

# WallFacer: Harnessing Multi-dimensional Ring Parallelism for Efficient Long Sequence Model Training

Ziming Liu

*National University of Singapore*

Shenggan Cheng

*National University of Singapore*

Kai Wang

*National University of Singapore*

James Demmel

*University of California, Berkeley*

Shaoyu Wang

*University of Southern California*

Zhongkai Zhao

*National University of Singapore*

Xuanlei Zhao

*National University of Singapore*

Yang You

*National University of Singapore*

## Abstract

Training Transformer models on long sequences in a distributed setting poses significant challenges in terms of efficiency and scalability. Current methods are either constrained by the number of attention heads or excessive communication overheads. To address this problem, we propose **WallFacer**, a multi-dimensional distributed training system for long sequences, fostering an efficient communication paradigm and providing additional tuning flexibility for communication arrangements. Specifically, WallFacer introduces an extra parallel dimension to substantially reduce communication volume and avoid bandwidth bottlenecks. Through comprehensive experiments across diverse hardware environments and on both Natural Language Processing (NLP) and Computer Vision (CV) tasks, we demonstrate that our approach significantly surpasses state-of-the-art methods that support near-infinite sequence lengths, achieving performance improvements of up to 77.12% on GPT-style models and up to 114.33% on DiT (Diffusion Transformer) models.

## 1 Introduction

Over the past decade, Transformer[41] models have made remarkable strides in diverse fields, including computer vision (CV) and natural language processing (NLP). As the technology has evolved, the ability to efficiently process long sequences with Transformer has emerged as a pivotal challenge. For instance, in text summarization, the ability to handle extensive sequences is vital, as the content to be summarized can range from lengthy chapters to entire books [19, 3]. Similarly, chat-based applications, such as ChatGPT [1], require the capacity to process extensive dialogue histories to ensure conversational consistency. There are also applications in other fields like video generation[5, 33] and protein structure prediction[16, 7].

The long context in the above scenarios has introduced several challenges for model training and inference: 1) **Efficiency**

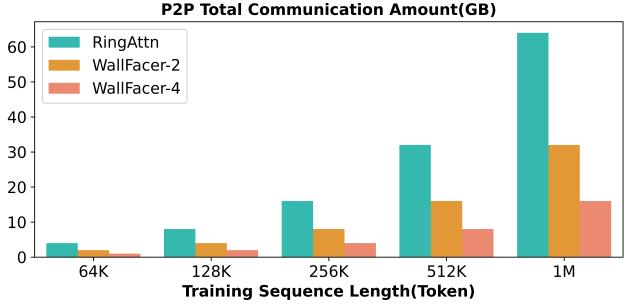


Figure 1: WallFacer-2 and WallFacer-4 theoretically save around 50% and 75% of total P2P communication volume for various sequence length.

**and Adaptability.** The challenge of efficiency predominantly lies in handling long sequences that require quadratic computations during attention, and in addressing the large amount of communication during distributed processing. 2) **Memory.** Besides the major obstacle of storing the model weight and optimizer states, the activation has also exceeded the capacity of a single GPU and risen as a new memory challenge due to the extreme sequence length. 3) **Scalability.** Current Transformer models usually require thousands of GPUs for pre-training, even with datasets of regular lengths. For longer sequences, ensuring an acceptable scaling speedup rate with both the sequence length and the number of GPUs increasing is even more critical to reducing time and economic costs.

Traditional parallelisms such as Data Parallelism[12, 40, 25, 44], Tensor Parallelism[40, 42, 43], and Pipeline Parallelism[14, 11, 24, 27] distribute the model, input batch, and the optimizer states, but can not directly address the large memory requirement of extremely long sequences as the sequence length dimension remains unchanged. To break through this obstacle, Sequence Parallelism has been introduced, splitting the input on the sequence length dimension. Mainstream Sequence Parallelism schemes can generally be classified into

two categories: those based on all-to-all communication, and those based on ring peer-to-peer communication. Methods like DeepSpeed Ulysses[15], which are based on all-to-all communication, offer efficiency but require the splitting of attention heads. Consequently, these methods are limited in scalability and can not be scaled to more devices than the number of attention heads. On the other hand, peer-to-peer communication methods[26, 22], such as Ring Attention[26], do allow for near-infinite context lengths; however, they require the transmission of complete keys and values across all GPUs, leading to significantly high communication loads. In summary, there remains a deficiency in communication-efficient methods that are capable of supporting near-infinite context lengths.

Table 1: Meanings of the symbols that are used in this paper

$P$	The number of GPUs
$C$	The parallel size of WallFacer (attention parallel size)
$H$	The hidden dimension size of the Transformer blocks
$N$	The total number of tokens within the whole sequence
$B$	The training batch size
$W$	The communication bandwidth between GPUs
$L$	The communication latency between GPUs

Inspired by the methodology of n-body communication optimization [9], we introduce WallFacer, a novel near-infinite-context Transformer training system with multi-ring sequence parallelism that incorporates an additional parallel dimension into the existing ring-style communication. Specifically, instead of including all GPUs in a single parallel group as done in Ring Attention [26], WallFacer groups the GPUs into teams and divides the peer-to-peer communication within these teams. This approach fosters an efficient communication paradigm and provides extra tuning flexibility for communication arrangements. With very little additional memory cost, WallFacer parallelism significantly reduces the peer-to-peer communication volume, as shown in Figure 1. Compared to previous works, WallFacer is not limited in supported sequence length by attention heads like DeepSpeed Ulysses[15] and Megatron Sequence Parallelism[20], and also shows better communication efficiency and scalability than Ring Attention[26].

In summary, our paper presents these contributions:

- We introduce a near-infinite-context training system for Transformer models, featuring a groundbreaking multi-ring sequence parallelism scheme. This scheme adds an additional dimension of parallelism and significantly reduces peer-to-peer communication, all while maintaining a minimal memory footprint.
- We offer a straightforward grid-search method that allows users to select the most suitable parallelism scheme based on their specific needs, maximizing the utility of

the available tuning space within our communication framework.

- We perform experiments on mainstream Transformer models, including GPT-style[36] and DiT-style[34], conducting performance and scaling tests across various computing clusters. Preliminary results indicate that our WallFacer system outperforms Ring Attention by up to 77.12% on the GPT model and up to 114.33% on the DiT model, showcasing its efficacy and scalability.

## 2 Background and Motivation

### 2.1 Transformer models and Long Sequence Training

Transformer-based models have become the mainstream approach in both Natural Language Processing (NLP) and Computer Vision (CV). GPT-style models [36] lead the field in NLP tasks, generating tokens in an auto-regressive manner. Meanwhile, the Diffusion Transformer (DiT) [34], which employs pure Transformer blocks within traditional diffusion processes, has gained widespread use over the past year for image and video generation. The key mechanism behind these Transformer-based models is attention[41], which captures the text feature by calculating the attention score between every two single tokens. A standard attention function is given as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

where  $Q$ ,  $K$ , and  $V$  are the query, value, and key of input  $x$  with shape  $(B, N, H)$  in the case of self-attention, and  $d_k$  represents the head dimension in commonly used multi-head attention. For convenience, we use the symbolic representations in Table 1. We can observe that in the original attention function, we need to store the intermediate value  $QK^T$  of size  $N^2$ , which means that the memory capacity requirement and computation amount both grow in quadratic proportion with the sequence length. To solve the memory IO and capacity bottleneck, methods like flash-attention[8] have been proposed. Flash-attention is designed upon the idea of softmax decomposition [18, 29, 35], enabling online softmax computation by storing some extra statistics. Flash-attention is thus capable of segmenting the  $Q$ ,  $K$ , and  $V$  matrices into blocks, allowing the entire computation to be performed within a single kernel for each block sequentially. This strategy effectively circumvents redundant read and write operations to the High Bandwidth Memory (HBM).

