architectures requires re-targeting the warp specialization. The hierarchical overlap of §3.4 assumes per-node 8-GPU NVLink islands with same-rank IB topology; clusters with different topologies need a different inter-node mapping. Hierarchical routing is currently applied only to the forward

all-gather; extending it to the backward reduce-scatter is left as future work. Empirical claims rely on production-scale long-context settings; smaller contexts may not amortize the scheduler overhead. Finally, our benchmarks cover a representative but not exhaustive set of QKV tensor configurations; a systematic sweep over batch sizes and query/key-value head ratios beyond those reported in §4.2 and §4.3 is left to future work.

**Broader impact.** By improving the throughput and energy efficiency of long-context training, FlashMask-3 lowers the cost barrier for academic and open-source long-context multimodal research and reduces the carbon footprint of large-scale training. The same efficiency gains can also lower the cost of training capability-amplifying models, which is an externality common to research on large-scale training infrastructure.

## B Extended Related Work

**IO-aware exact attention.** FlashAttention [Dao et al., 2022] introduced tiling and online softmax [Milakov and Gimelshein, 2018, Rabe and Staats, 2021] to compute exact attention in $O(N)$ memory without materializing the $N \times N$ score matrix. FlashAttention-2 [Dao, 2023] swapped the loop order and optimized warp-level work partitioning, reaching 50–73% of A100 peak throughput. FlashAttention-3 [Shah et al., 2024] further added Hopper-native TMA/WGMMA pipelining and FP8 support, achieving 740 TFLOPs/s on H100. Block-Sparse FlashAttention [Dao et al., 2022] augments the tiled kernel with an $O(N^2/B^2)$ binary block mask for tile skipping; FlashMask's column-wise sparse representation achieves $O(N)$ storage with token-level granularity.

**Compiler-driven attention.** FlexAttention [He et al., 2024] JIT-compiles arbitrary `mask_mod` functions into Triton [Tillet et al., 2019] kernels, offering maximal flexibility at the cost of per-mask compilation overhead and a code path separate from FA3's Hopper-native pipeline. Under distributed CP, each rank independently reconstructs the local block mask at $O(QK/B^2)$ cost. FlashMask-3 avoids both issues through a fixed kernel binary and zero-communication mask localization.

**Learnable block-sparse attention.** NSA [Yuan et al., 2025] combines compression, top-$k$ block selection, and sliding-window branches in an end-to-end differentiable architecture that is hardware-aligned and natively trainable from pretraining. MoBA [Lu et al., 2025] routes each query to its top-$k$ KV blocks via MoE-style gating, enabling seamless switching with full attention. Both learn *which* tokens to attend to and require architectural modification. FlashMask-3 solves a complementary problem: efficiently computing a *pre-specified* structural mask with no model changes required.

**Structured mask representations.** MagiAttention [Tao and Huang, 2025] represents masks as AttnSlice triples $(Q_{\text{range}}, K_{\text{range}}, \text{MaskType})$, cut along Q/K axes for distribution; this requires a dedicated FFA kernel with per-slice atomics. FlashMask [Wang et al., 2024] uses column-wise sparse boundary vectors at $O(N)$ storage, covering 12+ structured mask patterns. FlashMask-3 extends this to distributed settings through an elementwise clip–shift–max transform that preserves the single-GPU kernel interface at zero cross-rank communication.

**Context parallelism.** Ring Attention [Liu et al., 2023] circulates KV on a device ring with dual-stream overlap, achieving $\sim$50% GPU utilization under causal masks due to inherent load imbalance [Li et al., 2023]. Striped Attention [Brandon et al., 2023] restores causal balance via round-robin token assignment (1.45–1.65$\times$ over Ring) but supports only causal masks. DeepSpeed Ulysses [Jacobs et al., 2023] converts to head parallelism via all-to-all, natively supporting arbitrary masks but requiring $N_{\text{heads}} \bmod P = 0$. USP [Fang and Zhao, 2024] unifies Ulysses (intra-node) with Ring (inter-node). Megatron CP [Narayanan et al., 2021] employs dual chunking for causal balance; under irregular masks such as shared-question (RLHF) or multi-document masks, its linear-cost assumption fails and per-rank utilization degrades significantly.

**Load balancing under irregular masks.** DCP [Jiang et al., 2025] (SOSP 2025) models CP scheduling as hypergraph partitioning over block-level workloads, achieving $<$5% imbalance at the cost of a heavyweight solver. MagiAttention [Tao and Huang, 2025] uses chunk-level LPT-greedy assignment [Graham, 1969] based on AttnSlice areas. FlashMask-3 derives per-chunk valid-block counts directly from the column-wise boundary vectors and solves the scheduling subproblem with provable optimality guarantees (§3.3), avoiding external solver overhead.

**Compute–communication overlap.** Ring Attention [Liu et al., 2023] relies on dual CUDA streams, which suffer SM contention when large attention kernels starve communication of resources. FLUX [Chang et al., 2024] demonstrates that kernel-level tile fusion eliminates this contention for GEMMs, but online softmax prevents straightforward extension to attention. MagiAttention uses DeepEP-based [Zhao et al., 2025] GroupCast/GroupReduce with stream-level multi-stage overlap. FlashMask-3 uses NVSHMEM [NVIDIA, 2023] GPU-initiated one-sided communication with zero-copy symmetric buffers shared with the attention kernel, achieving high overlap without perturbing the WGMMA critical path. Mesh-Attention [Chen et al., 2025] reduces communication volume from $O(Nd)$ to $O(Nd/\sqrt{P})$ via a 2D assignment matrix, a topological optimization orthogonal and composable with FlashMask-3's latency-focused NVSHMEM overlap.

# C Proofs and Additional Technical Details

## C.1 Distributed mask transform: definitions, complete proof, and resource profile

We restate the building blocks of §3.2 as formal definitions, prove Theorem 1 in full, and report the operator-level resource profile of the transform.

**Definition 1** (Single-chunk local clip)**.** For a query chunk $[O, O+C)$ in global coordinates and a global boundary vector $\mathbf{x} \in \mathbb{Z}_{\geq 0}^{N_K}$, the single-chunk local clip is

$$\big[\mathcal{L}_{O,C}(\mathbf{x})\big]_k \;=\; \mathrm{clip}\big(\mathbf{x}_k - O,\; 0,\; C\big), \qquad k \in [0, N_K).$$

**Definition 2** (Multi-chunk shifted clip, merge, and rank-local mask)**.** Suppose rank $r$ owns $n_r$ disjoint query chunks $\{[O_i, O_i + C)\}_{i=0}^{n_r-1}$ sorted by global offset $O_0 < O_1 < \cdots < O_{n_r-1}$, mapped to the rank-local concatenated coordinate system $\bigsqcup_{i=0}^{n_r-1}[iC, (i{+}1)C)$. The multi-chunk localized boundary on rank $r$ is

$$\big[\mathcal{M}^{(r)}(\mathbf{x})\big]_k \;=\; \max_{0\leq i<n_r}\Big(\big[\mathcal{L}_{O_i,C}(\mathbf{x})\big]_k + \mathbf{1}\big[\mathcal{L}_{O_i,C}(\mathbf{x})_k > 0\big]\cdot iC\Big), \quad k \in [0, N_K), \tag{6}$$

which is the operator form of the main-paper transform (1). The rank-local column-wise sparse mask is then $\mathbf{FM}_l^{(r)} = \big(\mathcal{M}^{(r)}(\mathrm{LTS}),\, \mathcal{M}^{(r)}(\mathrm{LTE}),\, \mathcal{M}^{(r)}(\mathrm{UTS}),\, \mathcal{M}^{(r)}(\mathrm{UTE})\big)$, retaining the same shape $4 \times N_K$, dtype, and column-wise sparse semantics as $\mathbf{FM}_g$.

**Lemma 1** (Cross-chunk contribution bounds)**.** *Under the notation of Definition 2, fix $k \in [0, N_K)$, $\tilde{q} \in [0, n_r C)$, and let $i^\star = \lfloor \tilde{q}/C \rfloor$, $\delta = \tilde{q} - i^\star C$, $q^\star = O_{i^\star} + \delta$. For any boundary vector $\mathrm{X} \in \{\mathrm{LTS}, \mathrm{LTE}, \mathrm{UTS}, \mathrm{UTE}\}$, define*

$$f^{\mathrm{X}}(i) \;=\; \big[\mathcal{L}_{O_i,C}(\mathrm{X})\big]_k + \mathbf{1}\big[\mathcal{L}_{O_i,C}(\mathrm{X})_k > 0\big]\cdot iC.$$

*Then the following bounds hold for all chunks $i \neq i^\star$:*

*(i) For $i < i^\star$:* $f^{\mathrm{X}}(i) \leq (i{+}1)C \leq i^\star C \leq \tilde{q}$.

