LoongTrain: Efficient Training of Long-Sequence LLMs with Head-Context Parallelism

Diandian Gu
School of Computer Science
Peking University

Peng Sun
Sensetime Research &
Shanghai AI Laboratory

Qinghao Hu
S-Lab, NTU &
Shanghai AI Laboratory

Ting Huang
Sensetime Research

Xun Chen
Sensetime Research

Yingtong Xiong
Shanghai AI Laboratory

Guoteng Wang
Shanghai AI Laboratory

Qiaoling Chen
S-Lab, NTU

Shangchun Zhao
Tencent

Jiarui Fang
Tencent

Yonggang Wen
Nanyang Technological University

Tianwei Zhang
Nanyang Technological University

Xin Jin
School of Computer Science
Peking University

Xuanzhe Liu
School of Computer Science
Peking University

Abstract

Efficiently training LLMs with long sequences is important yet challenged by the massive computation and memory requirements. Sequence parallelism has been proposed to tackle these problems, but existing methods suffer from scalability or efficiency issues. We propose LoongTrain, a novel system to efficiently train LLMs with long sequences at scale. The core of LoongTrain is the 2D-Attention mechanism, which combines both *head-parallel* and *context-parallel* techniques to break the scalability constraints while maintaining efficiency. We introduce Double-Ring-Attention and analyze the performance of device placement strategies to further speed up training. We implement LoongTrain with the hybrid ZeRO and *Selective Checkpoint++* techniques. Experiment results show that LoongTrain outperforms state-of-the-art baselines, i.e., DeepSpeed-Ulysses and Megatron Context Parallelism, in both end-to-end training speed and scalability, and improves Model FLOPs Utilization (MFU) by up to 2.88×.

Keywords: Distributed Training, Sequence Parallelism, Distributed Attention

1 Introduction

With the emergence of Large Language Models (LLM) in recent years, researchers have investigated and proposed many advanced training methodologies in a distributed way, such as data parallelism (DP) [23, 25, 26, 36], tensor parallelism (TP) [15], pipeline parallelism (PP) [4, 20], PyTorch FSDP [52], and automatic parallelization frameworks [53]. Recently, LLMs with long sequences have driven the development of novel applications that are essential in our daily lives, including generative AI [33] and long-context understanding [5, 16, 54]. With the increased popularity of ChatGPT, long dialogue processing tasks have become more important for chatbot applications than ever [45]. In addition to these scenarios for language processing, Transformer-based giant models also achieve impressive performance in computer vision [3, 49, 50] and AI for science [6, 30], where inputs with long sequences are critical for complex tasks such as video stream processing [41] and protein property prediction [9].

Training LLMs with long sequences requires massive memory resources and computation. To tackle these challenges, sequence parallelism (SP) has been proposed [21, 24, 29, 34], which can be basically divided into two categories: head parallelism (HP) [21] and context parallelism (CP) [29, 34]. In Attention blocks, HP methods keep the whole sequence and compute attention for different heads in parallel, while CP methods split the QKV (Query, Key, and Value) tensors into chunks along the sequence dimension. However, both face limitations when applied to extremely-long-sequence LLMs at a large scale. **First, HP meets the scalability issue.** In HP, the degree of SP inherently cannot exceed the number of attention heads [21]. Therefore, there is an upper bound for the degree that HP can scale out. **Second, CP meets the communication inefficiency issue.** CP [29, 34] employs a peer-to-peer (P2P) communication primitive. However, P2P encounters issues of low intra-node bandwidth utilization and low inter-node network resource utilization. This bottleneck makes it challenging to overlap communication with computation when scaling out the context-parallel dimension. For example, our experiments show that Ring-Attention can spend 1.8× time on communication than on computation when running Grouped Query Attention (GQA) on 64 GPUs with a sequence length of 128K (Figure 5(d)).

To bridge these gaps, we propose LoongTrain, an effective training framework for long-sequence LLMs on large-scale

S	Sequence Length (Tokens)	d_{sp}	Sequence Parallel Size
H	Number of Attention Heads	d_{dp}	Data Parallel Size
H_{kv}	Number of KV Heads	d_{hp}	Head Parallel Size
D	Hidden Dimension Size	d_{cp}	Context Parallel Size
B	Global-Batch Size (Tokens)	w	Inner Ring Size

Table 1. Notations used in this paper.

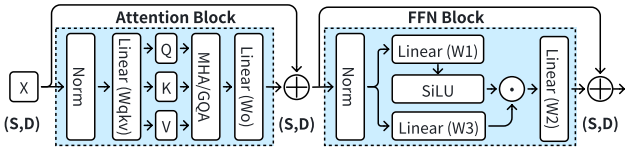


Figure 1. A typical Transformer layer contains an Attention block and a Feed-Forward Network (FFN) block.

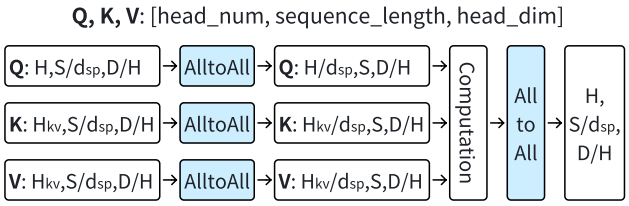


Figure 2. Ulyssess-Attention performs head-parallel computation across GPUs with two steps of `AlltoAll`.

GPU clusters. Our key idea is to address the scalability constraints of HP while mitigating the inefficiencies of CP by introducing a novel *2D-Attention* mechanism. This mechanism parallelizes attention across both HP and CP dimensions. Specifically, it distributes the QKV tensors across GPUs based on the head dimension and partitions these tensors into chunks within the CP dimension. By doing so, LoongTrain enhances scalability through the integration of CP and reduces the number of P2P steps by confining the CP dimension size. In addition, this design provides more opportunities for computation-communication overlap.

To further improve the communication efficiency of Attention blocks in certain circumstances, we introduce *Double-Ring-Attention*, which utilizes all of the inter-node NICs efficiently for higher peer-to-peer communication bandwidth. We also analyze how different *placement strategies* can boost the communication efficiency in different 2D-Attention configurations. Finally, we implement advanced techniques such as applying ZeRO across both DP and PP dimensions and a whitelist-based gradient checkpointing mechanism *Selective Checkpoint++* to further improve the end-to-end LLM training performance. Evaluation results on training LLMs with up to 1M sequences show that LoongTrain can bring up to 2.88× performance improvement compared to existing state-of-the-art solutions.

LoongTrain has been deployed to train multiple long-sequence LLMs within our organization. The system is implemented within our internal training framework, which can be accessed at https://github.com/InternLM/InternEvo.

2 Background

2.1 LLM Architecture with MHA/GQA

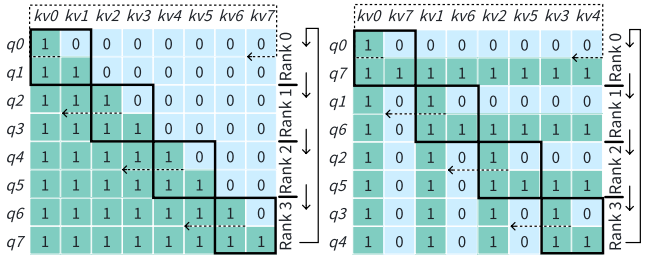
LLMs like GPT [8] and LLaMA [43] utilize the Transformer architecture [46], which consists of multiple layers. As shown in Figure 1, each layer includes an Attention block and a Feed-Forward Network (FFN) block. Within the Attention block, a linear module projects the input tensor into three tensors: Query (Q), Key (K), and Value (V), which are used for attention computation. Then, each layer includes an FFN, which operates independently on each position within the sequence. $\text{FFN}(x) = W_2(\text{SiLU}(W_1(x)) \times W_3(x))$, where W_1, W_2, W_3 are all linear modules.

Multi-Head Attention (MHA) [47] splits Q, K, and V into H heads. Suppose the original Q, K, and V tensors have the shape (S, D). They will be reshaped to $(H, S, D/H)$. MHA performs attention computation for each head independently and then combines the outputs of all heads. Grouped Query Attention (GQA) [2] divides the H query heads into G groups, with each group sharing a single set of KV heads. In this case, the transformed K and V tensors have $H_{kv} = H/G$, resulting in a shape of $(H_{kv}, S, D/H)$. For example, LLaMA3-8B [1] employs GQA with $H_{kv} = 8$ and $H = 32$.