However, the sequence length can reach hundreds of thousands, when dealing with multi-round chatting, or high-resolution long video generation. It then becomes necessary to distribute the sequence across multiple GPUs. This distribution helps to reduce both the memory and computation demands on any single device. This strategy is also known

as sequence parallelism. We will explore these methods in further detail in the following section.

## 2.2 Sequence Parallelism

Sequence parallelism was first conceptualized in a study published in 2021 [23], which introduced the concept as a strategy to distribute computational sequences across multiple processing steps. Presently, Sequence parallelism can be divided into two main categories: attention-head-sharding-based and peer-to-peer-communication-based. The former involves distributing the attention heads of multi-head attention across multiple GPUs, whereas the latter resembles a distributed version of flash-attention, relying on peer-to-peer communication to transfer keys, values, and intermediate statistics.

### 2.2.1 Attention-Head-Sharding-Based

Here we introduce the two representative methods, Deep-Speed Ulysses and Megatron Sequence Parallelism.

**DeepSpeed Ulysses.** DeepSpeed Ulysses[15] presents a strategy for training with long sequences by leveraging all-to-all collective communication, but is largely limited in its scalability. Within the attention block, Ulysses transitions from sequence parallelism to a method akin to tensor parallelism with two all-to-all communication. It divides the query, key, and value matrices across the attention heads, thereby preserving the original attention computation structure.

The principal merit of Ulysses lies in its simplicity and ease of implementation. Nonetheless, its reliance on the number of attention heads for partitioning activations introduces the following limitations. 1) Ulysses is limited in scalability, as it can only be expanded to as many GPUs as there are attention heads, which restricts the maximum sequence length that can be processed. This limitation becomes particularly problematic when employing techniques like grouped-query attention (GQA) [2] or multi-query attention (MQA) [39], which further reduce the number of available attention heads for Keys and Values. 2) Ulysses encounters load-balancing issues when the number of attention heads is not evenly divisible by the number of GPUs, complicating its deployment across diverse hardware configurations. 3) The need for head-splitting makes Ulysses hard to combine with Tensor Parallelism.

**Megatron Sequence Parallelism.** Megatron Sequence Parallelism focuses on minimizing memory usage and reducing the necessity for activation recomputation. The training approach is based on Tensor Parallelism (TP). While TP is employed in the Linear and Self-Attention blocks, operations such as the LayerNorm and Dropout functions are not distributed, leading to replications within the tensor parallel group. This requires an additional all-gather operation and a further reduce-scatter operation during the forward propagation to alternate between Tensor Parallelism and Sequence Parallelism. The Self-Attention and MLP layers continue to

rely on Tensor Parallelism and must process the full sequence length. Like DeepSpeed Ulysses, it is also limited by the number of attention heads.

### **2.2.2 Ring-peer-to-peer-communication-based**

The primary method in peer-to-peer-communication-based strategies is Ring Attention[26].

**Ring Attention.** Introduced in 2023, Ring Attention[26] innovatively partitions the sequence dimension and utilizes a ring-style peer-to-peer (P2P) communication pattern to transfer Keys and Values across all GPUs. Each GPU receives the key and value matrices from the preceding rank, updates the local attention score, and then forwards them to the next rank, as is shown in Figure 2. This method employs an online-softmax and updates attention scores incrementally, allowing the computation of attention scores without retaining the full sequence length. Thus, it potentially supports infinite context, provided sufficient computing resources are available. However, the requirement for  $P$  steps of P2P communication renders this approach less efficient in environments with high-latency communication.

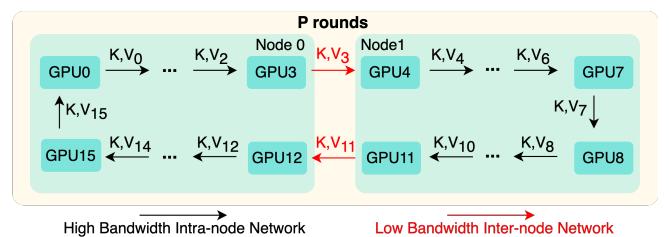


Figure 2: An example of Ring Attention Computation on 16 GPUs in two nodes. The Communication is largely limited by the inter-node bottleneck.

## 2.3 Motivation

Through observation, we identify two main drawbacks of Ring Attention. First, the communication overhead is exceedingly high because every GPU in the system must send and receive keys and values for nearly the entire sequence length before completing the attention computation. Second, variations in bandwidth between and within computing nodes can cause communication bottlenecks. As illustrated in Figure 2, the bandwidths between GPUs 3 and 4 and between GPUs 11 and 12 are lower than those between other GPUs in the ring. Despite this, the system is forced to operate in a complete circle, which can result in unnecessary idle times for other GPUs.

Inspired by techniques used in n-body simulations [13], we propose an approach that addresses both challenges simultaneously. In n-body simulations, such as those calculating

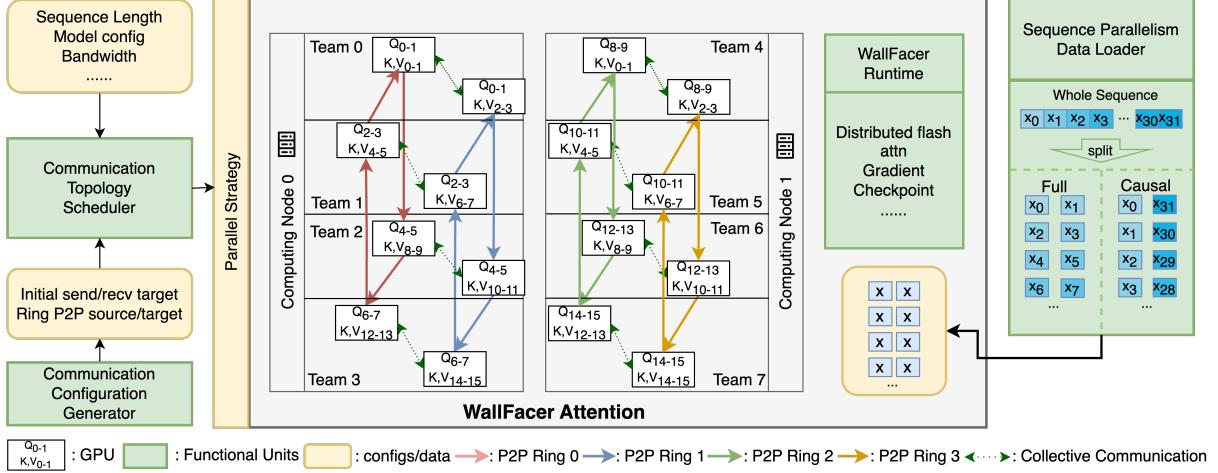


Figure 3: An overview of the WallFacer Training System

direct forces between particles (e.g., gravitational forces), researchers discovered that communication overhead can be minimized by introducing additional dimensions of communication [9]. Similarly, we propose grouping GPUs into teams within the ring-style sequence parallelism framework and distributing communication tasks among team members. This approach reduces the overall communication volume and mitigates the bandwidth bottlenecks previously mentioned. Building upon this concept, we developed the WallFacer training system, which we will detail in the following section.

### 3 WallFacer Training System

#### 3.1 Overview

The WallFacer training system is comprised of five main components. At the core of the system is WallFacer Attention, which leverages multiple ring-style P2P (peer-to-peer) communication strategies to enhance the efficiency of distributed attention computation. The Communication Configuration Generator plays a critical role in initially assigning keys and values to their corresponding ranks. Meanwhile, the Communication Topology Scheduler outlines the placement of parallelism across various computing nodes. The Dataloader is designed to organize tokens within each sub-sequence according to their mask (causal or full) and distribute them across different GPUs. Finally, the WallFacer Runtime incorporates additional supporting techniques for the training process, such as gradient checkpointing. We will now explore the details of these components in depth.