*(ii) For $i > i^\star$ with $\mathrm{X}_k \leq q^\star$:* $f^{\mathrm{X}}(i) = 0$.

*(iii) For $i > i^\star$ with $\mathrm{X}_k > q^\star$:* $f^{\mathrm{X}}(i) \geq (i^\star{+}1)C > \tilde{q}$.

*Proof.* (i) By the clip upper bound, $[\mathcal{L}_{O_i,C}(\mathrm{X})]_k \leq C$, so $f^{\mathrm{X}}(i) \leq C + iC = (i{+}1)C$. Since $i < i^\star$, we have $(i{+}1)C \leq i^\star C \leq \tilde{q}$.

(ii) For $i > i^\star$, disjointness gives $O_i \geq O_{i^\star} + C > q^\star \geq \mathrm{X}_k$. Hence $\mathrm{X}_k - O_i < 0$, the clip vanishes, and $f^{\mathrm{X}}(i) = 0$.

(iii) For $i > i^\star$ with $\mathrm{X}_k > q^\star = O_{i^\star} + \delta$, we have $\mathrm{X}_k \geq O_{i^\star} + \delta + 1$. If additionally $\mathrm{X}_k > O_i$ (so the clip is positive), then $f^{\mathrm{X}}(i) \geq 1 + iC \geq (i^\star{+}1)C + 1 > \tilde{q}$. If $\mathrm{X}_k \leq O_i$, the clip vanishes and $f^{\mathrm{X}}(i) = 0$; however, in this case chunk $i^\star$ itself satisfies $\mathrm{X}_k - O_{i^\star} \geq \delta + 1 > 0$, so $f^{\mathrm{X}}(i^\star) \geq (\delta + 1) + i^\star C = \tilde{q} + 1 > \tilde{q}$, and the max is already dominated by chunk $i^\star$. □

**Theorem 1** (Distributed mask correctness)**.** *For every locally-indexed query $\tilde{q} \in [0, n_r C)$ and global key $k \in [0, N_K)$,*

$$\tilde{q} \in \big[\mathrm{LTS}_k^{(r)}, \mathrm{LTE}_k^{(r)}\big) \cup \big[\mathrm{UTS}_k^{(r)}, \mathrm{UTE}_k^{(r)}\big) \iff q^\star \in [\mathrm{LTS}_k, \mathrm{LTE}_k) \cup [\mathrm{UTS}_k, \mathrm{UTE}_k),$$

*where $q^\star = O_{i^\star} + \delta$, $i^\star = \lfloor \tilde{q}/C \rfloor$, $\delta = \tilde{q} - i^\star C$.*

*Proof.* It suffices to prove the equivalence for the lower-triangular pair; the upper-triangular pair follows by substituting $(\mathrm{UTS}, \mathrm{UTE})$ for $(\mathrm{LTS}, \mathrm{LTE})$ throughout. Recall that $\mathrm{LTS}_k^{(r)} = \max_i f^{\mathrm{LTS}}(i)$ and $\mathrm{LTE}_k^{(r)} = \max_i f^{\mathrm{LTE}}(i)$.

**($\Rightarrow$) $q^\star \in [\mathrm{LTS}_k, \mathrm{LTE}_k)$ implies $\tilde{q} \in [\mathrm{LTS}_k^{(r)}, \mathrm{LTE}_k^{(r)})$.**

Assume $\mathrm{LTS}_k \le q^\star < \mathrm{LTE}_k$.

*Step 1 (Upper bound is strict: $\mathrm{LTE}_k^{(r)} > \tilde{q}$).* Since $\mathrm{LTE}_k > q^\star = O_{i^\star} + \delta$, we have $\mathrm{LTE}_k - O_{i^\star} \ge \delta + 1 > 0$, so the clip satisfies $[\mathcal{L}_{O_{i^\star}, C}(\mathrm{LTE})]_k \ge \delta + 1$ and the indicator is active:

$$f^{\mathrm{LTE}}(i^\star) \;=\; [\mathcal{L}_{O_{i^\star}, C}(\mathrm{LTE})]_k + i^\star C \;\ge\; (\delta + 1) + i^\star C \;=\; \tilde{q} + 1.$$

Hence $\mathrm{LTE}_k^{(r)} \ge f^{\mathrm{LTE}}(i^\star) > \tilde{q}$.

*Step 2 (Lower bound is met: $\mathrm{LTS}_k^{(r)} \le \tilde{q}$).* Since $\mathrm{LTS}_k \le q^\star = O_{i^\star} + \delta$, we have $\mathrm{LTS}_k - O_{i^\star} \le \delta$, giving $[\mathcal{L}_{O_{i^\star}, C}(\mathrm{LTS})]_k \le \delta$. If the clip is positive, $f^{\mathrm{LTS}}(i^\star) \le \delta + i^\star C = \tilde{q}$; if zero, $f^{\mathrm{LTS}}(i^\star) = 0 \le \tilde{q}$. For $i \ne i^\star$: Lemma 1(i) bounds chunks $i < i^\star$ by $\tilde{q}$, and part (ii) (with $\mathrm{LTS}_k \le q^\star$) zeros chunks $i > i^\star$. Thus $\mathrm{LTS}_k^{(r)} = \max_i f^{\mathrm{LTS}}(i) \le \tilde{q}$.

Combining: $\mathrm{LTS}_k^{(r)} \le \tilde{q} < \mathrm{LTE}_k^{(r)}$, i.e., $\tilde{q} \in [\mathrm{LTS}_k^{(r)}, \mathrm{LTE}_k^{(r)})$.

**($\Leftarrow$) $q^\star \notin [\mathrm{LTS}_k, \mathrm{LTE}_k)$ implies $\tilde{q} \notin [\mathrm{LTS}_k^{(r)}, \mathrm{LTE}_k^{(r)})$.**

*Case 1 ($q^\star < \mathrm{LTS}_k$: query below the interval start).* Then $\mathrm{LTS}_k - O_{i^\star} > \delta \ge 0$, so $[\mathcal{L}_{O_{i^\star}, C}(\mathrm{LTS})]_k \ge \delta + 1 > 0$ and

$$f^{\mathrm{LTS}}(i^\star) \;\ge\; (\delta + 1) + i^\star C \;=\; \tilde{q} + 1 \;>\; \tilde{q}.$$

Hence $\mathrm{LTS}_k^{(r)} \ge f^{\mathrm{LTS}}(i^\star) > \tilde{q}$, giving $\tilde{q} < \mathrm{LTS}_k^{(r)}$ and thus $\tilde{q} \notin [\mathrm{LTS}_k^{(r)}, \mathrm{LTE}_k^{(r)})$.

*Case 2 ($q^\star \ge \mathrm{LTE}_k$: query at or above the interval end).* Then $\mathrm{LTE}_k - O_{i^\star} \le \delta$, so $[\mathcal{L}_{O_{i^\star}, C}(\mathrm{LTE})]_k \le \delta$. If the clip is positive, $f^{\mathrm{LTE}}(i^\star) \le \delta + i^\star C = \tilde{q}$; if zero, $f^{\mathrm{LTE}}(i^\star) = 0 \le \tilde{q}$. For $i < i^\star$, Lemma 1(i) gives $f^{\mathrm{LTE}}(i) \le \tilde{q}$. For $i > i^\star$: since $\mathrm{LTE}_k \le q^\star + 1 \le O_{i^\star} + C \le O_i$ (by integrality and disjointness), the clip vanishes and $f^{\mathrm{LTE}}(i) = 0$. Thus $\mathrm{LTE}_k^{(r)} \le \tilde{q}$, so $\tilde{q} \ge \mathrm{LTE}_k^{(r)}$ and $\tilde{q} \notin [\mathrm{LTS}_k^{(r)}, \mathrm{LTE}_k^{(r)})$.

**Indicator function.** The indicator $\mathbf{1}[\cdot > 0]$ in (6) is essential: without it, a chunk with vanishing clip would contribute $f^{\mathrm{X}}(i) = iC$, potentially dominating the maximum for large $i$ and corrupting the localized boundary.