2.2 Distributed LLM Training

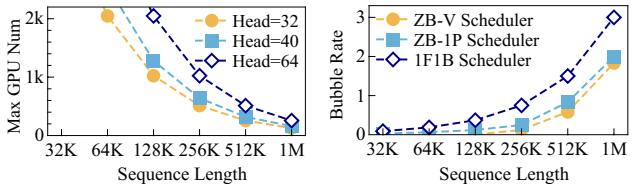
Hybrid parallelism [31] and Zero Redundancy Optimizer (ZeRO) [39] are commonly employed to train LLMs at scale. Specifically, data parallelism (DP) divides input data into chunks, distributing them across multiple GPUs to parallelize training. Tensor parallelism (TP) distributes model parameters across GPUs along specific dimensions, enabling parallel computation of the model layers [32]. Pipeline parallelism (PP) splits layers of a model into multiple stages, distributing them across GPUs [18, 20]. Each pipeline stage depends on the outputs of previous stages, leading to computation stalls known as pipeline bubbles. Advanced pipeline schedulers, such as 1F1B [18] and ZeRO-Bubble [37], have been proposed to reduce the bubble ratio. ZeRO [39] addresses redundant memory usage across DP ranks. ZeRO-1 partitions optimizer states across GPUs, ensuring each GPU stores only a fraction of the optimizer state. ZeRO-2 extends this by also sharding gradients, and ZeRO-3 further distributes model parameters.

To support long-sequence training, sequence parallelism (SP) has emerged as an effective technique to mitigate activation memory footprints [21, 22, 24]. In SP, the input and output tensors of each Transformer layer are partitioned into d_{sp} chunks along the sequence dimension. Megatron-LM integrates SP with TP across different modules [22]. Specifically, TP is utilized to parallelize the linear modules, while



(a) Without Load Balance **(b)** With Load-Balance

Figure 3. Ring-Attention performs context-parallel computation, and organizes communication in a ring fashion. 1 or 0 represents that whether there is computation between QKV.



(a) Maximum GPU Scalability **(b)** Pipeline Bubble Rate

Figure 4. Limited scalability of Ulysses-Attention constrained by a global batch size of 4M tokens. (a) Maximum GPU scalability without Pipeline Parallelism. (b) Pipeline bubble rate, using $d_{dp} = 4, d_{sp} = 64, d_{pp} = 4$ on 1024 GPUs.

SP is applied to normalization and dropout modules. To ensure consistency in computational results, Megatron-LM incorporates necessary `AllGather` and `ReduceScatter` operations to transfer activations during training. However, as the sequence length increases, the communication overhead associated with transferring activations also grows, leading to significant communication challenges [19, 21].

To address this problem in the integration of SP and TP, recent approaches implement SP across all linear modules and utilize ZeRO-3 to reduce memory footprints. This eliminates the need for collective communications on activations. They perform `AllGather` to collect the parameters of linear modules before computation, which do not increase with the sequence length. Following this strategy, two methods have been introduced to facilitate distributed attention computation: Ulysses-Attention [21] and Ring-Attention [24, 29], as described below.

2.3 Distributed Attention

Ulysses-Attention [21] performs head-parallel computation across GPUs ($d_{hp} = d_{sp}$), as depicted in Figure 2. Given the QKV tensors, which are split along the sequence dimension, Ulysses-Attention first performs `AlltoAll` to ensure that each GPU receives the complete sequence of QKV for H/d_{sp} heads. Each GPU then computes the attention for different heads in parallel. Finally, another `AlltoAll` operation gathers the results across the head dimension while re-partitioning along the sequence dimension.

Ring-Attention [24, 29] leverages blockwise attention [14, 27, 38] and performs context-parallel computation ($d_{cp} = d_{sp}$), as shown in Figure 3. This method partitions QKV tensors into chunks along the sequence dimension, with each GPU initially assigned one chunk. For each query chunk, its corresponding attention output is computed by iterating over all KV chunks. Communication is organized in a ring fashion, where each GPU simultaneously sends and receives KV chunks, allowing communication to be overlapped with computation. FlashAttention [14] can still be used to maintain the IO-aware benefits of memory-efficient computation.

However, the standard Ring-Attention approach is not load-balanced when applying a causal attention mask, since only the lower triangular portion of the matrix needs to be computed. To address this issue, several methods have been proposed, such as DistFlashAttn [24] and Striped-Attention [7]. As shown in Figure 3(b), Megatron-LM reorders the input sequence tokens along the sequence dimension to achieve load balance in its implementation. In this paper, Ring-Attention is assumed to be load-balanced by default.

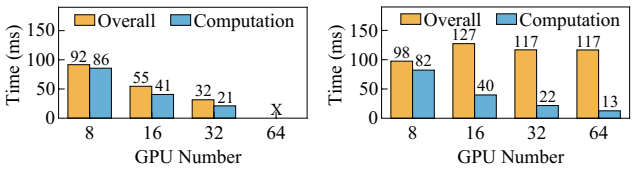
3 Motivation & Observation

Given the long computation time of LLM training, especially with long sequences, it is essential to scale long-sequence model training to large-scale clusters. However, current SP approaches face two significant challenges: limited scalability and high communication overhead.

3.1 Limited Scalability of Ulysses-Attention

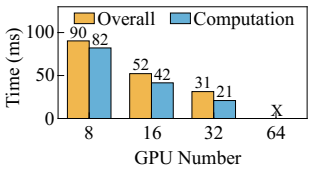
Ulysses-Attention cannot scale long-sequence training to large-scale clusters due to the limitations in the maximum degrees of SP, DP, and PP. First, SP is sensitive to the number of attention heads. When using MHA, the SP degree cannot exceed the number of attention heads; while in the case of GQA, the SP degree is limited by the number of key/value heads. For instance, LLaMA3-8B uses GQA with 8 key/value heads, meaning that the maximum SP degree is 8 when using Ulysses-Attention. Even if we repeat key/value heads, as detailed in Section 4.1, the maximum SP degree remains 32.

It is impractical to rely on increasing the degree of DP to scale out the training process due to the constraint of the global batch size. For instance, when training a Transformer model with 32 attention heads and employing a global batch size of 4M tokens—as exemplified in the world model training [28]—and a sequence length of 1M tokens, the maximum attainable degree of DP is 4. Under these conditions, the training process can only be scaled up to 128 GPUs when utilizing Ulysses-Attention. The maximum number of GPUs that Ulysses-Attention could use within the constraint of a 4M global batch size is illustrated in Figure 4 (a).

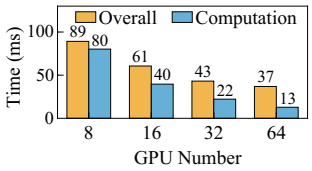


(a) Ulyssess-Attention (MHA)

(b) Ring-Attention (MHA)



(c) Ulyssess-Attention (GQA)



(d) Ring-Attention (GQA)

Figure 5. Forward time evaluation of Ulysses-Attention and Ring-Attention on 8 physical nodes, each equipped with 8 NVIDIA Ampere GPUs connected by NVLINK. Each node has four 200 HDR NICs. In the test, we set $H = 32$, $D = 4096$, and $S = 128K$ for MHA, and $H_{kv} = 8$ for GQA.

While we can scale out long-sequence training to more GPUs by increasing the degree of PP, it can lead to a high bubble rate. Due to the global batch size constraint, we have a limited number of micro-batches, which introduce a significant bubble rate. As shown in Figure 4(b), the bubble rate reaches 2 even under zero-bubble mechanisms, such as the ZB-V and ZB-1P schedulers [37]. This level of inefficiency is unacceptable for effective LLM training.

3.2 Inefficient Performance of Ring-Attention

While Ring-Attention demonstrates the potential to scale SP to large degrees, its performance is hindered by significant communication overheads. We evaluated the performance of Ring-Attention and Ulysses-Attention with a sequence length of 128K on a testbed comprising 64 GPUs[1]. As illustrated in Figure 5, Ulysses-Attention and Ring-Attention exhibit similar computation time, which decreases nearly linearly with the increased number of GPUs. However, as the degree of SP increases, Ring-Attention encounters bottlenecks due to the P2P communication required for transferring KV chunks over the network. Specifically, with MHA, the overall execution time for Ring-Attention does not improve when scaling from 32 GPUs to 64 GPUs. Although GQA reduces the communication volume by 4×, Ring-Attention still takes 1.8× more time on communication than on computation when using 64 GPUs.

[1]To scale training with 1M sequence length to 2048 GPUs, constrained by the global batch size of 4M tokens, d_{sp} would need to be scaled to 512. In this scenario, each query/key/value chunk contains 2K tokens, analogous to scaling the training with 128K sequence length on 64 GPUs.

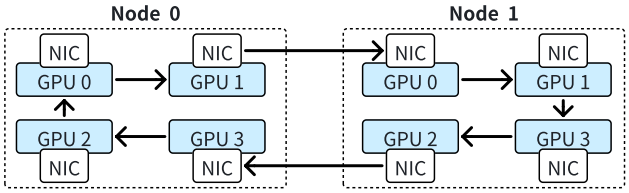


Figure 6. Ring-Attention uses one NIC for sending key/value chunks and another NIC for receiving key/value chunks.

The performance inefficiency of Ring-Attention primarily stems from three factors. First, due to the small communication size, the intra-node communication via NVLINK is more sensitive to the communication latency rather than the bandwidth. When running GQA with a sequence length of 128K on 8 GPUs, the communication volume is 64MB per step. This size does not fully utilize the high bandwidth of NVLINK, resulting in high communication latency that cannot be overlapped with computation. Second, when scaling Ring-Attention, the computation time per step decreases quadratically, whereas the communication volume per step only decreases linearly. This scaling exacerbates the imbalance between computation and communication, making communication the performance bottleneck. Third, Ring-Attention does not fully utilize network resources due to its ring-based communication design. Despite the widespread use of multi-rail networks in GPU clusters [35, 48], Ring-Attention utilizes one NIC for sending KV chunks and another NIC for receiving KV chunks, as shown in Figure 6. So in a single step, all other ranks must wait for the slowest rank using inter-node P2P communication. Thus, it is difficult to overlap communication with computation when scaling Ring-Attention to a large scale.