#### 3.2 WallFacer Attention

As discussed in the previous section, a major limitation of Ring Attention is the extensive amount of peer-to-peer (P2P)

communication required, which becomes problematic in environments with weak connections between computing nodes. We enhance the ring sequence parallelism by introducing an additional dimension. This is achieved by duplicating the queries, keys, and values, thus dividing the communication tasks and distributing them within each team. This segmentation markedly improves computational efficiency and scalability.

We will now go through the specifics of the WallFacer Parallelism and provide a theoretical analysis demonstrating its advantages over Ring Attention.

##### 3.2.1 Training Process of WallFacer Parallelism

In the WallFacer system, GPUs are grouped into *Teams* to coordinate computation and communication tasks more efficiently, unlike the isolated operations in traditional settings. The queries, keys, and values are gathered within the team, so each member of the team holds the same activation of the whole team before attention. WallFacer introduces an additional parameter,  $C$ , which determines the replication factor of the input and, consequently, the number of GPUs within each team. The range of  $C$  is from 1 to  $\sqrt{P}$ . When  $C$  equals one, the algorithm falls back to Ring Attention. When  $C$  equals  $\sqrt{P}$ , the algorithm becomes a completely collective-communication-based one with no rings. When  $1 < C < \sqrt{P}$ , it becomes a structure with multiple rings looping concurrently. There are three main distinctions between the operational processes of WallFacer and Ring Attention: QKV\_matmul & all-gather, attention iteration, and reduce-scatter. Throughout the attention process, asynchronous communication is employed alongside the early launch of communication kernels to maximize the overlap of computation and communication tasks. The complete forward process of a WallFacer Transformer block is outlined in Algorithm 1.

**Forward Propagation.** In Figure 4, we have an example

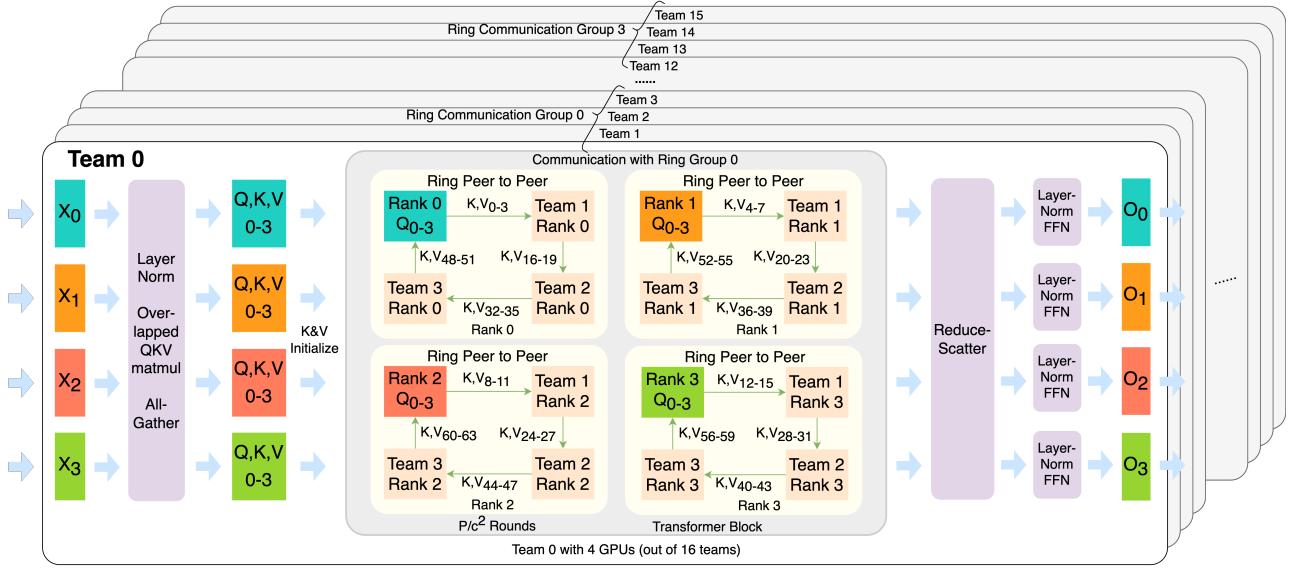


Figure 4: An example of WallFacer Attention on 64 GPUs. Each team member forms a sub-ring with members with the same local rank from other teams in the same ring communication group, reducing the communication volume of each team member by 75%.

---

#### Algorithm 1 WallFacer Attention Block (Forward)

---

**Require:** Input sequence  $\mathbf{x}$ , Linear Function **query**, **key**, and **value**, attention parallelism size  $\mathbf{c}$ , global rank  $\mathbf{r}$ , global size  $\mathbf{gs}$ , team process group  $\mathbf{pg}$ , init send/recv target  $\mathbf{r}_{\text{send}}$  and  $\mathbf{r}_{\text{recv}}$

- 1: compute the gathered  $\mathbf{q}_{\text{team}}, \mathbf{k}_{\text{team}}, \mathbf{v}_{\text{team}} = \text{AllGather\_QKVmatmul}(\mathbf{query}, \mathbf{key}, \mathbf{value}, \mathbf{x}, \mathbf{pg})$
- 2: launch the asynchronous send and receive request  $\mathbf{req}_{\text{send}}$  and  $\mathbf{req}_{\text{recv}}$ , sending  $\mathbf{k}_{\text{team}}, \mathbf{v}_{\text{team}}$  to  $\mathbf{r}_{\text{send}}$  and receiving  $\mathbf{k}_{\text{next}}, \mathbf{v}_{\text{next}}$  from  $\mathbf{r}_{\text{recv}}$
- 3: get the ring P2P target  $\mathbf{r}_{\text{next}}$  and  $\mathbf{r}_{\text{last}}$  with  $\text{get\_P2P\_ranks}(\mathbf{r}, \mathbf{gs}, \mathbf{c})$
- 4: initialize attention score  $\mathbf{O}$ , extra statistics  $\mathbf{lse}$  to zero.
- 5: **for**  $1 \leq i \leq \text{world\_size}/c^2$  **do**
- 6:   wait for  $\mathbf{req}_{\text{send}}$  and  $\mathbf{req}_{\text{recv}}$
- 7:    $\mathbf{k}_{\text{current}} = \mathbf{k}_{\text{next}}, \mathbf{v}_{\text{current}} = \mathbf{v}_{\text{next}}$
- 8:   launch  $\mathbf{req}_{\text{send}}$  to send  $\mathbf{k}_{\text{current}}$  and  $\mathbf{v}_{\text{current}}$  to  $\mathbf{r}_{\text{next}}$ , launch  $\mathbf{req}_{\text{recv}}$  to receive  $\mathbf{k}_{\text{next}}$  and  $\mathbf{v}_{\text{next}}$  from  $\mathbf{r}_{\text{last}}$
- 9:   calculate  $\mathbf{lse}, \mathbf{O} = \text{forward\_iteration}(\mathbf{lse}, \mathbf{O}, \mathbf{q}_{\text{team}}, \mathbf{k}_{\text{current}}, \mathbf{v}_{\text{current}})$
- 10: **end for**
- 11: compute  $\mathbf{O}_{\text{final}} = \text{ReduceScatter\_combine}(\mathbf{lse}, \mathbf{O}, \mathbf{pg})$
- 12: return  $\mathbf{O}_{\text{final}}$

---

of one team of four GPUs out of all the 64 GPUs performing WallFacer-style attention. Each training iteration begins with the dataloader splitting the entire input sequence of length  $N$  into  $N/P$  sub-sequences, which are then loaded onto each GPU. As previously mentioned, the next step involves computing the queries, keys, and values. These are computed separately via matrix multiplication, followed immediately by the launch of the all-gather kernel, which gathers the above QKVs within the team, allowing for the overlap of up to two-thirds of the communication with computation.