**Upper-triangular pair and conclusion.** Replacing $(\mathrm{LTS}, \mathrm{LTE})$ with $(\mathrm{UTS}, \mathrm{UTE})$ and repeating the above two-case analysis yields $\tilde{q} \in [\mathrm{UTS}_k^{(r)}, \mathrm{UTE}_k^{(r)}) \Leftrightarrow q^\star \in [\mathrm{UTS}_k, \mathrm{UTE}_k)$. Taking the union over both pairs completes the proof. □

Table 2: Resource profile of the FlashMask-3 distributed mask preprocessing on rank $r$ owning $n_r$ chunks. $N_K$ denotes the global key length. All operations are elementwise and parallelized over the key dimension.

| Operation | Time | Memory | Cross-rank comm. | Kernel interface |
|---|---|---|---|---|
| Single-chunk clip (Def. 1) | $O(N_K)$ | $O(N_K)$ | 0 | shared |
| Multi-chunk shift+merge (Def. 2) | $O(n_r N_K)$ | $O(N_K)$ | 0 | shared |
| Total mask preprocessing | $O(n_r N_K)$ | $O(N_K)$ | **0** | **shared** |

### C.2 Workload computation for each Q-chunk

The load balancer of §3.3 requires a per-chunk compute load $w_i$ that reflects the mask-induced sparsity. For each pair ($Q$-block, $K$-block), the blockwise max/min of the rank-local boundary vectors (§2.2) classifies the tile as fully-masked, partially-masked, or unmasked, with associated costs

$$\text{cost}(q,k) \ \in \ \{0,\ \alpha_p,\ \alpha_u\},$$

where $\alpha_p \geq \alpha_u > 0$. The cost $\alpha_p$ reflects the additional row-interval mask applied to partially-masked tiles (§3.1), while fully-masked tiles incur zero cost because they are skipped entirely by the KV Index Generator. Aggregating these tile costs across all key blocks and all batch/head dimensions yields, for each chunk $i$, its total compute load:

$$w_i \ = \sum_{\substack{q\in\text{chunk } i,\\ k\in[0,N_K/B)}} \text{cost}(q,k).$$

This computation is performed before HCC-IPO scheduling. Its time complexity is $O(N_K^2/B)$ per batch/head (one pass over the $N_Q/B \times N_K/B$ tile grid), and the resulting $\{w_i\}$ are broadcast-free since every rank computes the same values from the same global mask.

### C.3 HCC-IPO: Complete Proofs and Pseudocode

This appendix supplies the deferred material from §3.3: the IPO scheduler in pseudocode, its 2-way and 3-way sub-solvers, and formal proofs of correctness.

**Notation.** We follow §3.3: $N$ chunks with non-negative integer weights $w_i \in \mathbb{Z}_{\geq 0}$, $P$ ranks, per-rank cardinality $\kappa = N/P$, assignment $\pi : [N] \to [P]$ with rank set $S_j(\pi) = \pi^{-1}(j)$ and rank load $L_j(\pi) = \sum_{i\in S_j(\pi)} w_i$. For an arbitrary chunk set $S$, define $L(S) \triangleq \sum_{i\in S} w_i$. The *makespan* is $\Phi(\pi) \triangleq \max_{j\in[P]} L_j(\pi)$; when the assignment is clear from context we write simply $\Phi$. The lower bound is $LB \triangleq \max\big(\lceil \frac{1}{P}\sum_i w_i \rceil,\ \max_i w_i\big)$.

#### C.3.1 Algorithm: IPO Scheduler for Stage 1

Algorithm 1 solves (2) by iteratively repartitioning the heaviest rank together with one or two partner ranks via exact combinatorial sub-solvers. Phase 2 performs pairwise repartitioning; Phase 3 performs triplewise repartitioning when Phase 2 alone cannot improve.

#### C.3.2 Sub-solvers

**EXACT2WAY.** Given a pool $U$ of $2\kappa$ chunks, returns a bipartition $(A, B)$ with $|A| = |B| = \kappa$ that minimizes $\max(L(A), L(B))$ over all $\binom{2\kappa}{\kappa}$ candidates. For $\kappa \leq 10$, we use Meet-in-the-Middle: split $U$ into two halves, enumerate all $2^\kappa$ subsets of each, bucket by cardinality, sort by partial sum, and binary-search complementary half-sums in $O(2^\kappa \kappa)$ time. For $\kappa > 10$, we binary-search the target value $T$ and solve a cardinality-constrained feasibility problem via Branch-and-Bound (BnB) with symmetry (skip assigning to the second bucket when both buckets have identical load and count), suffix-sum (backtrack when remaining items cannot fill both buckets within $T$), and equal-weight pruning (avoid redundant branches when consecutive items share the same weight).

**FEAS3WAY.** Given a pool $U$ of $3\kappa$ chunks and a target $T$, decides whether $U$ admits a tripartition $(A, B, C)$ with $|A| = |B| = |C| = \kappa$ and $\max(L(A), L(B), L(C)) \leq T$. Uses depth-first BnB over element-to-bucket assignments in descending weight order, with four pruning rules: (i) cardinality saturation (skip buckets already holding $\kappa$ items), (ii) load overflow (skip if adding the current item exceeds $T$), (iii) suffix-sum infeasibility (the same as 2-way solver), and (iv) bucket-state symmetry (skip assignments to buckets with identical load and count, as they yield equivalent subtrees). Active only when $\kappa \leq 8$.

#### C.3.3 Theoretical Guarantees

**Theorem 2** (IPO correctness)**.** *Let $\pi^*$ denote the output of Algorithm 1.*

**Algorithm 1** IPO scheduler for cardinality-constrained min-max partitioning

**Require:** Chunk weights $\{w_i\}_{i=1}^{N}$, number of ranks $P$, cardinality $\kappa = N/P$
**Ensure:** Assignment $\{S_j\}_{j=1}^{P}$ with $|S_j| = \kappa$
$\{S_j\} \leftarrow \text{LPT}(\{w_i\}, P, \kappa)$ ▷ Assign heaviest chunk to lightest rank
$LB \leftarrow \max\big(\lceil \sum_i w_i/P \rceil,\ \max_i w_i\big)$
**repeat**
  $H \leftarrow \arg\max_j L(S_j)$ ▷ Heaviest rank (smallest index on ties)
  **if** $L(S_H) \le LB$ **then return** $\{S_j\}$ ▷ Certified optimal
  **end if**
  ▷ **Phase 2 (pairwise):** evaluate all partners, commit best
  **for all** $R \in [P] \setminus \{H\}$ **do**
    $(S'_H, S'_R) \leftarrow \textsc{Exact2Way}(S_H \cup S_R,\ \kappa)$; record $R$ if global makespan improves
  **end for**
  **if** best $R^*$ yields global makespan $< \Phi$ **then** update $S_H, S_{R^*}$; **continue**
  **end if**
  ▷ **Phase 3 (triplewise, $\kappa \le 8$):** commit first feasible triple
  **for all** $(R_1, R_2) \in \binom{[P]\setminus\{H\}}{2}$, ascending by load, pruned by bounds **do**
    $(S'_H, S'_{R_1}, S'_{R_2}) \leftarrow \textsc{Feas3Way}(S_H \cup S_{R_1} \cup S_{R_2},\ \kappa,\ \Phi - 1)$
    **if** feasible and global makespan $< \Phi$ **then** update; **break**
    **end if**
  **end for**
**until** neither phase improves $\Phi$
**return** $\{S_j\}_{j=1}^{P}$

*(a)* ***Monotone convergence.*** *The makespan $\Phi$ strictly decreases at each iteration that commits an update, and the algorithm terminates in finitely many iterations.*
*(b)* ***Optimality.*** *If $\Phi(\pi^*) = LB$, then $\pi^*$ is a global optimum of* (2).
*(c)* ***Determinism.*** *The output $\pi^*$ is a deterministic function of $(\{w_i\}, P, \kappa)$, ensuring all CP ranks derive the same assignment without communication.*

*Proof of (a).* The algorithm commits an update only when the resulting global makespan is strictly less than the current $\Phi$: Phase 2 commits only if the best partner yields a new global maximum $< \Phi$, and Phase 3 commits only if a tripartition with all three loads $\le \Phi - 1$ exists and the resulting global maximum $< \Phi$. Hence each committed iteration satisfies $\Phi^{(t+1)} < \Phi^{(t)}$. When no phase can improve $\Phi$, the loop terminates. Since $\Phi \in \mathbb{Z}_{\ge 0}$ (all $w_i$ are non-negative integers) and $\Phi \ge LB \ge 0$ for any feasible assignment, the algorithm terminates in at most $\Phi^{(0)} - LB$ committed iterations, where $\Phi^{(0)}$ is the LPT makespan. □

*Proof of (b).* For any feasible $\pi$ with $|S_j(\pi)| = \kappa$: (i) $\Phi(\pi) \ge \frac{1}{P}\sum_i w_i$ since the maximum is at least the average, and integrality gives $\Phi(\pi) \ge \lceil \frac{1}{P}\sum_i w_i \rceil$; (ii) the heaviest chunk resides in some rank, so $\Phi(\pi) \ge \max_i w_i$. Hence $\Phi(\pi) \ge LB$ for all feasible $\pi$. When Algorithm 1 exits via the certificate check with $\Phi(\pi^*) \le LB$, we obtain $\Phi(\pi^*) = LB \le \Phi(\pi)$ for every feasible $\pi$, certifying global optimality. □

*Proof of (c).* All algorithmic choices are deterministic functions of the current state $(\{S_j\}, \{w_i\})$: (i) LPT sorts by descending $w_i$ with ascending index as tie-breaker, and fills by ascending load with ascending rank index as tie-breaker; (ii) $H$ is the smallest-index rank among those with maximum load; (iii) Phase 2 evaluates all partners and commits the one yielding the lowest global makespan (ties broken by smallest partner index); (iv) Phase 3 iterates partner pairs in a fixed order (ascending load, ties by index) and commits the first feasible triple; (v) sub-solvers enumerate in descending-weight, ascending-index order, yielding a unique output for each input. Since the initial state is uniquely determined by the input via (i), and each iteration maps to a unique successor via (ii)–(v), the output $\pi^*$ is uniquely determined. □

The condition $\Phi = LB$ acts as a verifiable optimality certificate, checkable in $O(N)$ time.