4 Distributed 2D-Attention

We introduce LoongTrain to address the scalability and efficiency challenges in training long-sequence LLMs. In particular, we propose 2D-Attention, which integrates head-parallel and context-parallel attention through a hybrid strategy, leveraging the benefits of both methods. This approach naturally overcomes the scalability limitations of head-parallel attention by incorporating context-parallel attention. To further reduce the communication overhead in Attention blocks, we design a Double-Ring-Attention mechanism and disclose the influence of device placement. Additionally, we briefly analyze the performance of 2D-Attention.

4.1 2D-Attention Overview

In LoongTrain, attention is parallelized across two dimensions: head parallelism (HP) and context parallelism (CP), which is referred to as *2D-Attention*. It organizes d_{sp} GPUs into a $d_{hp} \times d_{cp}$ grid, forming d_{hp} CP process groups of size

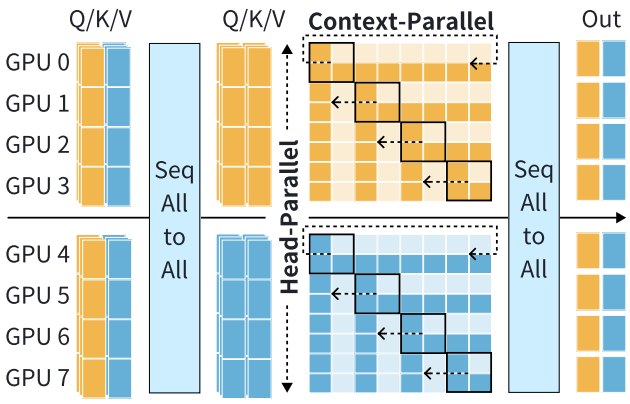


Figure 7. 2D-Attention design. Different colors represent different attention heads. In this example, $d_{cp} = 4$, $d_{hp} = 2$.

Algorithm 1 2D-Attention Mechanism (Forward Phase)

1: **Input:** Q, K, V, d_{hp}, d_{cp}
2: KV Replication: $\hat{K}, \hat{V} \leftarrow$ **Replica**(K, V)
3: Distribute QKV: $Q', K', V' \leftarrow$ **SeqAlltoAll**$(Q, \hat{K}, \hat{V})$
4: **for all** CP process groups **do**
5: $\quad O' \leftarrow$ **DoubleRingAttention**(Q', K', V', d_{cp})
6: Gather output: $O \leftarrow$ **SeqAlltoAll**(O')
7: **Output:** Attention Output of shape $(H, S/d_{sp}, D/H)$

d_{cp} and d_{cp} HP process groups of size d_{hp}. Thus, we have

$$d_{sp} = d_{hp} \times d_{cp}.$$

Algorithm 1 and Figure 7 illustrate the forward pass of 2D-Attention. In Figure 7's configuration, each CP process group contains four GPUs. The input tensors, Q (queries), K (keys), and V (values), are divided along the sequence dimension, with each segment shaped as $(H, S/d_{sp}, D/H)$. 2D-Attention handles head parallelism across CP groups, while context parallelism is executed within each CP group.

The computation of MHA in 2D-Attention involves three steps. ❶ The `SeqAlltoAll` communication operation distributes the QKV tensors based on the head dimension across d_{hp} GPUs and re-partitions them along the sequence dimension across d_{cp} GPUs, transforming the shape of QKV to $(H/d_{hp}, S/d_{cp}, D/H)$. This step ensures that each CP group receives the entire sequence of QKV with H/d_{hp} attention heads, as illustrated in Figure 7. ❷ Each CP group independently performs Double-Ring-Attention, as detailed in Section 4.3, resulting in an output tensor of shape $(H/d_{hp}, S/d_{cp}, D/H)$. During this stage, each GPU computes attention using the local QKV and exchanges partitioned KV chunks via P2P communication, transferring $2\times(H/d_{hp})\times(S/d_{cp})\times(D/H) = 2SD/d_{sp}$ elements through NVLINK or network. ❸ Finally, another `SeqAlltoAll` consolidates the attention outputs across the head dimension and re-partitions the sequence dimension, transforming the output tensor to $(H, S/d_{sp}, D/H)$.

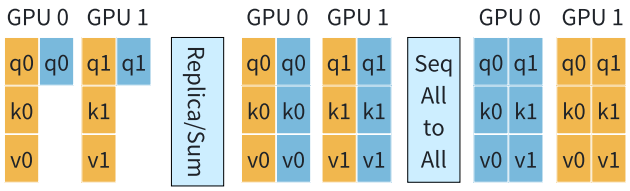


Figure 8. When $H_{kv} < d_{hp}$, 2D-Attention replicates KV tensors before `SeqAlltoAll` during forward pass, and aggregates these replicated KV tensors' gradients during backward pass. Different colors represent different attention heads.

In the backward pass, a `SeqAlltoAll` transforms the gradients of the attention output from shape $(H, S/d_{sp}, D/H)$ to $(H/d_{hp}, S/d_{cp}, D/H)$. Subsequently, each CP process group engages in context-parallel computations for the gradients by iteratively sending and receiving the partitioned KV chunks and their gradients. Finally, another `SeqAlltoAll` communication operation is employed to transform the gradients of QKV back to $(H, S/d_{sp}, D/H)$.

4.2 KV Replication for GQA

In MHA computation, d_{hp} can be set to up to H. However, when directly computing GQA, d_{hp} is constrained by the number of KV heads H_{kv}. Since $H_{kv} < H$, this constraint limits the search space for the two-dimensional parallel strategy in 2D-Attention, potentially hindering optimal performance.

2D-Attention uses KV replication to address the constraint of limited KV heads in GQA (Figure 8). In the forward pass, the input KV tensors are shaped as $(H_{kv}, S/d_{sp}, D/H)$. To align the number of KV heads with the head-parallel size, 2D-Attention replicates KV tensors, resulting in the shape of $(\hat{H}_{kv}, S/d_{sp}, D/H)$, where $d_{hp} \leq \hat{H}_{kv} \leq H$. A `SeqAlltoAll` operation transforms KV to $(\hat{H}_{kv}/d_{hp}, S/d_{cp}, D/H)$. KV replication can potentially increase network traffic at this stage. We will analyze this impact on communication in Section 4.5.

4.3 Double-Ring-Attention

2D-Attention may incur high communication overhead if we directly use Ring-Attention for CP computation if the CP groups are inter-node. As discussed in Section 3.2, Ring-Attention does not fully utilize the network resources because of its ring-based communication design.

To fully utilize available NICs for inter-node communication, we propose Double-Ring-Attention, which partitions the d_{cp} GPUs into multiple inner rings. As illustrated in Figure 9 and Alogrithm 2, GPUs within each CP group form several inner rings, while the inner rings collectively form an outer ring. Assuming each inner ring consists of w GPUs, a CP process group would thus have d_{cp}/w concurrent inner rings. Let $W_{i,j}$ denote the j-th GPU in the i-th inner ring. ❶ Initially, each inner ring performs conventional Ring-Attention, which involves w micro-steps. In each micro-step,

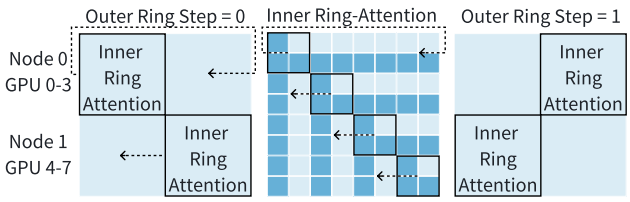


Figure 9. An illustration of Double-Ring-Attention. In this example, $d_{cp} = 8$, inner ring size is 4 and outer ring size is 2.

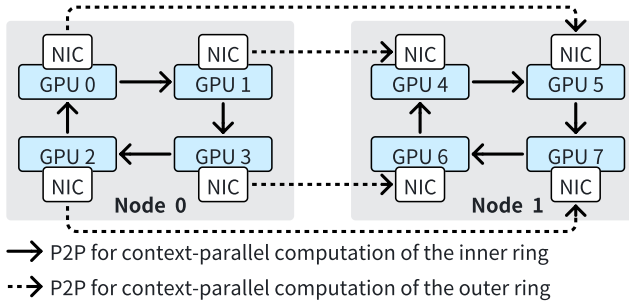


Figure 10. Communication in Double-Ring-Attention. In this example, GPUs in the same node create an inner ring with intra-node P2P communications. An outer ring requires inter-node P2P communications, utilizing all available NICs.

a GPU performs attention computation using local QKV, while simultaneously sending and receiving KV chunks necessary for the subsequent micro-step. ❷ Once the computations within all inner rings are complete, the outer ring advances to the next step and initiates a new round of Ring-Attention for each inner ring. There are d_{cp}/w outer ring steps in total. In the new outer ring step, GPUs within each inner ring use new KV chunks as the initial value, fetched from GPUs of the neighboring outer ring. This P2P communication can be overlapped with computation: $W_{i,j}$ sends its initial KV chunk to $W_{i+1,j}$ and concurrently receives a KV chunk from $W_{i-1,j}$ while computing the current inner ring.