Once this phase is complete, each GPU within the team possesses the same Q, K, and V, each of a length of  $\frac{CN}{P}$ . To distribute the communication and computation tasks among the team members, we divide the original workload based on four specific ranks assigned to each GPU by the Communication Configuration Generator. These ranks determine each GPU's partners and position within the P2P ring, details of which will be discussed in Section 3.3.

Following the setup, the Keys and Values are dispatched to their designated locations within the cluster to establish the initial sub-ring, setting the stage for the multi-ring iteration phase of WallFacer attention. Given that each sub-sequence is  $\frac{CN}{P}$  long and each GPU is tasked with computing the attention score for  $\frac{1}{C}$  of the whole sequence, it results in  $\frac{N/C}{CN/P} = P/C^2$  rounds of communication. This implies that there are  $P/C^2$  GPUs in one ring.

The iteration process, conducted in a Flash-Attention style, involves storing the log-sum-exp (lse) and intermediate output O, which are updated step by step. Queries are retained locally, while Keys and Values circulate through the ring via P2P.

communication. After completing the iterations, each team member accumulates the attention scores for the entire team’s sub-sequence of Queries with  $1/C$  of the Keys and Values from the full sequence.

A simple reduce-scatter operation is then employed to amalgamate the intermediate results and distribute them among the team members. Each GPU ultimately contains the final attention score for its portion of the sequence over the entire sequence. The output of this forward attention block is finalized after a standard LayerNorm and FeedForward layer process.

**Backward Propagation.** The major distinction between backward and forward propagation is the inability to calculate queries independently during the backward phase. Unlike forward propagation, the backward phase requires the complete set of keys and values to calculate the gradient for queries, and vice versa. To manage this, we have structured the gradient calculation into two loops: the key & value outer loop and the query inner loop.

In the outer loop, gradients for keys and values are tracked and maintained fixed on the corresponding GPUs within the sub-rings; these gradients do not transfer between GPUs. The inner loop, however, handles the gradients for queries, which start initialized as zero and are circulated along the sub-rings together with the Queries themselves. During each iteration, the approach mirrors the backward computation method used in flash-attention[8], where the updated gradient of the current query shard is passed to the next GPU in the ring, while the gradients for keys and Values are retained for subsequent query shards.

Initial communication ranks for sending and receiving are still configured by the Communication Configuration Generator. Additionally, there is an extra P2P communication required at the end of the process to send the query and its gradient back to their original location following the loop.

### 3.2.2 Theoretical Analysis

In this section, we will discuss two major questions: how much communication reduction WallFacer can bring compared with Ring Attention, and how much extra memory cost it comes with. During the analysis, we will employ a case study using the WallFacer system with an attention parallel size of  $C = 4$  on a llama-30B model, which consists of 64 layers. For this model, referred to as model M, the batch size =  $B$  is set to 1, the sequence length =  $N$  to 65, 536, the hidden dimension =  $H$  to 6, 656, and the number of GPUs =  $P$  to 64. Additionally, the computation will utilize bfloat16 precision.

**Communication Analysis.** Let’s analyze the communication overhead within one forward Transformer block on a single GPU. For Ring Attention, the communication is primarily due to the ring P2P loop. As the total number of iterations done is  $P$ , the total communication overhead can be calculated

as:

$$p\left(\frac{2BNH}{PW} + L\right) = \frac{2BNH}{W} + PL \quad (2)$$

and this overhead can be partially overlapped with the attention computation.

For WallFacer, the communication overhead comes from both collective and P2P. The collective overhead for all-gather and reduce-scatter is:

$$\frac{4BNH(C - 1)}{PW} \quad (3)$$

while the P2P communication can be similarly computed as:

$$\frac{P}{C^2}\left(\frac{2CBNH}{PW} + L\right) = \frac{2BNH}{CW} + \frac{P}{C^2}L \quad (4)$$

The advantages of WallFacer over Ring Attention during the ring-P2P phase are evident in three main aspects: 1) **Reduced Communication and Latency:** Ring Attention requires  $C$  times more communication than WallFacer, significantly increasing the bandwidth requirement across the entire cluster. For the llama 30B model M, the total communication volume of ring P2P communication and collective communication volume for Ring Attention and WallFacer can be computed as 1.625 GB and 0.152 GB(collective) + 0.406GB (P2P) = 0.558GB. Furthermore, while Ring Attention necessitates  $P$  iterations per attention block, WallFacer only requires  $\frac{P}{C^2}$ , reducing the latency overhead by  $C^2$ . 2) **Localized Communication:** In scenarios like those depicted in Figure 3, WallFacer’s ring P2P communication can be confined within the same computing node, where bandwidth is typically much higher than between computing nodes. Conversely, Ring Attention demands inter-node communication during every iteration, which can be less efficient. 3) **Enhanced Overlap of Communication and Computation:** During each iteration, the communication volume of WallFacer is  $C$  times higher than that of Ring Attention, while the computational volume during attention is approximately  $C^2$  times greater. This higher computation-to-communication ratio makes it easier for WallFacer to overlap P2P communication with computation, enhancing overall efficiency.

Additionally, the collective communication in WallFacer is minimal due to: 1) Efficient Overlapping: The all-gather communication overlaps significantly with the QKV matrix multiplication, and the reduce-scatter communication partially coincides with the final attention score update. 2) Scale Considerations: Compared to the P2P communication column, there is a  $P$  in the denominator, which is substantial during large-scale training. This implies that collective communication constitutes a very small portion of the total communication volume.

**Memory Analysis.** In this section, we estimate the theoretical peak memory requirements necessary to store the model weights, activations, and optimizer states. Our implementation utilizes the Adam Optimizer [17], bfloat16 precision, and

Zero-2 optimization [37]. Since the WallFacer architecture does not alter the model weights or the optimizer, we can assume that the memory costs associated with the model and the optimizer remain constant across both methods. We name the memory cost for the model and optimizer as  $M_{m+o}$ . As for the activation, we refer to the size of one single activation of a sub-sequence on one GPU as

$$A = \frac{B \times N \times H}{P} \quad (5)$$

As we use the checkpointing scheme from [22], a model of  $Y$  layers needs to save  $Y + 1$  activations as checkpoints. Now we calculate the approximate peak memory after Q, K, and V are already calculated and before the attention computation at the last layer of the whole model. For Ring Attention and WallFacer, the peak memories are:

$$PM_{Ring} = M_{m+o} + (Y + 1)A + 3A = M_{m+o} + (Y + 4)A \quad (6)$$

$$PM_{Wall} = M_{m+o} + (Y + 1)A + 3CA = M_{m+o} + (Y + 3C + 1)A \quad (7)$$

, where C is the WallFacer attention dimension. And for the example model M, the peak memory would be  $M_{m+o} + 68A$  and  $M_{m+o} + 77A$ , and the extra memory cost compared with Ring Attention is less than 13.2%, while the P2P communication volume is reduced by about 75%. In a word, the extra memory cost is acceptable as a tradeoff for the communication reduction.

### 3.3 Communication Configuration Generator

The Communication Configuration Generator, as introduced in the previous section, plays a crucial role in the initial setup of queries, keys, and values on GPUs and determining each GPU's position within the sub-rings.

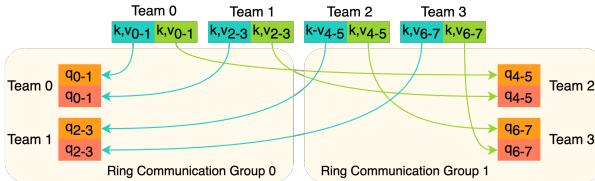


Figure 5: An example of ring initialization process of 8GPUs and 4 sub-rings in the Communication Configuration Generator

Initially, for the setup stage, it is essential to establish the sub-rings by rearranging the activation positions. Specifically, during forward propagation, the queries do not require rearrangement; however, the keys and values must be transmitted to their corresponding positions in the ring prior to commencing the loop. As illustrated in Figure 5, this initialization ensures that each team member holds a different shard of keys

and values. Moreover, it guarantees that no two teams within the same ring possess identical keys and values.