**Discussion.** The cardinality-constrained min-max $P$-way partitioning problem (2) is NP-hard in general, and an exact solver would require Branch-and-Bound over the full $P^N$ assignment space. IPO deliberately restricts each iteration to 2- or 3-rank sub-problems whose exact solutions are tractable (via MitM or BnB with tight pruning), trading global exhaustiveness for polynomial per-iteration cost. This local-search structure cannot theoretically guarantee convergence to the global optimum in all cases. In practice, however, the LB certificate (Theorem 2(b)) provides a sufficient condition that is verifiable in $O(N)$ time: whenever IPO attains $\Phi = LB$, global optimality is certified without requiring full enumeration.

Table 3 ablates the contribution of each IPO phase. We report the *sub-optimality rate* (fraction of cases where the method exceeds the proven optimum) and the *maximum gap* (worst-case relative deviation from optimal). The reference optimum is obtained by an exact $P$-way BnB solver with a generous node budget; for configurations where this solver times out, we use the LPT+P2+P3 result as reference (denoted †).

Table 3: IPO phase ablation across representative CP configurations. *Subopt*: fraction of random instances not matching the optimum. *Max gap*: worst-case relative excess over optimal makespan. *P3 wins*: cases where Phase 3 improves upon Phase 2. All configurations use 200 random instances (100 for $N = 128$).

| | | LPT | | LPT+P2 | | LPT+P2+P3 | |
|---|---|---|---|---|---|---|---|
| Configuration | $(N, P, \kappa, w_{\max})$ | Subopt | Gap | Subopt | Gap | Subopt | P3 wins |
| Uniform | $(64, 16, 4, 10^3)$ | 99.0% | 6.54% | 86.0% | 6.29% | 0% | 172 |
| Uniform (wide) | $(64, 16, 4, 10^4)$ | 100% | 8.67% | 93.5% | 3.42% | 0% | 187 |
| Causal | $(64, 8, 8, 200)$ | 93.5% | 2.51% | 53.5% | 1.39% | 0% | 107 |
| Uniform† | $(128, 16, 8, 200)$ | 85.0% | 1.08% | 48.0% | 0.79% | 0% | 48 |

LPT alone is suboptimal in 85–100% of cases. Phase 2 (exact pairwise repartitioning) substantially reduces the gap but cannot escape local minima requiring simultaneous three-rank adjustments. Phase 3 resolves all remaining cases, achieving a 100% LB-certified optimality rate across all of our tested configurations.

### C.3.4 Non-interference of Stages 2 and 3

*Proof that Stages 2 and 3 do not interfere.* We show Stage 3 preserves Stage 2's communication reduction while improving compute balance.

*Step 1.* Stage 2 performs one-for-one chunk swaps between nodes $m_{\max}$ and $m_{\min}$, preserving $|S_j| = \kappa$ for every rank.

*Step 2.* Stage 3 runs IPO restricted to ranks $\mathcal{R}_m$ on chunk pool $T_m = \bigcup_{j \in \mathcal{R}_m} S_j$, independently per node. No chunk leaves its node, so $T_m$ is preserved exactly.

*Step 3.* By (3), $\mathrm{Comm}(m, \pi) = \left|\bigcup_{i \in T_m} K_i \setminus \mathrm{Local}(m)\right|$ depends only on $T_m$ (which determines both the KV-dependency union and the locally available K-blocks). Since $T_m$ is unchanged (Step 2), $\mathrm{Comm}(m, \pi)$ is invariant under Stage 3.

*Step 4.* By Theorem 2(a) applied per node, the per-node makespan $\max_{j \in \mathcal{R}_m} L_j$ is non-increasing under Stage 3. Taking the global maximum, $\Phi$ is non-increasing across Stage 3. □

## C.4 Forward all-gather protocol and pseudocode

This appendix formalizes the forward AG overlap protocol of §3.4.

### C.4.1 Write-pointer synchronization

The AG communication kernel runs $N_{\text{cta}}$ CTAs on a high-priority stream, dynamically claiming work units (each covering $R$ KV rows) via an atomic counter. A monotonically increasing *write pointer* (`wptr`) signals progress to the attention kernel: all rows at offsets $<$ `wptr` are guaranteed globally visible.

**Algorithm 2** Hierarchical AG communication kernel (per CTA)

**Require:** SR buffer $\mathcal{B}$ ($P$ slots); `copy_chunk_mask`; per-rank semaphores

**while** work remains **do**
  $w \leftarrow$ ATOMICADD(`counter`, 1) ▷ Dynamic scheduling
  Map $w \rightarrow$ (`target_rank`, `src_pe`, `is_phase1`)
  **if** $\neg$ `is_phase1` $\wedge$ `copy_chunk_mask`$[w]$ **then** ▷ Sparse skip (Phase 2 only)
    Advance `wptr`; **continue**
  **end if**
  WAITSEMAPHORE(`semaphores[target_rank]`) ▷ Source data ready
  **if** `is_phase1` **then** ▷ Cross-node (RoCE)
    NVSHMEMGET($\mathcal{B}$[`dst`] $\leftarrow$ `target_rank`'s local KV)
    CONGRUENCENOTIFY(same-node peers) ▷ Enable Phase 2
  **else** ▷ Intra-node (NVLink)
    NVSHMEMGET($\mathcal{B}$[`dst`] $\leftarrow$ `src_pe`'s $\mathcal{B}$[`slot`])
  **end if**
  Advance `wptr` via reduce-min (§C.4.1)
**end while**

**Producer (AG kernel).** Each CTA publishes its latest completed work ID to a per-CTA slot. Warp 0 reads all $N_{\text{cta}}$ slots via `__reduce_min_sync`, yielding a conservative contiguous frontier $w_{\min}$. A `__threadfence` then ensures KV store visibility before `atomicMax(wptr,` $w_{\min} \times R$`)` advances the pointer.

**Consumer (attention kernel).** The KV Loader iterates blocks right-to-left. For block $n$, it computes `target`$(n) = b\,(S_{\text{total}} - S_{\text{local}}) + (S_{\text{total}} - nB_N - S_{\text{local}})$. Blocks within the local shard ($S_{\text{total}} - nB_N - S_{\text{local}} < 0$) skip the wait; remote blocks spin on `wptr` via `ld.volatile.global` with a cached last-seen value.

### C.4.2 Deadlock prevention

Two mechanisms prevent the attention grid from starving the AG kernel of SM resources. Since the attention kernel depends on `wptr`, the AG kernels must be scheduled ahead of the computation kernels to guarantee forward progress: (1) each AG CTA sets a bit in a GPU-memory coordinator word upon physical scheduling; a sentinel kernel on the compute stream spin-waits until all bits are set before the attention grid launches; (2) the attention tile scheduler is capped at `num_sm`$-$`sm_margin` tiles, reserving capacity for the AG kernel. Yet in practice, we find that the sentinel stream coordinator kernel alone is sufficient to ensure correct SM scheduling, while enabling `sm_margin` may negatively impact computation performance. `sm_margin` is therefore set to 0 throughout the experiments.

### C.4.3 Hierarchical two-phase all-gather

Algorithm 2 gives the per-CTA logic for routing communication through congruence groups $\mathcal{C}(r)$ (Eq. 4).

**Phase 1 (RoCE).** Each device fetches $N-1$ chunks from cross-node congruence-group partners. Upon completing all work units for a remote rank, it broadcasts a semaphore signal to $G-1$ same-node peers, enabling their Phase 2 fetches.

**Phase 2 (NVLink).** Each device retrieves $(G-1) \cdot N$ chunks from same-node neighbors whose Phase 1 data is already resident, gated by the congruence-notify semaphore.