Double-Ring-Attention offers superior communication efficiency compared to the original Ring-Attention. It fully utilizes available network resources to transfer KV chunks across nodes and overlaps these communication processes with computational tasks. For example, in the configuration of Figure 10, 8 GPUs are arranged into two inner rings, each containing 4 GPUs. During computation within an inner ring, GPUs 0-3 employ distinct NICs to send KV chunks to GPUs 4-7.Additionally, P2P within the inner rings can be entirely initiated within a single node, thereby avoiding the need to wait for inter-node P2P communication at every micro-step. We will analyze the communication cost of Double-Ring-Attention and discuss the choice of w in Section 4.5.

Algorithm 2 Double-Ring Attention Mechanism

Input: Q, K, V, d_{cp}, w
for Outer_Ring_Step = 0 to $d_{cp}/w - 1$ **do**
  P2P.async_send(KV, next_outer_rank)
  $\hat{K}, \hat{V} \leftarrow$ P2P.async_recv(previous_outer_rank)
  for Inner_Ring_Step = 0 to $w - 1$ **do**
    P2P.async_send(KV, next_inner_rank)
    $K', V' \leftarrow$ P2P.async_recv(previous_inner_rank)
    $blocl\_out, block\_lse \leftarrow$ Attention(Q, K, V)
    $out, lse \leftarrow$ Update($out, lse, blocl\_out, block\_lse$)
    P2P.synchronize(inner_ring_p2p)
    $K, V \leftarrow K', V'$ ▷ *update KV for next inner ring*
  P2P.synchronize(outer_ring_p2p)
  $K, V \leftarrow \hat{K}, \hat{V}$ ▷ *update KV for next outer ring*
Output: out

4.4 Head-First & Context-First Device Placement

Given d_{hp} and d_{cp}, there are two device allocation strategies: head-first placement and context-first placement. The selection of an appropriate placement strategy is critical due to the disparity between inter-node and intra-node bandwidths in GPU clusters. For instance, DGX-A100 nodes provide an intra-node bidirectional bandwidth of 600 GB/s per GPU through NVLINK, while the inter-node bidirectional bandwidth is only 400 GB/s per node. The choice of device placement directly influences the distribution of inter-node and intra-node communication for two types of operations in 2D-Attention: SeqAlltoAll and P2P. Figure 11 shows examples of head-first and context-first placement.

In head-first placement, GPUs of the same HP group are given high priority for colocation on the same node. As illustrated in Figure 11(a), GPUs 0 and 1 are assigned to the same HP group but to different CP groups. This configuration can efficiently leverage NVLINK for SeqAlltoAll, as it only requires a standard NCCL AlltoAll within the HP process group. However, head-first placement leads to higher inter-node traffic during Double-Ring-Attention, because GPUs within the same CP group are more likely to be distributed across different nodes, increasing the inter-node traffic.

In context-first placement, GPUs of the same CP group are prioritized for colocation on the same node. As shown in Figure 11(b), GPUs 0-3 are allocated to the same CP group. Thus, in this example, Double-Ring-Attention generates only intra-node traffic, significantly reducing the communication latency per P2P operation. However, when $d_{cp} > 8$, P2P necessitates inter-node interconnections. Fortunately, the double-ring approach proposed in Section 4.3 leverages multiple NICs to maintain high efficiency. Maintaining the use of a standard NCCL AlltoAll within an HP group necessitates reordering the input QKV tensors across nodes, which increases network traffic for each Transformer layer. To mitigate this issue, we adopt the approach used in Megatron-LM,

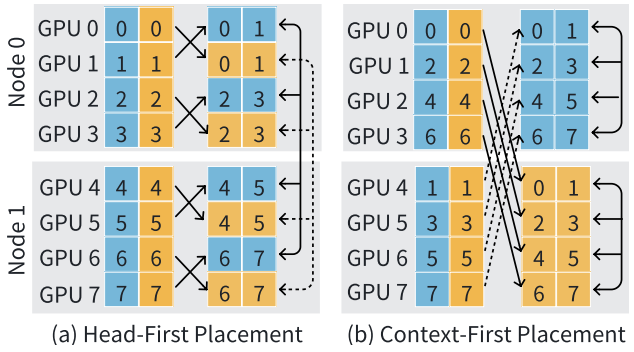


Figure 11. Context-first placement vs. head-first placement. Different colors represent different attention heads. In context-first placement, a post-processing function within the data loader is required to adjust input sequence placement at the start of each batch.

implementing a post-processing function within the data loader to adjust input tensor placement at the start of each batch. This obviates the need for on-the-fly data movement for QKV tensors. Even with this optimization, `SeqAlltoAll` still demands significant inter-node communication traffic.

4.5 Performance Analysis

4.5.1 Scalability Analysis. 2D-Attention enhances the scalability of long-sequence training by integrating head parallelism and context parallelism through a hybrid strategy. It overcomes the limitations of head parallelism by incorporating context-parallel attention, distributing computation across a grid of GPUs organized as $d_{hp} \times d_{cp}$. This allows sequence parallelism to scale to an unlimited number of GPUs. Additionally, in the case of GQA, 2D-Attention can scale d_{hp} to H using KV replication, ensuring flexible processing and a large search space for optimal performance.

4.5.2 Computation Analysis. Given a sequence (S, D), the computational complexity of attention is $O(S^2D)$. The computation time can be formulated as $T^{fwd}_{comp} = \alpha S^2 D$, where α represents the proportionality constant for the forward computation time. In 2D-Attention, the forward computation time for each micro-step within the inner ring is described as $\alpha \left(S/d_{cp}\right)^2 D/d_{hp}$. Since $d_{sp} = d_{hp} \times d_{cp}$, we have:

$$T^{fwd}_{comp} = \alpha S^2 D/(d_{cp} d_{sp}).$$

There are w micro-steps in an inner ring and d_{cp}/w outer ring steps. The total forward computation time can be expressed as: $d_{cp} \times T^{fwd}_{comp}$. For the backward pass, the computation time for each micro-step is described as:

$$T^{bwd}_{comp} = 3\alpha S^2 D/(d_{cp} d_{sp}).$$

This is because the backward computation kernel naturally requires additional computations, such as activation recomputing and gradient calculations as in FlashAttention [14].

4.5.3 P2P Communication Analysis. The shape of a KV chunk is defined by: $(\max(H_{kv}, d_{hp})/d_{hp}, S/d_{cp}, D/H)$, where $H_{kv} = H$ in MHA, and the KV tensors are replicated to match the head-parallelism size if $d_{hp} > H_{kv}$. The size of a KV chunk can be calculated as follows:

$$Size(kv) = \max(H_{kv}, d_{hp})/H \times 4SD/d_{sp},$$

where the factor of 4 accounts for two tensors with data type FP16. When using Double-Ring-Attention, given the inner ring size w, each GPU launches $(w-1)$ P2P communications for the inner ring and one P2P communication per outer ring step (except the last one) in the forward phase. The communication size for each P2P communication is equivalent to $Size(kv\_chunk)$. GPUs concurrently launch P2P communications for inner rings and outer rings. Each P2P communication time depends on the slowest rank, due to the ring communication fashion. The forward execution time per inner ring, considering the overlap between communication and computation can be formulated as follows:

$$T^{fwd}_{inner\_ring} = A \times (w-1) + B,$$

where A and B are defined as:

$$A = \max(T^{fwd}_{comp}, T^{fwd}_{P2P\_inner}), B = \max(T^{fwd}_{comp}, T^{fwd}_{P2P\_outer}).$$

The backward execution time per inner ring can be expressed with similar expressions.

The per P2P communication time remains unaffected by d_{cp} (assuming no KV replication), as $Size(kv\_chunk)$ remains constant regardless of d_{cp}. However, the computation time per micro-step decreases linearly when d_{cp} is increased. Thus, it becomes more challenging to effectively overlap computation and communication, and Ring-Attention exhibits poor performance in large clusters due to high communication overhead. 2D-Attention outperforms Ring-Attention since it provides more opportunities for computation-communication overlap by limiting d_{cp}.

Selection of Inner Ring Size. When selecting context-first placement, ranks of the same CP group are consolidated to as few nodes as possible. In this case, there are w concurrent P2P communications for the outer ring. To fully utilize network resources, w should match the number of NICs per node. When w is smaller than that of NICs, we cannot fully utilize all NICs for P2P. Conversely, when w is larger than that of NICs, GPUs may share the same NIC for P2P, leading to worse performance due to congestion.