Furthermore, Team 0 and Team 1 form what is termed a **team group**, where ring P2P communication occurs exclusively within this group, similarly for Team 2 and Team 3. A team group is defined as a collection of teams that participate in the same sub-rings. The number of teams within a team group can be calculated as  $Team\ size = \frac{P}{C^2}$ . We give the details of the determination of initial sending targets in the case of placing teams within the same computing nodes in Algorithm 2, and the receiving source can also be calculated similarly.

---

#### Algorithm 2 get\_init\_send()

---

**Require:** inter-team rank  $\mathbf{r}_t$ , intra-team rank  $\mathbf{r}_a$ , inter-team dimension  $\mathbf{d}_t$ , intra-team dimension  $\mathbf{d}_a$

- 1: team group size =  $\mathbf{d}_t / \mathbf{d}_a$
- 2: target team group rank =  $\mathbf{r}_a$
- 3: target team = target team group rank \* team group size +  $\mathbf{r}_t // \mathbf{d}_a$
- 4: target device intra-team rank =  $\mathbf{r}_t \% \mathbf{d}_a$
- 5: target global rank = target team \*  $\mathbf{d}_a$  + target device intra-team rank
- 6: return target global rank

---

After the initialization of activations, we can set up the rings by providing the GPUs their last and next GPU within their rings, as is described in Algorithm 3

---

#### Algorithm 3 get\_P2P\_config()

---

**Require:** inter-team rank  $\mathbf{r}_t$ , intra-team rank  $\mathbf{r}_a$ , inter-team dimension  $\mathbf{d}_t$ , intra-team dimension  $\mathbf{d}_a$

- 1: team group size =  $\mathbf{d}_t / \mathbf{d}_a$
- 2: self team group rank =  $\frac{r_t}{team\ group\ size}$
- 3: next team in group =  $(r_t + 1)\% team\ group\ size + team\ group\ size \times self\ team\ group\ rank$
- 4: last team in group =  $(r_t - 1)\% team\ group\ size + team\ group\ size \times self\ team\ group\ rank$
- 5: next device global rank =  $r_a + next\ team\ in\ group \times d_a$
- 6: last device global rank =  $r_a + last\ team\ in\ group \times d_a$
- 7: return next device global rank, last device global rank

---

### 3.4 Communication Topology Scheduler

The communication topology scheduler employs a grid-search algorithm to discover the optimal parallelism configuration, tailored to the specifics of the current cluster, model, and input data.

When using Ring Attention, all GPUs form a single ring, which considerably limits the flexibility of adjusting the placement scheme. In contrast, WallFacer expands the tuning space with its unique multi-ring structure in two significant ways.

First, the ring configuration in WallFacer is determined by the parallelism dimension  $C \in [1, \sqrt{P}]$ . Smaller  $C$  results in less collective communication, a higher total communication volume, and more inter-node communication, while higher  $C$  leads to more communication volume during all-gather and reduce-scatter, which may exceed the overhead of the QKV matrix multiplication that we use for overlapping, but can highly reduce the overall communication volume.

Second, the additional dimension provided by WallFacer allows for two strategies in parallelism placement. One strategy is to keep the all-gather and reduce-scatter operations intra-node, which confines these processes within the same computing node. The other strategy is to keep the ring P2P communication intra-node, reducing latency and potentially enhancing performance. The choice between these strategies depends on the extent of overlapping achievable within the ring and the communication volume during collective communications.

Based on the above tuning space, we provide an auto-scheduler, grid-searching through all possible configurations for the one that provides the highest throughput, which can be described as:

$$\begin{aligned} \text{Config} = \arg \max_{C, \text{placement}} & (\text{Profile}(C \in [1, \sqrt{P}], \\ & \text{placement} \in [\text{P2P\_intra}, \text{Collect\_intra}]) \end{aligned} \quad (8)$$

The scheduler requires only a few iterations to profile the performance of automatically generated configurations. The time spent on this profiling is negligible compared to the entire training process, as it only needs to be performed once for each new computing cluster used for training.

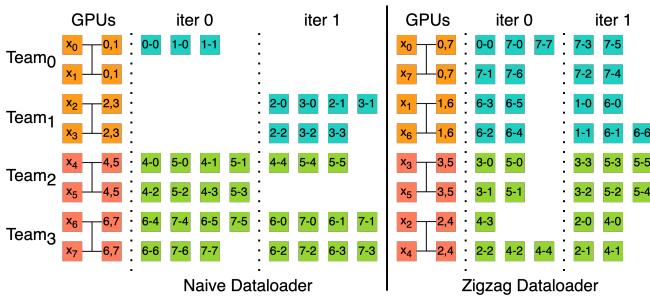


Figure 6: A comparison between naive and zigzag dataloader for 8 GPUs with attention parallel dimension of 2. The corresponding initialization can be found in Figure 5 with the same configuration. The improvement of efficiency from load-balancing increases with the number of GPUs.

### 3.5 Sequence Parallelism Dataloader

To accommodate both full mask and causal mask configurations for self-attention, we have implemented two distinct

data loading schemes for load-balancing. For full masks, sequences are straightforwardly divided into sub-sequences of equal lengths and distributed among the devices. However, causal masks present a challenge due to the unbalanced computational load across GPUs; sub-sequences at the beginning of a sequence require significantly more computation than those at the end. To address this imbalance and achieve load equilibrium among GPUs, we modify the ZigZag scheme introduced by [45], illustrated in Figure 6. The figure illustrates the simplest case of zigzag load-balancing. Notably, the effectiveness of this strategy improves as the number of GPUs increases. This improvement correlates with the expanding difference in computation volume between the first and the last token, which escalates as the sequence length extends. This approach ensures that the total workload on each GPU is balanced, eliminating the need for additional communication mechanisms like those employed in DistFlashAttention[22].

### 3.6 WallFacer Runtime

WallFacer is written in PyTorch[32] and uses the PyTorch torch.autograd.function and NCCL[31] backend for forward and backward implementation. WallFacer also employs multiple techniques during runtime to improve its overall training efficiency.

**Ingetrate Flash Attention.** The WallFacer attention mechanism involves multiple iterations that loop over Keys and Values, with each iteration still using traditional self-attention with corresponding Query, Key, and Value (QKV). This approach enables WallFacer to incorporate flash attention effectively, extending its capability by preserving intermediate states across iterations. Additionally, WallFacer enhances the efficiency of the forward process with the help of torch JIT to fuse kernels aside from flash attention.

**Overlap communication with computing.** In WallFacer attention, P2P communication and self-attention computing are interleaved across iterations, each incurring considerable time. To mitigate this, WallFacer employs a double buffering technique to asynchronously execute communication and computing kernels, effectively overlapping these processes and enhancing GPU utilization.

**Save recomputation with checkpoints.** WallFacer adopts the checkpointing strategy introduced by DistFlashAttn[22], placing checkpoints at the end of the self-attention phase rather than the FFN of each transformer layer. This checkpoint placement effectively obviates the need to recompute the self-attention forward process during the backward pass, avoiding redundant attention computation.

## 4 Evaluation

The computational resources we use in the experiments include a local Nvidia H100 cluster with eight nodes and three Nvidia A100 clusters, as listed in table 2. We utilize two



Figure 7: Throughput evaluation of Ring Attention and WallFacer on 32 GPUs from three different clusters. We place the performance of WallFacer with both C=2 and C=4 in the figure. The configurations are marked in the titles of the sub-figures. For instance, A100\_40GB\_8(1B, 512K) represents that the experiment is on machines with 8 Nvidia A100 40GB GPUs in each node, the model used has one billion parameters, and the sequence length is 512k.

model types of total three settings, as listed in table 3. For the DiT(Diffusion Transformer) model, we use similar configurations as those in Stable Diffusion 3[10]. We utilize the backbone Diffusion Transformer only, without other components like the text and image encoders. During training, both models use bfloat16 precision and a batch size of 1 to accommodate longer input sequences.