**Sparse skip constraint.** Phase 1 fetches are never skipped regardless of the mask, because same-node peers depend on the fetched data for Phase 2.

**Fetch budget.**

| Scheme | RoCE fetches/device | NVLink fetches/device |
|---|---|---|
| Flat AG | $P-1$ | 0 |
| Hierarchical AG | $N-1$ | $(G-1) \cdot N$ |

For $P$=32, $G$=8, $N$=4: cross-node traffic drops from 24 to 3 fetches per device ($\approx$90% reduction).

#### C.4.4 Overall timeline

1. Copy local $K, V$ into SR buffer; barrier PEs with per-rank semaphores.
2. Compute `copy_chunk_mask` (sparsity check).
3. Launch AG kernel (comm stream) $\rightarrow$ sentinel wait (compute stream) $\rightarrow$ attention kernel.

The attention kernel performs a single uninterrupted pass over the full post-AG $K, V$, with no cross-step softmax rescaling and no ring-step bookkeeping.

### C.5 Backward multi-stage reduce-scatter overlap

This appendix formalizes the backward RS overlap protocol of §3.4.

#### C.5.1 Per-chunk synchronization

The forward AG uses a contiguous write-pointer frontier `wptr` because the attention kernel scans KV blocks sequentially. The backward kernel is *column-parallel*: each CTA owns one KV chunk and iterates over all Q blocks independently, requiring no cross-chunk ordering. The communication kernel therefore replaces the write pointer with per-chunk `work_done` flags: after fetching chunk $c$, the CTA issues `__threadfence` (ensuring data visibility) then sets `work_done[`$c$`]` $= 1$. The backward compute CTA polls this flag directly before loading its chunk.

#### C.5.2 Bitmask-AMO reduce-scatter protocol

RS is realized as a sparse all-to-all via NVSHMEM one-sided `put`. Each device simultaneously acts as a *producer* (sending $\mathrm{d}K/\mathrm{d}V$ to chunk owners) and a *consumer* (receiving and reducing contributions from other ranks).

**Protocol.**

1. **Consumer**: zeroes its receive buffer, then broadcasts a 64-bit bitmask (one bit per expected producer) to all producers via NVSHMEM AMO, signalling readiness.
2. **Producer**: waits for the readiness signal, issues `nvshmem_putmem` for each tile, then after all tiles for a target rank complete: `nvshmem_fence` (orders data before signal) followed by `nvshmem_long_atomic_add(`$-$`(1` $\ll$ `my_pe))` to clear its bit on the consumer's bitmask.
3. **Consumer**: spins until bitmask $= 0$ (all producers done), then launches a local fp32 reduction kernel to accumulate into $\mathrm{d}K_{\text{acc}}, \mathrm{d}V_{\text{acc}}$.

A masked chunk that skips the `put` still clears its bit, preventing the consumer from blocking indefinitely.

#### C.5.3 Dual load balancing

- **SM-level**: CTAs dynamically claim work via an atomic counter, preventing stalls from uneven mask sparsity.
- **Network-level**: work IDs are interleaved across target ranks (target $=$ `wid` $\bmod C +$ base_rank), distributing traffic uniformly at any instant.

#### C.5.4 Multi-stage pipeline

The KV sequence is partitioned into $T$ segments of $C$ chunks each, where $C = 2^{\lfloor \frac{1}{2} \log_2 P \rfloor}$ (e.g., $P$=16 $\Rightarrow$ $C$=4, $T$=4). The RS of segment $t$ overlaps with the AG and backward compute of segment $t$+1. Because the forward-saved LSE eliminates cross-segment softmax rescaling, this partition incurs no overhead beyond fp32 $\mathrm{d}Q$ accumulation. Algorithm 3 gives the full multi-stage backward loop.

**Algorithm 3** Multi-stage backward overlap

**Require:** $Q, K, V, \mathrm{d}O$; SR buffer $\mathcal{B}$; dKV send/recv buffers $\mathcal{D}_s, \mathcal{D}_r$; segments $T$, chunks/seg $C$
**Ensure:** $\mathrm{d}Q, \mathrm{d}K, \mathrm{d}V$

Compute `copy_chunk_mask`; copy local KV $\rightarrow \mathcal{B}$; barrier all PEs
**for** $t = 0, 1, \ldots, T-1$ **do**
  ▷ **Stage 1**: segmented AG (comm stream)
  **for all** CTA (dynamic scheduling) **do**
    Fetch segment-$t$ KV chunk $c$ into $\mathcal{B}$ (skip if masked)
    `__threadfence()`; `work_done[`$c$`]` $\leftarrow 1$ ▷ Per-chunk ready flag
  **end for**
  ▷ **Stage 2**: backward attention (comp stream)
  Each compute CTA polls `work_done[`$c$`]` before loading its KV chunk
  Compute $\mathrm{d}Q_t$ (accumulated in fp32), write $\mathrm{d}K_t, \mathrm{d}V_t \rightarrow \mathcal{D}_s$
  ▷ **Stage 3**: reduce-scatter (producer/consumer streams, overlaps Stage 1–2 of $t{+}1$)
  *Consumer*: zero $\mathcal{D}_r$; broadcast readiness bitmask to producers (AMO)
  **for all** CTA (dynamic scheduling) **do** ▷ Producer
    $w \leftarrow$ AtomicAdd(`counter`, 1); target $\leftarrow (w \bmod C) +$ base_rank
    **if** chunk masked **then** clear own bit on target's bitmask; **continue**
    **end if**
    NvshmemPut($\mathcal{D}_r$[target] $\leftarrow \mathcal{D}_s$[tile])
    **if** last tile for target **then** `nvshmem_fence()`; clear bit via AMO
    **end if**
  **end for**
  *Consumer*: wait bitmask $= 0$; $\mathrm{d}K_{\text{acc}}, \mathrm{d}V_{\text{acc}} \mathrel{+}= \textsc{Reduce}(\mathcal{D}_r)$
**end for**

# D More Results

## D.1 Experimental setup and supplementary results

This appendix details the experimental configuration for the single-device kernel benchmark (§4.1) and the CP-scale distributed benchmark (§4.2), followed by supplementary results.

### D.1.1 Single-device kernel benchmark

**Tensor configuration.** We fix head dimension $D{=}128$ and set the number of attention heads $H{=}32$ (MHA, no GQA). Batch size is chosen to saturate GPU memory:

$$B = \max\big(\lfloor 128{\times}1024 \,/\, S \rfloor,\ 1\big), \tag{7}$$

yielding $B{=}16$ at $S{=}8$K and $B{=}1$ at $S{=}128$K. Query, key, and value tensors share the same shape $[B, S, H, D]$. The mask boundary tensor has shape $[B, H_{\text{mask}}, S, C_{\text{mask}}]$, where $H_{\text{mask}} \in \{1, H\}$ (broadcast across heads when $H_{\text{mask}}{=}1$) and $C_{\text{mask}} \in \{1, 2, 4\}$ depending on mask complexity.

**TFLOPs/s computation.** We account for block-level sparsity following FlashMask's tile-skipping semantics. Let $\rho$ denote the effective density (fraction of non-skipped tiles). The FLOPs formulas are:

$$\mathrm{FLOPs}_{\text{fwd}} = \rho \cdot 4\eta_{\text{fwd}} BHS^2D, \tag{8}$$

$$\mathrm{FLOPs}_{\text{bwd}} = \rho \cdot 4\eta_{\text{bwd}} BHS^2D, \tag{9}$$

$$\mathrm{FLOPs}_{\text{total}} = \rho \cdot 4\eta_{\text{total}} BHS^2D, \tag{10}$$

and TFLOPs/s $= \mathrm{FLOPs} \,/\, (t_{\text{ms}} \times 10^9)$. Empirically, $\eta_{\text{fwd}} = 1$, $\eta_{\text{bwd}} = 2.5$ and $\eta_{\text{total}} = \eta_{\text{fwd}} + \eta_{\text{bwd}}$.

**Timing.** Each configuration is warmed up for 10 iterations. Measurements are collected over 50 timed iterations using CUDA events (`paddle.device.Event`), and the mean elapsed time is reported.

**Mask types.** We evaluate 12 representative mask patterns covering diverse real-world workloads: Full, Causal, Sliding Window, Causal Document, Document, Share-Question, Global Sliding Window, Causal Blockwise, Prefix-LM Document, Prefix-LM Causal, QK-sparse, and Random Eviction.