GQA vs. MHA. During 2D-Attention of GQA, each P2P transfer involves $\hat{H}_{kv}/H \times 2SD/d_{sp}$ elements, where $\hat{H}_{kv}$ represents the number of KV heads after KV replication. Compared to MHA, GQA requires less communication when $\hat{H}_{kv} < H$. Specifically, when applying 2D-Attention for GQA, it results in less communication volume in the CP process group as long as $H_{kv} < d_{hp}$, because KV replication is not applied in this case. However, if $d_{hp} = H$, GQA and MHA will have the same communication volume due to KV replication.

4.5.4 SeqAlltoAll Communication Analysis.

The size of a Q chunk and output chunk can be calculated as follows:

$$Size(q) = Size(out) = 2SD/d_{sp}.$$

`SeqAlltoAll` performs NCCL `AlltoAll` on d_{hp} GPUs. The size of the data that each GPU sends out in both the forward and backward phases can be expressed as follows:

$$AlltoAll\_Volume = \sum_{i \in \{(q,k,v,out\}} Size(i) \times (d_{hp} - 1)/d_{hp}.$$

With a larger d_{hp}, $AlltoAll\_Volume$ increases, making the operation more substantial; if $d_{hp} = 1$, no `SeqAlltoAll` is required but $P2P\_Volume$ increases. With head-first placement, more `AlltoAll`-related traffic is carried by intra-node NVLINK, and vice versa for context-first placement.

Therefore, there is a trade-off between d_{cp} and d_{hp}, as well as between the head-first and context-first placement. LoongTrain's overall goal is to minimize the communication time that cannot be overlapped with computation. The problem can be formulated as:

$$\min T_{SeqAlltoAll} + (T^{fwd}_{inner\_ring} + T^{bwd}_{inner\_ring}) \times (d_{cp}/w).$$

In the formulation, $T_{SeqAlltoAll}$ represents the `SeqAlltoAll` communication time. There are d_{cp}/w inner rings to complete the execution of attention.

4.5.5 Memory Analysis.

When using 2D-Attention, each GPU should save its input QKV chunks (after `SeqAlltoAll`) as the activation. Thus, given a fixed sequence length, 2D-Attention can also reduce the activation memory usage by increasing d_{sp}. Similar to Ring-Attention, each GPU of LoongTrain maintains a buffer of $Size(kv)$ for inner ring `P2P` communication. However, LoongTrain requires another memory buffer of $Size(kv)$ for outer ring `P2P` communication. Experiment results in Section 6 show that this memory overhead is small and does not hinder scalability.

5 End-to-end System Implementation

We describe the end-to-end system implementation of LoongTrain for training LLMs on our internal framework with two techniques: Hybrid ZeRO and *selective checkpoint++*.

5.1 Hybrid ZeRO for Norm and Linear Modules

In LoongTrain, all modules except for attention (e.g., Linear, LayerNorm, etc.) utilize Zero [39]. ZeRO is originally designed to reduce redundant memory usage across DP ranks. When directly using ZeRO, for instance in Figure 12, it works for GPU-0 and GPU-2, as well as GPU-1 and GPU-3, which belong to the same DP group. GPU-0 and GPU-1 would each hold half of the parameters or optimizer states, but these values remain identical, leading to redundant memory usage.

LoongTrain addresses these redundancies by applying ZeRO not only across the DP dimension but also along the SP dimension. This hybrid approach shards model states across both dimensions, distributing the model states across more GPUs. As a result, only $1/(d_{dp} \times d_{sp})$ of the model states are kept in GPU memory, significantly reducing the redundant memory usage. Such approach is also used in existing frameworks like DeepSpeed-Ulysses [21]. However, the latency of collective communication operations demonstrates a positive correlation with the communication scale [42, 51]. Consequently, as $d_{dp} \times d_{sp}$ scales up to hundreds of GPUs, the communication overhead becomes significant. In LoongTrain, we adopt the approach of AMSP [11], which introduces three flexible sharding strategies: Full-Replica, Full-Sharding, and Partial-Sharding. These strategies enable the Norm and Linear modules to select an appropriate sharding number across $d_{dp} \times d_{sp}$ GPUs, effectively balancing the GPU memory usage and communication overhead.

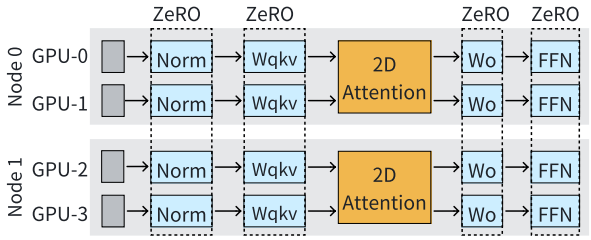


Figure 12. LoongTrain applies ZeRO to Norm and Linear modules across both DP and SP dimensions.

5.2 Selective Checkpoint++

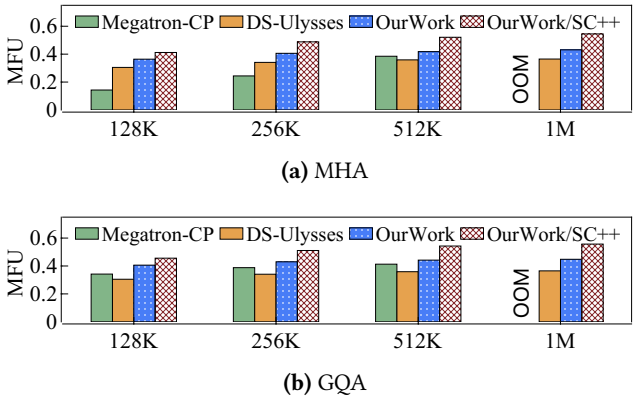
Long sequence training leads to significant memory costs, making gradient checkpointing a common practice. During forward propagation, the gradient checkpointing mechanism stores only the input tensors of the wrapped function by the `checkpoint` function. If the dropped activation values are needed during backward propagation, they are recomputed. Typically, when we wrap the `checkpoint` function around an entire Transformer layer, the total memory required for activations of a Transformer layer is $2SD/d_{sp}$ in FP16.

While saving the checkpoints of the entire model significantly reduces the memory footprint, it introduces additional computation overhead [14]. Given that the recomputation time for attention blocks is particularly long, a straightforward approach is to keep the activations of attention blocks and use checkpointing for the other parts of the model selectively with the provided APIs [22]. However, this solution is not memory-efficient. During backward propagation, each attention block requires extra memory to save the QKV tensors (size $6SD/d_{sp}$ in FP16) and `softmax_lse` (size $4SH/d_{sp}$ in FP32) [10]. To reduce memory usage, DistFlashAttn [24] places the attention module at the end of each Transformer layer. This strategy eliminates the need to recompute the attention module during the backward phase and only requires storing the output of the attention module.

	MHA (TGS)				MHA (MFU)				GQA (TGS)				GQA (MFU)			
System	128K	256K	512K	1M	128K	256K	512K	1M	128K	256K	512K	1M	128K	256K	512K	1M
DS-Ulysses	629.9	418.3	243.1	130.6	0.305	0.341	0.359	0.365	629.9	418.3	243.1	130.6	0.305	0.341	0.359	0.365
Megatron-CP	296.8	300.0	260.1	OOM	0.143	0.244	0.385	OOM	706.2	476.3	279.6	OOM	0.342	0.388	0.413	OOM
HP1/CP32	285.0	287.4	250.4	121.2	0.138	0.234	0.369	0.339	668.5	480.0	282.5	153.0	0.323	0.391	0.417	0.428
HP2/CP16	311.1	314.9	267.3	151.6	0.151	0.256	0.394	0.423	740.8	501.3	290.1	155.9	0.359	0.408	0.428	0.436
HP4/CP8	548.9	469.2	283.6	154.1	0.266	0.382	0.408	0.431	814.4	517.4	295.1	159.5	0.394	0.421	0.435	0.446
HP8/CP4	752.4	498.1	286.1	154.1	0.364	0.406	0.418	0.431	838.1	528.1	299.5	160.1	0.406	0.430	0.442	0.448
HP16/CP2	714.3	472.4	278.9	150.9	0.346	0.385	0.412	0.422	771.4	498.6	288.0	155.1	0.373	0.406	0.425	0.433
HP32/CP1	700.1	459.3	268.8	146.0	0.339	0.374	0.397	0.408	717.1	468.4	262.4	147.5	0.347	0.381	0.387	0.412

Table 2. Performance comparison of end-to-end training between LoongTrain, DS-Ulysses, and Megatron-CP. HPn/CPm denotes our proposed system LoongTrain (head-first placement) with head parallelism size n and context parallelism size m.

LoongTrain implements the *selective checkpoint++* mechanism without modifying the model structure. It adds attention modules to a *whitelist*. During the forward pass, when encountering a module in the *whitelist*, the modified `checkpoint` function saves its outputs. Specifically, for attention, it saves the attention output with the size of $2SD/d_{sp}$ and `softmax_lse` with the size of $4SH/d_{sp}$. During the backward pass, when encountering a module in the *whitelist*, the `checkpoint` function does not perform recomputation. Instead, it retrieves the stored outputs and continues the computation graph. This eliminates the need to recompute attention during the backward pass, requiring an additional $(2SD + 4SH)/d_{sp}$ memory size per Transformer layer. Additionally, *selective checkpoint++* is compatible with other offload techniques [40], which involve offloading attention outputs to memory or NVMe storage.