Table 2: Cluster Configurations. All GPUs are connected with NVLink with computing nodes. #GPU represents local size times the number of nodes.

GPU	#GPUs	Memory	Inter-node
Nvidia H100	8*8	80GB	InfiniBand
Nvidia A100	16*2	40GB	100Gbps Ethernet
Nvidia A100	8*4	40GB	100Gbps Ethernet
Nvidia A100	16*2	40GB	100Gbps Ethernet

Table 3: Model Configurations

Model Name	#Heads	#Layers	Hidden Dim
GPT 3B	12	16	4096
GPT 7B	32	32	4096
DiT 1B	24	24	1536

In the evaluation section, we aim to answer three major questions:

- How much improvement in throughput can WallFacer bring? Additionally, how adaptable is WallFacer to clusters with both good and poor inter-node connections?

- Is the additional memory cost incurred by WallFacer acceptable considering the throughput improvement it offers?
- How does WallFacer perform in scenarios of weak and strong scaling? Specifically, does it outperform Ring Attention when scaled to handle longer inputs?

## 4.1 Throughput and Adaptability

Our first experiment aims to assess the performance of WallFacer and Ring Attention across different clusters with varying environments, testing the adaptability of both methods. There are several factors influencing the efficiency of ring-style attention computation:

**Theoretical Computation-Communication Volume Ratio:** Primarily determined by the sequence length used during training. Attention computation exhibits a computational complexity of  $O(N^2 \cdot H)$ , whereas P2P communication complexity is  $O(N \cdot H)$ . Thus, the model configuration does not impact this ratio; only the sequence length does. A larger  $N$  increases the computation-communication ratio, facilitating easier overlap of communication with computation. We evaluate varying sequence lengths to explore performance under different ratios.

**Compute Capability and Connectivity of GPUs:** The computing overhead, given a specific volume, affects the computation-communication overhead ratio. Higher compute capabilities make overlapping more challenging. We utilize two sets of GPUs in this evaluation: Nvidia A100 40GB and Nvidia H100 80GB, with the latter offering significantly higher theoretical tflops on bf16 computations. Connectivity



Figure 8: The normalized relative memory cost of different configurations of WallFacer compared with Ring Attention on different clusters.

is considered in two parts: intra-node and inter-node. Our clusters are equipped with NVLink, ensuring robust intra-node communication. For inter-node communication, while InfiniBand offers high bandwidth at a higher commercial cost, some clusters utilize Ethernet for connectivity. Specifically, our H100 nodes leverage InfiniBand with eight adapters per node for superior inter-node bandwidth, whereas the Google Cloud servers use Ethernet. The diversity in node configurations (8-GPU and 16-GPU nodes) allows us to assess adaptability across different topologies.

This evaluation not only highlights the inherent differences between the schemes but also tests their flexibility in various hardware settings.

The results of our evaluation are illustrated in Figure 7. We measure throughput in thousands of tokens per second. To better demonstrate how to select the optimal configuration of WallFacer under each condition, we included two configurations, Wall-2 and Wall-4, in the figure. We omit configurations with lower performance for clarity. As indicated in the figure, in all six settings, at least one configuration of WallFacer achieves higher throughput than Ring Attention, with performance improvements of 114.33%, 41.4%, 62.87%, 42.45%, 35.98%, 19.9%, 77.12%, and 34.62%. This advantage is primarily due to the additional parallel dimension that WallFacer introduces. Unlike Ring Attention, which requires inter-node P2P communication in each iteration, WallFacer’s P2P communication is mostly confined intra-node, except for initial data transfers. This experiment clearly demonstrates WallFacer’s superior performance across various environments.

Another observation is that the optimal configuration for WallFacer may vary depending on the environment, reflecting differences in the computation-communication ratio and the trade-offs between collective and P2P communication. For example, the best  $C$  value for A100\_16 is 2, while for A100\_8, it is 4. This variation can be attributed to the topological differences: the 16-GPU-node cluster benefits less from reduced P2P communication volume, making a smaller  $C$  preferable to decrease collective communication volume while maintaining P2P communication at an acceptable level. This finding

Table 4: Supported Sequence Length of Ring Attention and WallFacer on one Nvidia A100 80GB GPU.

Supported Seq Len on one 80GB A100 GPU (K Tokens)			
Model Size	Length	Ring Attention	WallFacer
3B	128	✓	✓
	256	✓	✓
	512	✓	✓
7B	128	✓	✓
	256	✓	✓
	512	✗	✗
13B	128	✓	✓
	256	✗	✗
	512	✗	✗

underscores the importance of the Communication Topology Scheduler we provide, which helps users avoid the complex process of manually analyzing these factors.

In summary, WallFacer demonstrates superior efficiency and adaptability in all six cases, thanks to its flexible parallelism scheme.

## 4.2 Memory Consumption

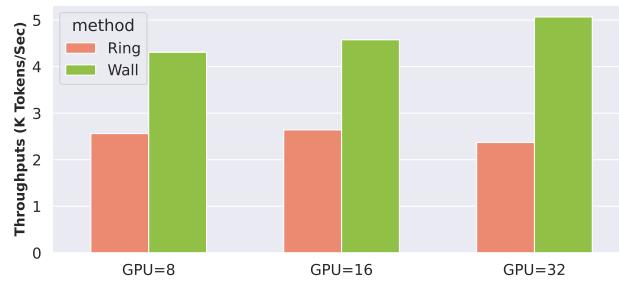
Before evaluating the memory consumption of WallFacer, we first compared the supported sequence lengths of WallFacer with those reported in the RingAttention paper [26]. As shown in Table 4, despite the slightly higher memory requirements of WallFacer, it still supports sequence lengths commonly used in training tasks.

Memory consumption in our experiments stems from both the model weights and activations, with additional costs for WallFacer arising from the duplication of QKV matrices prior to attention computation. To quantitatively assess the additional memory required by WallFacer, we monitored the maximum memory allocated by PyTorch [32] during our experi-

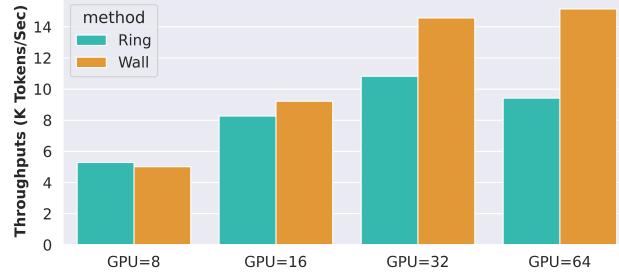
ments. It is important to note that memory fragmentation in PyTorch can impact memory allocation efficiency. To mitigate this, we limited the sequence lengths in our training to prevent PyTorch from triggering `cuda_free` when the allocated memory approaches the GPU’s limit, which would otherwise introduce significant overhead.

The results, displayed in Figure 8, reveal that for the configurations yielding the highest throughput, Ring Attention consumes between 7.9% and 30.79% less GPU memory than WallFacer, except the DiT 512k Wall-4 setting. However, considering the substantial throughput gains provided by WallFacer, the additional memory usage is deemed acceptable. We observed that the maximum supported sequence lengths were not significantly impacted by this additional memory usage.

Moreover, in scenarios involving larger models, the relative increase in memory consumption due to QKV duplication diminishes. This reduction is explained by equation 7, which shows that as the model size increases—owing to more parameters and additional Transformer layers—the proportion of memory consumed by model weights and activations grows, whereas the absolute extra memory incurred by QKV multiplication remains constant. This phenomenon is further evidenced by the fact that the extra memory ratio for experiments with the 7B model is significantly smaller than that for the 3B and 1B model.