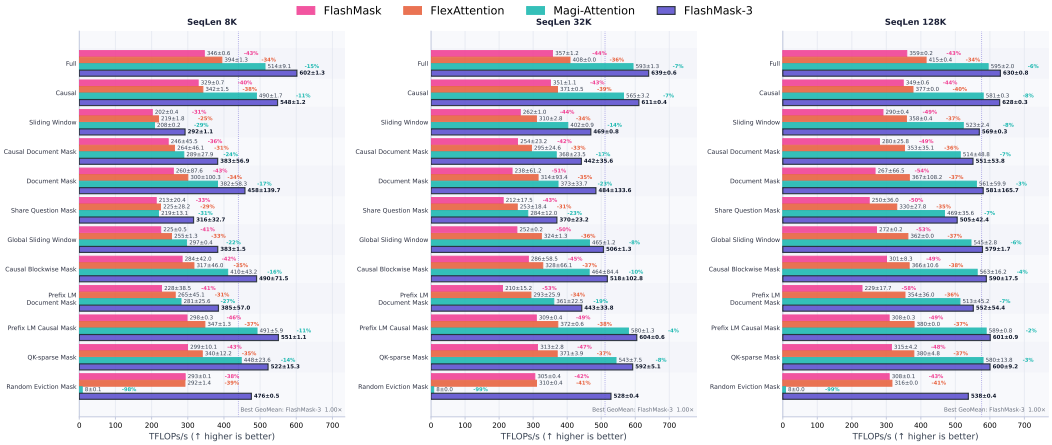



Figure 9: Single-device throughput across mask types and sequence lengths. FlashMask-3 consistently outperforms all baselines across all 12 mask patterns.

### D.1.2 CP-scale distributed benchmark

**Tensor configuration.** We fix $B{=}32/P$, $D{=}128$, $H_Q{=}64$ (Q heads), $H_{KV}{=}1$ (KV heads, i.e., GQA with ratio 64:1). Each rank holds a local KV sequence of $S_{\text{local}}{=}8192$ tokens; the total sequence length is $S = S_{\text{local}} \times P$ where $P$ is the CP size. QKV tensors per rank: $Q \in \mathbb{R}^{B\times S_{\text{local}}\times H_Q\times D}$, $K, V \in \mathbb{R}^{B\times S_{\text{local}}\times H_{KV}\times D}$. The mask tensor has shape $[B, H_{KV}, S, C_{\text{mask}}]$ (global mask on the full sequence).

**Timing protocol.** We use CPU wall-clock timing with explicit synchronization barriers. Before each timed phase (forward or backward), all ranks execute `barrier()` followed by `device.synchronize()`, then record `time.perf_counter()`. After each phase, the same barrier–synchronize sequence is repeated before reading the clock. This protocol isolates forward and backward timings, preventing load imbalance in one phase from contaminating the other. The rationale is twofold: (1) in real training pipelines, forward and backward are naturally separated by pipeline scheduling; (2) distributed collectives provide implicit inter-rank synchronization, so the barrier–synchronize pair ensures a clean measurement boundary.

**TFLOPs/s computation.** Same formulas as Eqs. (8)–(10), divided by $P$ (each rank computes $1/P$ of the total work under ideal balance). The GQA head count $H_Q{=}64$ is used in the FLOPs numerator.

### D.1.3 Single-device results

Figure 9 presents per-mask-type throughput (TFLOPs/s, forward+backward combined) across all tested sequence lengths and mask patterns.

Table 4 summarizes the relative improvements of FlashMask-3 over each baseline, aggregated across all mask types. We additionally present representative long- and short-sequence settings (128K and 8K, respectively) for illustration.

**Analysis.** FlashMask-3 achieves **+55.3%** mean improvement over FlexAttention and **+15.0%** over Magi-Attention across all configurations. The advantage is most pronounced at long sequences (128K), where FlashMask-3 reaches +58.9% over FlexAttention and up to +140.5% over FlashMask. At shorter sequences (8K), the gains remain substantial (+50.6% over FlexAttention on average), demonstrating robustness across operating regimes. The improvement over FlashMask (mean +82.6%, up to $2.4\times$) reflects the combined benefit of the Hopper-native SM90 kernel architecture and the improved tile scheduler.

### D.1.4 Full comparison results under CP scaling

Figure 10 compares FlashMask-3 (Balance+Overlap) against MagiAttention, Megatron CP (ring), and Megatron CP (all-gather) across all tested mask types and CP sizes.

Table 4: FlashMask-3 throughput improvement over baselines (forward+backward TFLOPs/s). Reported as $(T_{\text{ours}}/T_{\text{baseline}} - 1) \times 100\%$.

| Baseline | Scope | Min | Max | Mean |
|---|---|---|---|---|
| FlexAttention | All | +33.6% | +70.5% | +55.3% |
| | 8K | +33.6% | +62.9% | +50.6% |
| | 128K | +51.9% | +70.0% | +58.9% |
| Magi-Attention | All | +2.0% | +44.1% | +15.0% |
| | 8K | +12.0% | +44.1% | +25.0% |
| | 128K | +2.0% | +8.9% | +5.9% |
| FlashMask | All | +44.9% | +140.5% | +82.6% |
| | 8K | +44.9% | +84.8% | +66.2% |
| | 128K | +74.4% | +140.5% | +97.3% |

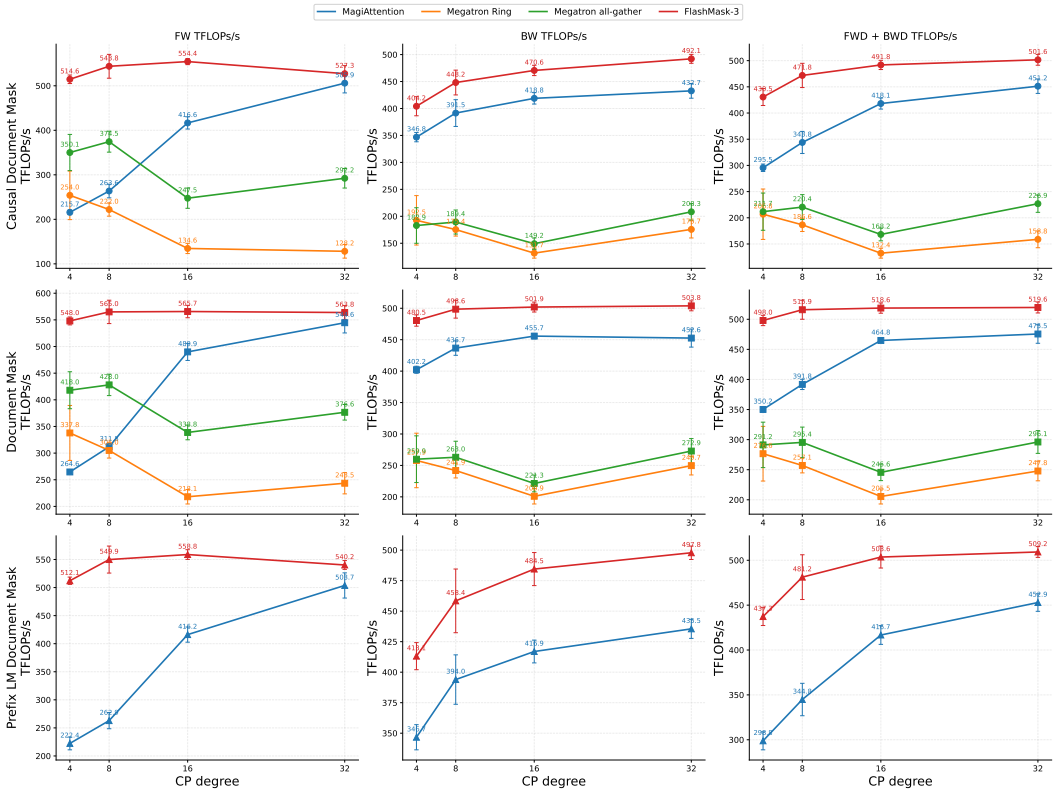



Figure 10: Q Head = 64, KV Head = 1. CP-scale throughput comparison (TFLOPs/s) across mask types and CP degrees. Columns: forward, backward, and total (forward+backward). Rows: mask types.

Table 5 reports FlashMask-3's speedup over each baseline, aggregated across all masks and CP sizes (geometric mean).

**Analysis.** FlashMask-3 achieves **1.26**× mean total throughput over MagiAttention, **2.42**× over Megatron CP (ring), and **2.05**× over Megatron CP (all-gather). The forward pass benefits most from the balanced-overlap protocol (up to 4.12× over Ring), as communication latency is fully hidden behind computation. In the backward pass, FlashMask-3 retains a consistent ∼2.2–2.4× advantage over both Megatron variants.