(a) MHA

(b) GQA

Figure 13. Performance comparison between Megatron-CP, DeepSpeed-Ulysses and our proposed LoongTrain on 32 GPUs with the sequence length from 128K to 1M.

6 Performance Evaluation

6.1 Experiment Setup

Testbed. We conduct performance evaluation on a cluster with 8 GPU servers unless specified otherwise. Each server is equipped with 8 NVIDIA Ampere GPUs, 128 CPU cores, and 80GB memory per GPU. Within each node, GPUs are interconnected via NVLINK. Inter-node communication is facilitated by 4 NVIDIA Mellanox HDR (200Gb/s) InfiniBand NICs, without SHARP.

System Configurations. We evaluate the training performance of LoongTrain using the configuration of LLaMA2-7B [44], where $D = 4096$ and $H = 32$ for MHA, and $H_{kv} = 8$ for GQA. The input sequence length is scaled from 128K to 1M. In all experiments, activation checkpointing is enabled by default. We analyze the performance of LoongTrain with different parallelism settings and device placements.

Evaluation Metrics. We focus on key performance metrics, including Model FLOPs Utilization (MFU) [12] and Tokens per GPU per Second (TGS). We use the formula provided in Megatron-LM [31] for calculating FLOPs and MFU. Notably, the FLOPs for attention are halved in this work to account for the causal mask, which reduces the number of elements in attention that require computation by approximately half. This differs from the FLOPs and MFU calculations used in other works [10, 13, 14], but is essential since attention accounts for the majority of the workload in long sequence training. Without this adjustment, the MFU can exceed 1, misrepresenting the actual system performance.

Baselines. We compare the performance of LoongTrain against two long sequence training frameworks: DeepSpeed-Ulysses (DS-Ulysses) [21] and Megatron Context Parallelism (Megatron-CP) [34]. DS-Ulysses employs head-parallel attention, while Megatron-CP utilizes Ring-Attention with load balancing. All baseline systems are integrated with FlashAttention-V2 [13]. The versions used are as follows: 1) DS-Ulysses: DeepSpeed V0.14.0; 2) Megatron-CP: Nemo v2.0.0rc0, NemoLauncher v24.05, Megatron-Core v0.7.0, TransformerEngine v1.6, Apex commit ID 810ffa.

6.2 Comparison with DS-Ulysses & Megatron-CP

Theoretically, 2D-Attention when $d_{cp} = 1$ is equivalent to DS-Ulysses and 2D-Attention when $d_{hp} = 1$ is equivalent to

			128K				256K				512K				1M			
			With SC++		W/O SC++		With SC++		W/O SC++		With SC++		W/O SC++		With SC++		W/O SC++	
	d_{cp}	d_{hp}	HF	CF	HF	CF	HF	CF	HF	CF	HF	CF	HF	CF	HF	CF	HF	CF
MHA	64	1	0.092	0.092	0.070	0.070	0.159	0.159	0.122	0.122	0.290	0.290	0.221	0.221	0.452	0.452	0.357	0.357
	32	2	0.099	0.158	0.077	0.126	0.173	0.278	0.133	0.219	0.316	0.434	0.243	0.353	0.475	0.486	0.394	0.406
	16	4	0.176	0.245	0.141	0.205	0.314	0.378	0.248	0.317	0.470	0.472	0.384	0.388	0.520	0.509	0.418	0.413
	8	8	0.283	0.321	0.236	0.282	0.434	0.420	0.361	0.357	0.502	0.478	0.409	0.394	0.527	0.521	0.424	0.420
	4	16	0.328	0.327	0.289	0.283	0.436	0.423	0.369	0.359	0.487	0.476	0.399	0.394	0.519	0.520	0.418	0.412
	2	32	0.320	0.329	0.284	0.293	0.421	0.421	0.353	0.357	0.474	0.478	0.388	0.394	0.517	0.517	0.415	0.406
GQA	64	1	0.255	0.255	0.196	0.196	0.379	0.379	0.308	0.308	0.470	0.470	0.378	0.378	0.508	0.508	0.406	0.406
	32	2	0.283	0.317	0.233	0.269	0.419	0.429	0.345	0.354	0.492	0.485	0.398	0.392	0.521	0.516	0.418	0.416
	16	4	0.354	0.338	0.309	0.294	0.466	0.437	0.385	0.373	0.505	0.494	0.410	0.404	0.531	0.526	0.425	0.426
	8	8	0.377	0.354	0.327	0.310	0.480	0.452	0.392	0.380	0.516	0.502	0.419	0.412	0.543	0.536	0.435	0.432
	4	16	0.354	0.341	0.310	0.308	0.457	0.437	0.377	0.373	0.500	0.493	0.409	0.405	0.532	0.529	0.428	0.419
	2	32	0.323	0.333	0.285	0.295	0.424	0.422	0.349	0.360	0.476	0.481	0.389	0.394	0.518	0.518	0.415	0.406

Table 3. End-to-end training performance (MFU) of 7B-MHA and 7B-GQA on 64 GPUs with $d_{sp} = 64$. SC++ stands for Selective Checkpoint++, HF for head-first, and CF for context-first. The highest MFU value in each column is highlighted.

Megatron-CP. To validate that our LoongTrain implementation is consistent with this theoretical analysis, we measured the TGS and MFU when training 7B-MHA and 7B-GQA on 32 GPUs using LoongTrain, DS-Ulysses, and Megatron-CP, with different sequence lengths. The comparison was limited to 32 GPUs because DS-Ulysses supports only head-parallelism, which is constrained by the number of attention heads. To ensure a fair comparison, all systems applied ZeRO-1 on Norm and Linear modules across the 32 GPUs, and did not use *Selective Checkpoint++*. The results are shown in Table 2.

When $d_{cp} = 1$, LoongTrain outperforms DS-Ulysses due to its superior overlap capability between communication and computation during the backward phase for Norm and Linear modules. When $d_{hp} = 1$, LoongTrain demonstrates slightly lower performance than Megatron-CP in MHA, but exhibits higher performance in GQA. Our analysis indicates both systems perform similarly in attention computation. The main performance disparity arises from the divergent choices in computation and communication operators. Notably, when processing the sequence length of 1M, Megatron-CP encounters out-of-memory errors due to increased pre-allocated GPU memory requirements for parameters and gradients.

For sequence lengths of 128K and 256K, Megatron-CP exhibits poor performance in MHA, as the `P2P` communication cannot be effectively overlapped with computation. However, with the sequence lengths of 512K and 1M, both Megatron-CP and LoongTrain-HP1/CP32 show better performance than DS-Ulysses for MHA. Additionally, in GQA, the communication volume per micro-step is reduced by a factor of 4. Consequently, Megatron-CP and LoongTrain-HP1/CP32 consistently outperform DS-Ulysses across all evaluated sequence lengths for GQA.

Then, we compare the end-to-end performance of the complete LoongTrain and the baselines. All of the techniques such as hybrid ZeRO and Selective Checkpoint++ are used. As shown in Figure 13, LoongTrain delivers larger MFU. The configuration of $d_{hp} = 8$ and $d_{cp} = 4$ is more efficient in this experiment. Compared to DS-Ulysses, LoongTrain improves the training performance of MHA and GQA by up to 1.49× and 1.53×, respectively. Compared to Megatron-CP, LoongTrain enhances the performance of MHA and GQA by up to 2.88× and 1.33×, respectively.

6.3 Analysis of LoongTrain Performance

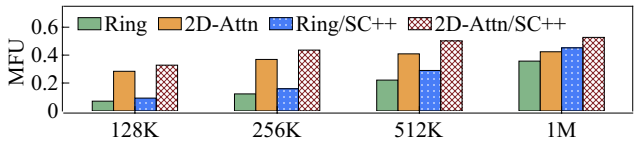
To analyze how much performance improvement can be brought by each design, we evaluated the performance of LoongTrain for training the 7B-MHA and 7B-GQA models on 64 GPUs with various sequence lengths and configurations. The evaluation results are presented in Table 3. We do not show the results for $d_{cp} = 1$ as d_{hp} cannot exceed the number of attention heads, which is 32. The end-to-end evaluation demonstrates that LoongTrain's designs (e.g., 2D-Attention) and implementation techniques (e.g., Selective Checkpoint++), significantly enhance the training performance across all cases. Figure 14 shows the end-to-end MFU results and the details are listed in Table 3.

When $d_{hp} = 1$, LoongTrain exhibits similarly poor performance as Ring-Attention for MHA: the MFU is less than 10% with the sequence length of 128K. When the sequence length increases to 1M, which entails a higher computational workload, the MFU is only 35.7% without Selective Checkpoint++. For GQA, Ring-Attention involves 4× less communication volume compared to MHA, leading to a higher MFU than MHA. Specifically in Ring-Attention, the MFU reaches 19.6% with the sequence length of 128K, and increases to 40.6% when the sequence length is 1M.