(a) DiT Strong Scaling on Nvidia A100 40GB GPUs. All configurations include inter-node communication.

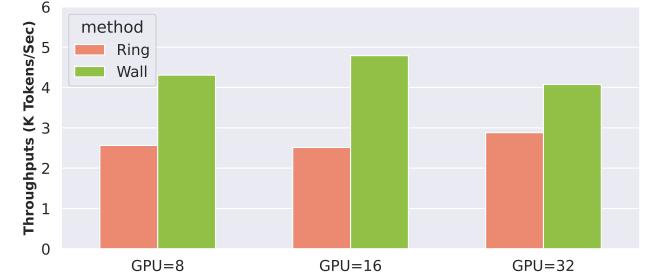


(b) GPT Strong Scaling on Nvidia H100 80GB GPUs.

Figure 9: Strong scaling experiments with fixed sequence length of 128K.

### 4.3 Strong and Weak Scaling

In the scaling tests we carry out experiments for both strong and weak scaling. Strong Scaling maintains the scale of the problem that we are trying to solve while increasing the computing resource that we use to speed it up. So in this experiment, we fix the sequence length to 128K while increasing the number of GPUs from 8 to 64 for the GPT model and from 8 to 32 for the DiT model. As is depicted in Figure 9, WallFacer shows gradually more advantage over Ring Attention as we increase the number of GPUs. When we use 64 GPUs, WallFacer shows 60.71% higher throughput than Ring Attention for the GPT model and 114.33% for the DiT model on 32 A100 GPUs. This can also be explained by the computation-communication ratio. When scaled to more GPUs, the local sequence length on each GPU becomes smaller, and as explained in the previous sections, makes it harder to overlap the P2P communication with attention computation. This is exactly the main advantage of WallFacer, reduction of total P2P communication volume. So WallFacer can give a promising performance when we are trying to speedup the training of a fixed amount of data with more GPUs.



(a) DiT Weak scaling on Nvidia A100 40GB GPUs of sequence length from 128K to 512K. All configurations include inter-node communication.



(b) GPT Weak scaling on Nvidia H100 80GB GPUs of sequence length from 64K to 512K.

Figure 10: Weak scaling Experiments

In our weak scaling experiments using DiT models, we scale the sequence length from 128k to 512k while proportionally increasing the number of GPUs from 8 to 32. Assuming a scaling factor of  $k$ , both the sequence length and the number of GPUs increase by  $k$ , resulting in a total computation

volume that increases by approximately  $k^2$ . Since the computational resources also increase by  $k$ , the theoretical overall throughput (tokens per second) should remain constant during this weak scaling setting. As shown in Figure 10, both WallFacer and Ring Attention exhibit this expected behavior, with throughput remaining relatively constant. However, WallFacer consistently achieves higher throughput across different numbers of GPUs. This is because, although the local computation-to-communication ratio stays the same in weak scaling, Ring Attention requires a higher proportion of inter-node communication as the number of GPUs increases, whereas WallFacer keeps most communication local.

In summary, WallFacer shows better scalability in both strong and weak scaling experiments, making it a better choice for large-scale Transformer model training.

## 5 Discussion

### 5.1 FlashAttention3 and Hopper GPUs

In addition to the original FlashAttention [8] used in our experiments, FlashAttention3 [38] has been introduced, specifically designed for Hopper and newer Nvidia GPUs. For FP16 precision, which is utilized in this paper, FlashAttention3 achieves a 1.5-2.0x speedup on Hopper GPUs. As discussed in Section 2.2.2, reducing attention computation overhead results in more P2P communication not being overlapped, further emphasizing the need to reduce communication volume. With the increasing adoption of Hopper GPUs, the significance of the WallFacer system will also grow.

### 5.2 WallFacer and Other Parallelisms

**Model Parallelism** As is well known, **tensor parallelism** shards activations by attention heads during attention computation, making it easily combinable with WallFacer with minimal effort. However, when combined with tensor parallelism, the need for attention heads can limit the scalability of head-based sequence parallel methods like Ulysses. **Pipeline parallelism**, on the other hand, divides the model across layers without altering the computation patterns within Transformer blocks, making WallFacer orthogonal to it.

**Other Sequence Parallelism** WallFacer is orthogonal with other attention-head-sharding-based sequence parallelism approaches, such as DeepSpeed Ulysses [15]. While Ulysses distributes attention heads across different devices, WallFacer can independently partition activations along the sequence length dimension. In future work, we can explore combining WallFacer with DeepSpeed Ulysses to expand the communication scheduling space, harnessing the scalability of WallFacer alongside the efficiency of Ulysses.

In summary, WallFacer can be seamlessly integrated with other parallel training techniques, enabling the creation of a hybrid distributed training system.

## 6 Related Work

**Attention Optimization.** Traditional full attention mechanisms necessitate  $O(n^2)$  memory for storing the outputs of  $QK^T$ , leading to significant computational and memory demands. To address these challenges within the GPU, several approaches have been devised to reduce both memory and computational requirements. Memory-efficient attention[35] introduces a straightforward algorithm that requires only  $O(1)$  memory relative to the sequence length, with an extension for self-attention that needs only  $O(\log n)$  memory. Flash Attention further minimizes I/O overhead and enhances overall efficiency. Additionally, optimization methods specifically tailored for inference, such as PagedAttention[21], are also being developed to improve the efficiency of attention computations. In this work, we utilize Flash Attention within each iteration to reduce the computation overhead.

**Long-Sequence Training Techniques.** Sequence Parallelism[23] was initially introduced to enhance the efficiency of parallel long-sequence training. Ring Attention[26] improved communication efficiency through memory-efficient methods[35], supporting near-infinite sequence lengths. DeepSpeed Ulysses[15] employs attention head splitting to achieve high efficiency, though it is constrained by the number of heads. Megatron Sequence Parallelism focuses on reducing memory costs during Tensor Parallelism, while DistFlashAttention[22] features a load-balance scheme and a novel gradient checkpoint method. Our work builds on these innovations, introducing a system that supports large-scale training with an efficient communication scheme.

**Techniques for Distributed Model Training.** Distributed model training encompasses two primary areas: 1) **Memory Management:** Various techniques aim to conserve GPU memory during distributed training, such as mixed precision training[28] and the ZeRO series[37]. In this work, we implement ZeRO-2 to manage optimizer states and gradients efficiently. 2) **Hybrid Parallelism:** Frameworks like Megatron[30] and Colossal AI[4] integrate multiple forms of parallelism. There are various existing Parallelism techniques like Pipeline Parallelism[14, 11, 24, 27] and Tensor Parallelism[40], which can be combined with WallFacer Parallelism to facilitate large-scale training. We are also considering the integration of additional frameworks such as [6] to enhance overlapping capabilities in future implementations.

## 7 Conclusion

WallFacer represents an advanced near-infinite-context Transformer model training system, featuring a communication-optimized multi-ring sequence parallelism scheme. Through comprehensive adaptability and scaling experiments, we demonstrate that our system not only achieves high efficiency across various training environments but also excels under both strong and weak scaling conditions for both Computer

Vision(CV) and Natural Language Processing(NLP) models. In an era increasingly demanding longer contexts for both NLP and CV, WallFacer is poised to make significant contributions to the industry and inspire innovative research in academia.