Table 5: FlashMask-3 speedup over distributed baselines (aggregated over all masks × CP sizes).

| vs. Baseline | Phase | Min | Max | Geo. Mean ± std |
|---|---|---|---|---|
| MagiAttention | FWD | 1.04× | 2.39× | 1.56 ± 0.53× |
| | BWD | 1.10× | 1.19× | 1.15 ± 0.03× |
| | Total | 1.09× | 1.46× | 1.26 ± 0.15× |
| Megatron Ring | FWD | 1.62× | 4.12× | 2.50 ± 0.97× |
| | BWD | 1.86× | 3.57× | 2.38 ± 0.56× |
| | Total | 1.80× | 3.71× | 2.42 ± 0.65× |
| Megatron AllGather | FWD | 1.31× | 2.24× | 1.57 ± 0.31× |
| | BWD | 1.85× | 3.15× | 2.21 ± 0.43× |
| | Total | 1.71× | 2.92× | 2.05 ± 0.39× |

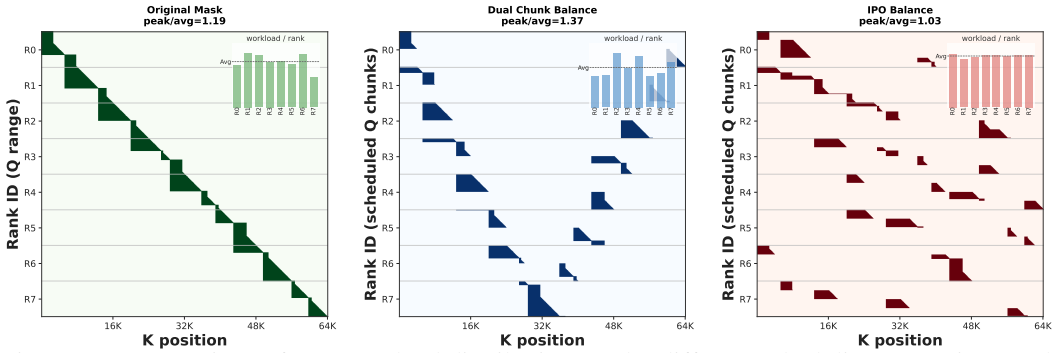



Figure 11: Comparison of compute load distributions under different scheduling strategies. Dark regions indicate work blocks assigned to each rank. The inset (upper-right of each subplot) shows per-rank compute load, with the dashed line denoting the average. The "peak/avg" in each title reports the peak-to-average load ratio across all 8 ranks; lower values indicate more balanced distributions.

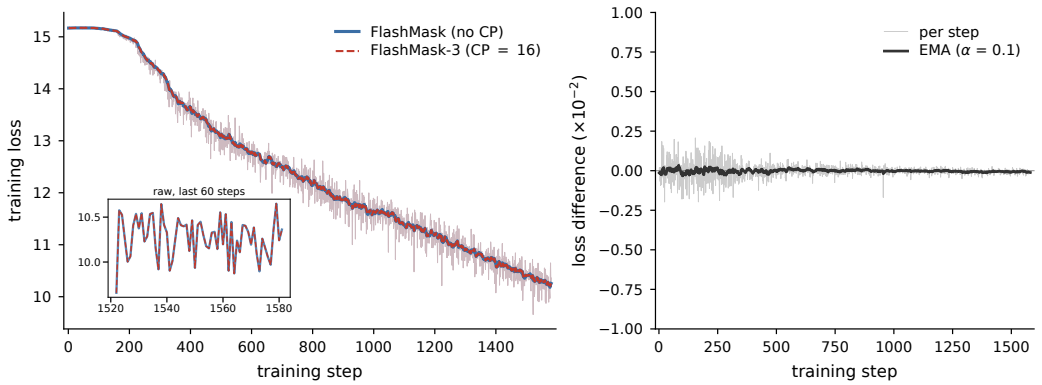
### D.2 Ablation study on different scheduling strategies

To further illustrate the effect, Figure 11 visualizes the per-rank mask layout under three scheduling strategies on a Causal Document Mask with $P$=8. The original contiguous assignment exhibits moderate imbalance (peak/avg = 1.19). Dual Chunk worsens it to 1.37 because its position-based pairing cannot account for mask-dependent per-chunk costs. IPO reduces peak/avg to 1.03 by reassigning chunks according to their sparsity-aware workload, achieving near-perfect compute balance across all ranks.

### D.3 End-to-end convergence check

§4.3 argues that FlashMask-3 changes only how attention work is placed and scheduled, never what is computed. We verify this end to end rather than assume it. The baseline runs FlashMask without context parallelism and ours runs FlashMask-3 at CP degree 16, with the same data order, the same initialization, and the same hyperparameters in both, so any divergence would have to originate in the attention path.

Figure 12 covers the 1,581 steps the two runs have in common. The loss curves are indistinguishable over the whole run: they leave the initial plateau at the same step, descend at the same rate, and reproduce the same step-to-step fluctuations, which the inset shows at full temporal resolution. The residual difference in panel (b) remains at the $10^{-3}$ level throughout and narrows as training proceeds, i.e. two orders of magnitude below the fluctuation of the loss itself between consecutive steps. A discrepancy of this magnitude is consistent with a reordered floating-point reduction rather than with a change to the optimization trajectory. Convergence over this horizon is therefore unaffected by context parallelism, as the exactness argument predicts.




(a) training loss (b) FlashMask-3 − FlashMask

Figure 12: Convergence under context parallelism. **(a)** Training loss of the two runs; faint lines are per-step values and bold lines an exponential moving average ($\alpha$=0.1), and the inset magnifies the raw curves over the last 60 steps. **(b)** Per-step loss difference between the two runs and its moving average, on the same step axis.

Table 6: Single-GPU throughput on five masks supported by FA3 (total forward+backward TFLOPs/s, mean ± std over 50 iterations). Parentheses give the ratio to FA3.

| Seq | Mask | FA3 | FlashMask-3 | FlexAttention | MagiAttention |
|---|---|---|---|---|---|
| 8K | Full | 604.43 ± 1.27 | 569.84 ± 1.88 (0.94) | 394.15 ± 1.34 (0.65) | 513.66 ± 8.95 (0.85) |
| 8K | Causal | 579.73 ± 0.35 | 520.85 ± 0.54 (0.90) | 341.68 ± 1.45 (0.59) | 482.17 ± 1.71 (0.83) |
| 8K | Sliding Window | 407.23 ± 1.54 | 290.30 ± 1.04 (0.71) | 218.93 ± 1.73 (0.54) | 166.22 ± 0.15 (0.41) |
| 8K | Causal Document | 478.96 ± 27.20 | 377.86 ± 35.26 (0.79) | 264.06 ± 18.37 (0.55) | 247.92 ± 47.43 (0.52) |
| 8K | Document | 546.29 ± 17.93 | 460.49 ± 31.77 (0.84) | 299.46 ± 26.68 (0.55) | 352.82 ± 48.32 (0.65) |
| 32K | Full | 616.65 ± 0.65 | 603.63 ± 1.48 (0.98) | 408.22 ± 0.02 (0.66) | 593.40 ± 1.30 (0.96) |
| 32K | Causal | 611.66 ± 0.59 | 587.87 ± 0.88 (0.96) | 371.48 ± 0.47 (0.61) | 563.30 ± 3.20 (0.92) |
| 32K | Sliding Window | 534.47 ± 1.71 | 465.00 ± 0.68 (0.87) | 309.52 ± 2.77 (0.58) | 378.13 ± 0.81 (0.71) |
| 32K | Causal Document | 532.68 ± 6.32 | 435.54 ± 15.28 (0.82) | 294.92 ± 8.42 (0.55) | 328.81 ± 21.48 (0.62) |
| 32K | Document | 592.03 ± 5.04 | 503.66 ± 6.68 (0.85) | 312.10 ± 19.68 (0.53) | 431.37 ± 17.05 (0.73) |
| 128K | Full | 603.12 ± 1.14 | 602.06 ± 0.92 (1.00) | 415.16 ± 0.40 (0.69) | 595.04 ± 2.00 (0.99) |
| 128K | Causal | 618.52 ± 0.48 | 587.92 ± 1.33 (0.95) | 377.21 ± 0.01 (0.61) | 580.09 ± 0.34 (0.94) |
| 128K | Sliding Window | 592.51 ± 0.46 | 561.46 ± 1.40 (0.95) | 358.07 ± 0.40 (0.60) | 514.87 ± 2.36 (0.87) |
| 128K | Causal Document | 591.87 ± 3.64 | 542.68 ± 6.73 (0.92) | 352.92 ± 3.70 (0.60) | 498.60 ± 8.79 (0.84) |
| 128K | Document | 615.62 ± 1.85 | 575.86 ± 4.11 (0.94) | 367.07 ± 7.90 (0.60) | 546.12 ± 3.83 (0.89) |

### D.4 Comparison with FlashAttention-3

Figure 5 already uses FA3 [Shah et al., 2024] as the kernel for the Full and Causal masks. Table 6 adds a direct head-to-head comparison on five masks that FA3 supports, with FlexAttention and MagiAttention measured against the same reference. Values are total forward+backward TFLOPs/s, mean ± one standard deviation over 50 iterations, and the ratio to FA3 is given in parentheses.