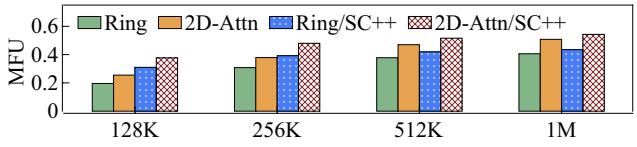
With 2D-Attention, LoongTrain significantly improves the training performance for MHA. Compared to Ring-Attention,

			MHA (Head-First)				MHA (Context-First)				GQA (Head-First)				GQA (Context-First)			
	d_{cp}	d_{hp}	128K	256K	512K	1M	128K	256K	512K	1M	128K	256K	512K	1M	128K	256K	512K	1M
Overall	64	1	296.4	597.8	1210	2897	296.4	597.8	1210	2897	86.0	225.1	713.5	2681	86.0	225.1	713.5	2681
	32	2	273.6	546.8	1106	2745	162.4	328.7	782.5	2663	75.4	198.5	679.5	2607	64.9	187.1	683.5	2589
	16	4	137.0	275.8	708.1	2595	87.4	213.8	691.5	2617	55.4	172.1	659.4	2559	59.9	179.1	668.3	2543
	8	8	72.2	187.9	658.3	2557	62.2	185.6	675.3	2539	52.1	166.2	644.1	2494	56.8	175.2	656.1	2495
	4	16	58.4	179.8	671.9	2575	60.1	182.6	680.6	2549	55.8	173.6	659.6	2530	57.3	177.2	661.7	2510
	2	32	60.8	186.0	684.9	2573	59.4	183.0	677.1	2553	60.8	185.8	683.9	2579	59.3	183.1	677.5	2555
SeqAlltoAll	64	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	32	2	2.23	3.20	5.49	10.00	7.19	13.27	25.10	49.26	1.89	2.51	3.92	6.58	4.92	8.65	16.29	31.59
	16	4	2.45	3.52	5.80	10.53	10.31	19.25	37.37	73.74	2.15	2.76	4.08	6.76	6.90	12.55	23.82	46.87
	8	8	3.00	4.15	6.27	11.22	12.05	22.26	42.82	83.30	2.64	3.24	4.43	7.31	8.13	14.60	27.51	53.35
	4	16	9.11	15.99	29.02	55.38	12.95	23.97	45.52	90.28	7.23	12.85	22.56	42.44	10.12	18.91	34.94	71.51
	2	32	13.42	23.43	42.73	81.47	14.56	25.41	48.25	100.0	13.40	23.35	42.85	81.76	14.31	25.75	48.43	106.8

Table 4. Average overall execution time (ms) and SeqAlltoAll time (ms) of a single 2D-Attention forward and backward operation on 64 GPUs with $d_{sp} = 64$. The lowest overall execution time in each column is highlighted.



(a) MHA



(b) GQA

Figure 14. MFU comparison on 64 GPUs with sequence lengths from 128K to 1M. Ring indicates $d_{hp} = 1$ in LoongTrain. 2D-Attn indicates the best-performing configuration.

2D-Attention enhances the MFU by 4.1×, 3.0×, 1.8×, and 1.2× for sequence lengths of 128K, 256K, 512K, and 1M, respectively. With Selective Checkpoint++, LoongTrain further boosts the training performance by 1.15×, 1.18×, 1.22×, and 1.24× for the same sequence lengths. Consequently, Figure 14(a) shows that LoongTrain's overall training performance is improved by 5.2×, 3.6×, 2.3×, and 1.5×, respectively. Additionally, we observe that to achieve higher training performance for MHA, LoongTrain tends to use a higher head parallelism size for sequence lengths of 128K and 256K. For sequence lengths of 512K and 1M, LoongTrain tends to use a balanced head and context parallelism size.

2D-Attention also works effectively for GQA. Compared to the performance of Ring-Attention, LoongTrain enhances the MFU for sequences of 128K, 256K, 512K, and 1M by 1.58×, 1.27×, 1.11×, and 1.07×, respectively. Incorporating Selective Checkpoint++, LoongTrain further elevates the training performance by 1.21×, 1.22×, 1.23×, and 1.25× for the same sequence lengths. Consequently, Figure 14(b) shows that the overall training performance is improved by 1.9×, 1.5×, 1.3×, and 1.3×, respectively. For GQA, a balanced head and context parallelism size is a more efficient configuration.

6.4 Analysis of 2D-Attention

We evaluated 2D-Attention by measuring the average overall execution time and `SeqAlltoAll` communication time for a single 2D-Attention forward operation under various configurations. The results are presented in Table 4.

Sequence Length Study. As discussed in Section 4.5, with a fixed sequence parallelism degree, a longer sequence length provides more opportunities for computation-communication overlap. When $d_{hp} = 1$ and the sequence length grows from 128K to 1M, the overall attention time for MHA only increases by 9.7×, from 296.4ms to 2897ms, despite the computational workload increasing by 64×. In this configuration, there are no `SeqAlltoAll` operations, indicating that the primary performance bottleneck lies in `P2P` operations. In the case of GQA, the overall attention time increases from 86.0ms to 2681ms. Across all sequence lengths, GQA demonstrates a shorter execution time compared to MHA due to the reduced communication volume.

MHA Study. The execution time of MHA can be reduced significantly under the most appropriate configuration from Table 4. Specifically, the execution time decreases from 296.4ms to 58.4ms when LoongTrain increases the head parallelism degree to 16 for 128K sequence length. When processing a sequence length of 1M, the overall execution time decreases from 2681ms to 2555ms when LoongTrain increases the head parallelism degree to 8. As discussed in Section 4.5, the communication volume per `P2P` operation remains unaffected by d_{hp}(as long as d_{sp} keeps the same), but the computation time per micro-step increases linearly with increased d_{hp}.

	MHA (CP=64, HP=1)				MHA (CP=16, HP=4)				GQA (CP=64, HP=1)				GQA (CP=16, HP=4)			
Inner Ring Size	128K	256K	512K	1M	128K	256K	512K	1M	128K	256K	512K	1M	128K	256K	512K	1M
1	295.9	597.7	1214	2913	86.3	213.8	697.9	2621	94.2	226.7	713.5	2668	60.7	180.6	673.3	2567
2	184.5	401.3	917.1	2823	72.6	205.7	710.7	2611	83.2	218.9	730.5	2650	60.8	182.6	671.2	2530
4	140.6	316.3	842.7	2754	69.1	199.4	704.4	2610	78.4	210.3	719.7	2669	60.3	182.0	675.2	2535
8	214.9	415.1	869.9	2815	77.4	198.7	705.3	2621	83.4	211.6	723.1	2674	61.0	183.1	677.4	2537

Table 5. Average execution time (ms) of a single 2D-Attention forward and backward operation (with Double-Ring-Attention and context-first device placement) on 64 GPUs with $d_{sp} = 64$. The lowest execution time in each column is highlighted.

Therefore, LoongTrain can more effectively overlap the P2P communication with computation by increasing d_{hp}, even though such a configuration introduces more SeqAlltoAll communication time.

GQA Study. GQA introduces less communication volume and is less sensitive to d_{cp} compared to MHA. For instance, processing a 128K sequence with $d_{cp} = 64$ results in an execution time of 86.0ms per GQA operation, which is 3.4× shorter than that of MHA. LoongTrain further reduces the GQA execution time by increasing d_{hp}, thereby enhancing the ability to overlap P2P communication with computation. By increasing d_{hp} to 8, LoongTrain decreases the GQA execution time from 86.0ms to 56.8ms for a sequence length of 128K, and from 2681ms to 2495ms for a sequence length of 1M. However, increasing d_{hp} beyond 8 does not further reduce the GQA execution time due to the significant increase in the SeqAlltoAll communication time. For example, when d_{hp} is increased from 8 to 32, the SeqAlltoAll communication time for processing a 128K sequence with head-first placement rises from 2.64ms to 13.40ms. In summary, to process GQA efficiently, the configuration of $d_{hp} = 8$ and $d_{cp} = 8$ avoids the large SeqAlltoAll overhead and effectively overlaps the computation with P2P communication.

Device Placement Study. As analyzed in Section 4.5, there is a trade-off between the SeqAlltoAll time and total execution time when choosing the placement strategy. Table 4 shows that when d_{cp} is large (e.g., $d_{cp} = 32$), a single Attention operation can benefit from context-first placement. Although the context-first strategy increases the SeqAlltoAll time, the overall time is more advantageous due to the reduced P2P communication time. However, as d_{hp} gets larger, head-first placement performs better. In these cases, the increased large SeqAlltoAll volumes become the bottleneck of the overall execution time. Therefore, only if SeqAlltoAll leverages the intra-node high-bandwidth NVLINK can LoongTrain achieve better overall performance.

Double-Ring-Attention Study. We compare the execution time of 2D-Attention with different inner ring sizes in Table 5. As expected, with MHA and shorter sequence length, P2P communication cannot be effectively overlapped with the computation. In these cases, Double-Ring-Attention achieves more speedup. For instance, when the sequence length is 128K and $d_{cp} = 16$, Double-Ring-Attention further reduces the attention operation time by a factor of 1.2, even if 2D-Attention is already applied. However, with longer sequence lengths, due to the increased computational workload, the P2P communication can be overlapped more, limiting the improvements from Double-Ring-Attention.