## References

- [1] Josh Achiam et al. *GPT-4 Technical Report*. 2023. arXiv: [2303.08774 \[cs.CL\]](https://arxiv.org/abs/2303.08774).
- [2] Joshua Ainslie et al. *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. 2023. arXiv: [2305.13245 \[cs.CL\]](https://arxiv.org/abs/2305.13245).
- [3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. *Longformer: The Long-Document Transformer*. 2020. arXiv: [2004.05150 \[cs.CL\]](https://arxiv.org/abs/2004.05150).
- [4] Zhengda Bian et al. “Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training”. In: *arXiv preprint arXiv:2110.14883* (2021).
- [5] Tim Brooks et al. Feb. 2024. URL: <https://openai.com/research/video-generation-models-as-world-simulators>.
- [6] Chang Chen et al. “Centauri: Enabling Efficient Scheduling for Communication-Computation Overlap in Large Model Training via Communication Partitioning”. In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. ASPLOS ’24. <conf-loc>, <city>La Jolla</city>, <state>CA</state>, <country>USA</country>, </conf-loc>; Association for Computing Machinery, 2024, pp. 178–191. ISBN: 9798400703867. DOI: [10.1145/3620666.3651379](https://doi.org/10.1145/3620666.3651379). URL: <https://doi.org/10.1145/3620666.3651379>.
- [7] Shenggan Cheng et al. “FastFold: Optimizing AlphaFold Training and Inference on GPU Clusters”. In: *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 2024, pp. 417–430.
- [8] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. 2022. arXiv: [2205.14135 \[cs.LG\]](https://arxiv.org/abs/2205.14135).
- [9] Michael B. Driscoll et al. “A Communication-Optimal N-Body Algorithm for Direct Interactions”. In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing* (2013), pp. 1075–1084. URL: <https://api.semanticscholar.org/CorpusID:3941907>.
- [10] Patrick Esser et al. *Scaling Rectified Flow Transformers for High-Resolution Image Synthesis*. 2024. arXiv: [2403.03206 \[cs.CV\]](https://arxiv.org/abs/2403.03206). URL: <https://arxiv.org/abs/2403.03206>.
- [11] Shiqing Fan et al. *DAPPLE: A Pipelined Data Parallel Approach for Training Large Models*. 2020. DOI: [10.48550/ARXIV.2007.01045](https://doi.org/10.48550/ARXIV.2007.01045). URL: <https://arxiv.org/abs/2007.01045>.
- [12] W Daniel Hillis and Guy L Steele Jr. “Data parallel algorithms”. In: *Communications of the ACM* 29.12 (1986), pp. 1170–1183.
- [13] Roger W. Hockney and James W. Eastwood. *Computer simulation using particles*. Hilger, 1989.
- [14] Yanping Huang et al. *GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism*. 2018. DOI: [10.48550/ARXIV.1811.06965](https://doi.org/10.48550/ARXIV.1811.06965). URL: <https://arxiv.org/abs/1811.06965>.
- [15] Sam Ade Jacobs et al. *DeepSpeed Ulysses: System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models*. 2023. arXiv: [2309.14509 \[cs.LG\]](https://arxiv.org/abs/2309.14509).
- [16] John M. Jumper et al. “Highly accurate protein structure prediction with AlphaFold”. In: *Nature* 596 (2021), pp. 583–589. URL: <https://api.semanticscholar.org/CorpusID:235959867>.
- [17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: [1412.6980 \[cs.LG\]](https://arxiv.org/abs/1412.6980).
- [18] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. *Reformer: The Efficient Transformer*. 2020. arXiv: [2001.04451 \[cs.LG\]](https://arxiv.org/abs/2001.04451).
- [19] Huan Yee Koh et al. “An Empirical Survey on Long Document Summarization: Datasets, Models, and Metrics”. In: *ACM Computing Surveys* 55 (2022), pp. 1–35. URL: <https://api.semanticscholar.org/CorpusID:250118028>.
- [20] Vijay Korthikanti et al. *Reducing Activation Recomputation in Large Transformer Models*. 2022. arXiv: [2205.05198 \[cs.LG\]](https://arxiv.org/abs/2205.05198).
- [21] Woosuk Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. 2023. arXiv: [2309.06180 \[cs.LG\]](https://arxiv.org/abs/2309.06180).
- [22] Dacheng Li et al. *DISTFLASHATTN: Distributed Memory-efficient Attention for Long-context LLMs Training*. 2024. arXiv: [2310.03294 \[cs.LG\]](https://arxiv.org/abs/2310.03294).
- [23] Shenggui Li et al. “Sequence Parallelism: Long Sequence Training from System Perspective”. In: *Annual Meeting of the Association for Computational Linguistics*. 2021. URL: <https://api.semanticscholar.org/CorpusID:246017095>.

- [24] Shigang Li and Torsten Hoefer. “Chimera: efficiently training large-scale neural networks with bidirectional pipelines”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2021, pp. 1–14.
- [25] Shigang Li et al. “Taming unbalanced training workloads in deep learning with partial collective operations”. In: *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2020, pp. 45–61.
- [26] Hao Liu, Matei Zaharia, and Pieter Abbeel. *Ring Attention with Blockwise Transformers for Near-Infinite Context*. 2023. arXiv: 2310.01889 [cs.CL].
- [27] Ziming Liu et al. “Hanayo: Harnessing Wave-like Pipeline Parallelism for Enhanced Large Model Training Efficiency”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’23. Denver, CO, USA, Association for Computing Machinery, 2023. ISBN: 9798400701092. DOI: 10.1145/3581784.3607073. URL: <https://doi.org/10.1145/3581784.3607073>.
- [28] Paulius Micikevicius et al. *Mixed Precision Training*. 2018. arXiv: 1710.03740 [cs.AI].
- [29] Maxim Milakov and Natalia Gimelshein. *Online normalizer calculation for softmax*. 2018. arXiv: 1805.02867 [cs.PF].
- [30] Deepak Narayanan et al. *Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM*. 2021. DOI: 10.48550/ARXIV.2104.04473. URL: <https://arxiv.org/abs/2104.04473>.
- [31] NVIDIA. *NVIDIA Collective Communications Library*. 2020. URL: <https://developer.nvidia.com/nccl>.
- [32] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. DOI: 10.48550/ARXIV.1912.01703. URL: <https://arxiv.org/abs/1912.01703>.
- [33] William Peebles and Saining Xie. *Scalable Diffusion Models with Transformers*. 2023. arXiv: 2212.09748 [cs.CV].
- [34] William Peebles and Saining Xie. *Scalable Diffusion Models with Transformers*. 2023. arXiv: 2212.09748 [cs.CV]. URL: <https://arxiv.org/abs/2212.09748>.
- [35] Markus N. Rabe and Charles Staats. *Self-attention Does Not Need  $O(n^2)$  Memory*. 2022. arXiv: 2112.05682 [cs.LG].
- [36] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: (2018). URL: <https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf>.
- [37] Samyam Rajbhandari et al. *ZeRO: Memory Optimizations Toward Training Trillion Parameter Models*. 2020. arXiv: 1910.02054 [cs.LG].
- [38] Jay Shah et al. *FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision*. 2024. arXiv: 2407.08608 [cs.LG]. URL: <https://arxiv.org/abs/2407.08608>.
- [39] Noam Shazeer. *Fast Transformer Decoding: One Write-Head is All You Need*. 2019. arXiv: 1911.02150 [cs.NE].
- [40] Mohammad Shoeybi et al. “Megatron-LM: Training multi-billion parameter language models using model parallelism”. In: *arXiv preprint arXiv:1909.08053* (2019).
- [41] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].
- [42] Boxiang Wang et al. “Tesseract: Parallelize the Tensor Parallelism Efficiently”. In: *Proceedings of the 51st International Conference on Parallel Processing*. 2022, pp. 1–11.
- [43] Qifan Xu et al. “An efficient 2d method for training super-large deep learning models”. In: *arXiv preprint arXiv:2104.05343* (2021).
- [44] Yang You et al. “Imagenet training in minutes”. In: *Proceedings of the 47th International Conference on Parallel Processing*. 2018, pp. 1–10.
- [45] Zilin Zhu et al. *Ring Flash Attention*. 2024. github: <https://github.com/zhuzilin/ring-flash-attention>.