FA3 specializes the masks it supports, so we do not expect FlashMask-3 to overtake it there; the question is what the general mask representation costs. On Full and Causal the ratio is 0.90 to 1.00 and improves with sequence length, reaching parity at 128K on Full. On Sliding Window and the two document masks it is lower at 8K and recovers to 0.92 to 0.95 at 128K, the regime long-context training operates in. Against the methods that accept masks outside the FA3 set the margin is large: FlexAttention stays between 0.53 and 0.69 throughout, and MagiAttention falls further behind on the structured and shorter cases, down to 0.41 on Sliding Window at 8K.

A second question is how much of the single-GPU gain comes from moving to Hopper rather than from the warp specialization of §3.1. Porting the Ampere FlashMask kernel onto the FA3 pipeline is not straightforward on its own: the inline mask-index computation spills registers and fills the

Table 7: Isolating the warp specialization from the Hopper port (forward TFLOPs/s, mean ± std). Parentheses give the ratio to FlashMask-3.

| Seq | Mask | FlashMask-3 | FlashMask | FA3-style Hopper port |
|---|---|---|---|---|
| 8K | Full | 596.53 ± 2.26 | 384.88 ± 1.84 (0.65) | 214.01 ± 0.23 (0.36) |
| 8K | Causal | 553.31 ± 1.13 | 373.46 ± 1.37 (0.67) | 241.86 ± 0.13 (0.44) |
| 8K | Sliding Window | 432.15 ± 6.60 | 256.63 ± 0.07 (0.59) | 35.77 ± 0.07 (0.08) |
| 8K | Causal Document | 483.97 ± 29.40 | 302.03 ± 15.27 (0.62) | 64.77 ± 15.83 (0.13) |
| 8K | Document | 532.26 ± 24.90 | 287.89 ± 34.54 (0.54) | 55.03 ± 14.14 (0.10) |
| 32K | Full | 614.19 ± 1.15 | 388.92 ± 1.84 (0.63) | 214.22 ± 0.08 (0.35) |
| 32K | Causal | 593.97 ± 0.87 | 379.46 ± 1.65 (0.64) | 244.95 ± 0.14 (0.41) |
| 32K | Sliding Window | 553.66 ± 2.78 | 274.60 ± 0.54 (0.50) | 31.20 ± 0.08 (0.06) |
| 32K | Causal Document | 515.48 ± 10.22 | 256.39 ± 9.50 (0.50) | 25.29 ± 2.81 (0.05) |
| 32K | Document | 544.56 ± 5.93 | 191.12 ± 36.15 (0.35) | 21.42 ± 2.48 (0.04) |
| 128K | Full | 613.49 ± 2.15 | 390.66 ± 0.19 (0.64) | 211.51 ± 0.53 (0.34) |
| 128K | Causal | 544.83 ± 0.55 | 359.79 ± 0.52 (0.66) | 245.99 ± 0.02 (0.45) |
| 128K | Sliding Window | 573.63 ± 0.94 | 274.44 ± 0.67 (0.48) | 29.96 ± 0.01 (0.05) |
| 128K | Causal Document | 554.50 ± 9.57 | 252.96 ± 9.03 (0.46) | 22.15 ± 2.44 (0.04) |
| 128K | Document | 577.55 ± 3.33 | 217.27 ± 40.17 (0.38) | 18.97 ± 2.08 (0.03) |

Table 8: Standalone scheduling latency against attention time ($P$=32, chunk 2048).

| Mask | Attention (ms) | Schedule (ms) |
|---|---|---|
| Causal Document | 35.08 ± 4.19 | 42.75 ± 8.90 |
| Document | 67.44 ± 8.78 | 23.52 ± 6.34 |
| Prefix LM Document | 35.84 ± 4.29 | 43.71 ± 6.33 |

Table 9: Balance preprocessing inside the 128-GPU run of §4.3.

| Sparsity | Step (ms) | Preprocess (ms) | Overhead (%) |
|---|---|---|---|
| 70% | 20014.2 ± 1317.6 | 119.9 ± 16.0 | 0.60 |
| 80% | 17887.3 ± 2020.8 | 110.1 ± 10.2 | 0.62 |
| 86% | 16174.8 ± 2146.6 | 120.6 ± 10.4 | 0.75 |

pipeline with non-MMA instructions. Table 7 compares the original FlashMask, an FA3-style Hopper port without the KV Index Generator, and the full kernel. We report the forward pass only: the register pressure appeared during the port itself and we did not implement a backward pass for that configuration, so forward is where all three are measured on equal footing.

The port alone does not deliver the gain. On Full and Causal it reaches 0.34 to 0.45 of the full kernel, below even the Ampere kernel it was meant to modernize, and on the structured masks it collapses to 0.03 to 0.13. The KV Index Generator removes this bottleneck by moving mask-index generation onto reclaimed producer threads, off the MMA critical path. The improvement is therefore attributable to the warp specialization rather than to the change of architecture, and a direct port is not a viable route to an expressive mask-aware kernel on Hopper.

### D.5 Scheduling cost and overlap robustness

Both the per-chunk weights $w_i$ (§C.2) and the HCC-IPO assignment operate on the four boundary vectors rather than on a dense mask, so their cost grows linearly with the sequence length and not with the quadratic tile count. Their latency is also not on the critical path. For a fixed set of weights the assignment is deterministic, so we compute it ahead of time and overlap it with the compute and communication of the previous iteration, and reuse it across layers through a cache keyed on the mask pattern.

The cost is not negligible in absolute terms, which is why it is prefetched rather than run inline. Table 8 reports the standalone scheduling latency at $P$=32 with chunk size 2048 next to the attention time under the same configuration: 23 to 44 ms of scheduling against 35 to 67 ms of attention, the same order of magnitude. Table 9 measures the same preprocessing inside the 128-GPU configuration of §4.3, in a separate run, where it accounts for 0.60% to 0.75% of the step. Prefetching therefore keeps the scheduling latency out of the end-to-end time.

Computing the assignment offline has two consequences. First, for a model whose layers use different mask types we do not switch the assignment per layer. We score each mask type, take a weighted sum over the proportion of those types in the stack, and solve HCC-IPO once on that aggregate profile.

When the mask type changes often from layer to layer, a single static assignment fits each individual layer less well, and re-solving at that frequency would add latency that prefetching cannot hide. Second, HCC-IPO does not replace the default layout unconditionally: we compare the predicted cost of the unified assignment against the Dual Chunk layout and keep HCC-IPO only when it is faster, so balanced workloads such as full or causal masks stay on Dual Chunk and pay no rebalancing cost.

The overlap depends on having enough attention compute to hide the all-gather and reduce-scatter, and very short or very sparse contexts leave less to hide behind. This is the amortization caveat of Appendix A. The degradation is graceful rather than abrupt: the stream coordination of Appendix C.4.2 keeps kernel ordering deadlock-free across sparsity levels, so in the worst case the overlap recovers less of the communication instead of failing.

## NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

**The checklist answers are an integral part of your paper submission.** They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: [Yes]

   Justification: Refer to the Abstract and Introduction sections.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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: Refer to the Conclusion, Limitations, and Broader Impact sections.

Guidelines:

- The answer [N/A] 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: Refer to Appendix B

   Guidelines:

   - The answer [N/A] 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: In the Experiments section, we discussed the methods used in our experiments, and our code is open-sourced at `https://github.com/PaddlePaddle/FlashMask`. It can be replicated following the instructions in the documentation in the repository.

   Guidelines:

   - The answer [N/A] 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: Our code is open-sourced at `https://github.com/PaddlePaddle/FlashMask`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: We provide detailed descriptions in the Experiments section.

   Guidelines:

   - The answer [N/A] 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: As stated at the beginning of the Experiments section, speed measurements are repeated and their variability is reported as one standard deviation (1-sigma): error bars in the figures and ± values in the tables denote ±1 std.

   Guidelines:

   - The answer [N/A] 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: We provide detailed descriptions in the Experiments section.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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: [N/A]

    Justification: There is no societal impact of the work performed

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

- The answer [N/A] 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 [N/A] 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: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] 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: [N/A]

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

Guidelines:

- The answer [N/A] 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: [N/A]

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

    Guidelines:

    - The answer [N/A] 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.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The LLM is used only for writing, editing and does not impact the core methodology in this paper.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.