As we theoretically analyzed in Section 4.5, when the inner ring size matches the number of NICs in one node (4 in our case), all NICs can be utilized for outer-ring communication, which is more effective. Table 5 also illustrates this trend. As discussed, the global batch size poses a challenge for the computation-communication ratio when scaling d_{sp} to 512 GPUs for a 1M sequence length. In such cases, Double-Ring-Attention is expected to be more useful.

7 Conclusion

We proposed LoongTrain, an efficient training framework for LLMs with long sequences. We designed the 2D-Attention, which combined both head-parallel and context-parallel approaches, to break the scalability constraints while maintaining high efficiency. We introduced the Double-Ring-Attention and device placement strategy to further improve the training efficiency. We implemented the LoongTrain system with hybrid parallelism and advanced gradient checkpoint techniques. Experiment results showed that LoongTrain provides a significant performance improvement over existing systems, such as DeepSpeed-Ulysses and Megatron CP.

8 Acknowledgements

We express our gratitude to Zilin Zhu from Tencent. Our research benefited from his GitHub repository "ring-flash-attention," which implements Ring-Attention with FlashAttention. Additionally, we are thankful to Jiarui Fang and Shangchun Zhao from Tencent for their pioneering work in integrating Ulysses and Ring-Attention, as demonstrated in the open-source project Yunchang [17]. Their guidance was instrumental in shaping this work. We also extend our thanks to Haoyu Yang and Jidong Zhai from Tsinghua University for their assistance in enhancing the performance of our implementation.

References

[1] AI@Meta. Llama 3 model card. 2024.

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

[3] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6836–6846, 2021.

[4] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of 17th European Conference on Computer Systems, EuroSys 2022*, pages 472–487, 2022.

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

[6] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Accurate medium-range global weather forecasting with 3d neural networks. *Nature*, 619(7970):533–538, 2023.

[7] 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.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Abel Chandra, Laura Tünnermann, Tommy Löfstedt, and Regina Gratz. Transformer-based deep learning for predicting protein properties in the life sciences. *Elife*, 12:e82819, 2023.

[10] Qiaoling Chen, Diandian Gu, Guoteng Wang, Xun Chen, YingTong Xiong, Ting Huang, Qinghao Hu, Xin Jin, Yonggang Wen, Tianwei Zhang, et al. Internevo: Efficient long-sequence large language model training via hybrid parallelism and redundant sharding. *arXiv preprint arXiv:2401.09149*, 2024.

[11] Qiaoling Chen, Qinghao Hu, Guoteng Wang, Yingtong Xiong, Ting Huang, Xun Chen, Yang Gao, Hang Yan, Yonggang Wen, Tianwei Zhang, and Peng Sun. Amsp: Reducing communication overhead of zero for efficient llm training, 2023.

[12] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

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

[14] 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.

[15] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *Proceedings of 26th Annual Conference on Neural Information Processing Systems, NeurIPS 2012.*, pages 1232–1240, 2012.

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

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

[18] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training, 2018. *URL https://arxiv. org/abs*, 1806.

[19] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, et al. Characterization of large language model development in the datacenter. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI'24)*, 2024.

[20] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. volume 32, 2019.

[21] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, 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.

[22] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5, 2023.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of 26th Annual Conference on Neural Information Processing Systems, NeurIPS 2012.*, pages 1106–1114, 2012.

[24] Dacheng Li, Rulin Shao, Anze Xie, Eric P Xing, Joseph E Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. Lightseq: Sequence level parallelism for distributed training of long context transformers. *arXiv preprint arXiv:2310.03294*, 2023.

[25] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Proceedings of 11th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2014*, pages 583–598, 2014.

[26] Mu Li, David G. Andersen, Alexander J. Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Proceedings of 28th Annual Conference on Neural Information Processing Systems, NeurIPS 2014.*, pages 19–27, 2014.

[27] Hao Liu and Pieter Abbeel. Blockwise parallel transformers for large context models. *Advances in Neural Information Processing Systems*, 36, 2024.

[28] Hao Liu, Wilson Yan, Matei Zaharia, and Pieter Abbeel. World model on million-length video and language with ringattention. *arXiv preprint arXiv:2402.08268*, 2024.

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

[30] Microsoft Azure Quantum Microsoft Research AI4Science. The impact of large language models on scientific discovery: a preliminary study using gpt-4. *arXiv preprint arXiv:2311.07361*, 2023.

[31] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[32] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[33] Jinjie Ni, Tom Young, Vlad Pandelea, Fuzhao Xue, and Erik Cambria. Recent advances in deep learning based dialogue systems: A systematic survey. *Artificial intelligence review*, 56(4):3055–3155, 2023.

[34] NVDIA. Megatron context parallelism, 2024.

[35] NVIDIA. Nvidia dgx superpod: Next generation scalable infrastructure for ai leadership: Reference architecture. 2023.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[37] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble pipeline parallelism. *arXiv preprint arXiv:2401.10241*, 2023.

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

[39] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[40] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 551–564, 2021.

[41] Ludan Ruan and Qin Jin. Survey: Transformer based video-language pre-training. *AI Open*, 3:1–13, 2022.

[42] Peng Sun, Yonggang Wen, Ruobing Han, Wansen Feng, and Shengen Yan. Gradientflow: Optimizing network performance for large-scale distributed dnn training. *IEEE Transactions on Big Data*, 8(2):495–507, 2019.

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

[44] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023.

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

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. volume 30, 2017.

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

[48] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. Optimized network architectures for large language model training with billions of parameters. *arXiv preprint arXiv:2307.12169*, 2023.

[49] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 558–567, 2021.

[50] Hao Zhang, Aixin Sun, Wei Jing, and Joey Tianyi Zhou. Span-based localizing network for natural language video localization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6543–6554, 2020.

[51] Zhen Zhang, Shuai Zheng, Yida Wang, Justin Chiu, George Karypis, Trishul Chilimbi, Mu Li, and Xin Jin. Mics: near-linear scaling for training gigantic model on public cloud. *Proceedings of the VLDB Endowment*, 16:37–50, 2022.

[52] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12):3848–3860, 2023.

[53] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578, 2022.

[54] Wenxuan Zhou, Kevin Huang, Tengyu Ma, and Jing Huang. Document-level relation extraction with adaptive thresholding and localized context pooling. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 14612–14620, 2021.

9 Appendix

Table 6 shows training performance (TGS) of 7B-MHA and 7B-GQA on 64 GPUs with $d_{sp} = 64$.

			128K				256K				512K				1M			
			With SC++		W/O SC++		With SC++		W/O SC++		With SC++		W/O SC++		With SC++		W/O SC++	
	d_{cp}	d_{hp}	HF	CF	HF	CF	HF	CF	HF	CF	HF	CF	HF	CF	HF	CF	HF	CF
MHA	64	1	190.2	190.2	145.3	145.3	195.4	195.4	149.4	149.4	196.8	196.8	149.9	149.9	161.7	161.7	127.6	127.6
	32	2	203.9	327.1	158.8	260.4	212.0	340.8	163.6	269.2	214.2	294.3	164.7	239.3	169.8	173.9	140.8	145.2
	16	4	363.2	505.9	290.4	422.5	386.0	464.6	304.7	389.1	318.7	319.7	260.0	262.7	185.7	182.1	149.3	147.5
	8	8	585.6	662.6	486.9	582.2	533.5	515.6	443.6	437.8	340.1	324.0	277.1	266.8	188.4	186.1	151.7	150.2
	4	16	676.9	675.9	596.3	585.0	535.2	519.5	452.4	441.1	329.9	323.0	270.4	266.8	185.5	185.9	149.3	147.2
	2	32	661.0	679.9	586.7	605.7	516.4	517.2	433.6	438.7	321.3	323.8	263.2	267.2	185.0	185.0	148.4	145.0
GQA	64	1	526.0	526.0	404.8	404.8	465.4	465.4	377.6	377.6	318.7	318.7	256.5	256.5	181.6	181.6	145.3	145.3
	32	2	585.3	655.0	480.6	555.4	514.6	527.2	424.0	435.1	333.5	328.5	270.0	265.9	186.4	184.6	149.5	148.9
	16	4	732.1	698.8	637.6	606.6	571.6	537.0	473.1	457.6	342.4	334.8	277.7	273.6	189.7	187.9	152.1	152.4
	8	8	779.7	730.6	676.0	640.8	588.9	554.7	481.3	466.4	349.8	340.6	284.3	279.2	194.0	191.6	155.6	154.3
	4	16	731.2	705.1	641.0	636.5	561.1	536.1	463.1	458.5	339.1	334.2	277.0	274.3	190.1	189.2	152.9	149.8
	2	32	666.4	687.5	589.2	609.7	520.3	517.6	428.1	441.6	322.8	325.9	264.0	267.3	185.1	185.1	148.3	145.0

Table 6. End-to-End Training Performance (TGS) of 7B-MHA and 7B-GQA on 64 GPUs with $d_{sp} = 64$. SC++ stands for *Selective-Checkpoint++*, HF for head-first, and CF for context-first. The highest TGS value in each column is highlighted.