

Zeppelin: Balancing Variable-length Workloads in Data Parallel Large Model Training

Chang Chen
charlie_chen@pku.edu.cn
Peking University

Qianchao Zhu
dysania@pku.edu.cn
Peking University

Peng Sun
sunpeng@pjlab.org.cn
Shanghai AI Laboratory

Tiancheng Chen
tiancheng.chen@inf.ethz.ch
ETH Zurich

Zerui Wang
wangzerui@pjlab.org.cn
Shanghai AI Laboratory

XiuHong Li
lixiuhong@pku.edu.cn
Peking University

Torsten Hoefler
htor@ethz.ch
ETH Zurich

Jiangfei Duan
dj021@ie.cuhk.edu.hk
The Chinese University of Hong Kong

Qinghao Hu
qinghao.hu@ntu.edu.sg
Nanyang Technological University

Chao Yang
chao_yang@pku.edu.cn
Peking University

Abstract

Training large language models (LLMs) with increasingly long and varying sequence lengths introduces severe load imbalance challenges in large-scale data-parallel training. Recent frameworks attempt to mitigate these issues through data reorganization or hybrid parallel strategies. However, they often overlook how computational and communication costs scale with sequence length, resulting in suboptimal performance. We identify three critical challenges: (1) varying computation-to-communication ratios across sequences of different lengths in distributed attention, (2) mismatch between static NIC-GPU affinity and dynamic parallel workloads, and (3) distinct optimal partitioning strategies required for quadratic attention versus linear components.

To address these challenges, we present *Zeppelin*, a novel training system that integrates three key techniques: (1) a hierarchical sequence partitioning method for the attention module that reduces communication overhead and balances computation, supported by an efficient attention engine that applies divergent parallel strategies; (2) a routing layer that orchestrates inter-node transfers to fully utilize NIC bandwidth; and (3) a remapping layer that transforms sequence layouts between attention and linear modules, ensuring high computational efficiency across both. Comprehensive evaluations across diverse configurations show that *Zeppelin* delivers an average 2.80 \times speedup over state-of-the-art methods.

CCS Concepts: • Computing methodologies → Distributed artificial intelligence.

Keywords: Machine Learning System, Large Model Training, Distributed System

ACM Reference Format:

Chang Chen, Tiancheng Chen, Jiangfei Duan, Qianchao Zhu, Zerui Wang, Qinghao Hu, Peng Sun, XiuHong Li, Chao Yang, and Torsten Hoefler. 2026. Zeppelin: Balancing Variable-length Workloads in Data Parallel Large Model Training. In *European Conference on Computer Systems (EUROSYS '26), April 27–30, 2026, Edinburgh, Scotland UK*. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3767295.3769369>

1 Introduction

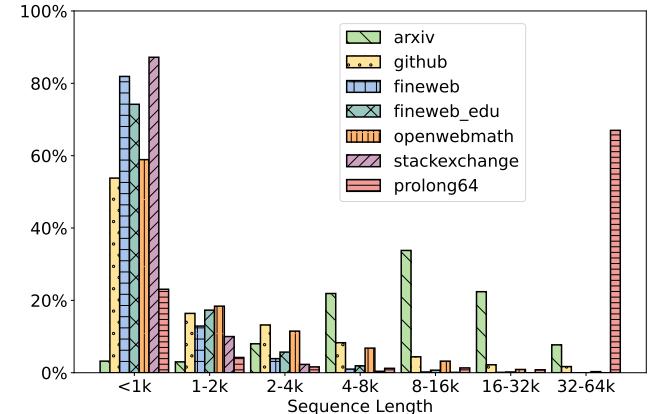


Figure 1. Sequence length distribution in multiple datasets: typical LLM training involves a mixture of datasets with diverse and often long-tailed sequence length distributions [18, 28, 40, 41, 46].

Data parallelism (DP) is a well-established paradigm for training large language models (LLMs). It distributes large mixtures of datasets across multiple devices, with each device processing its local batch using an identical model replica. Two recent trends in pretraining data mixture [12, 21, 34, 50] are the use of much longer sequences and the necessity

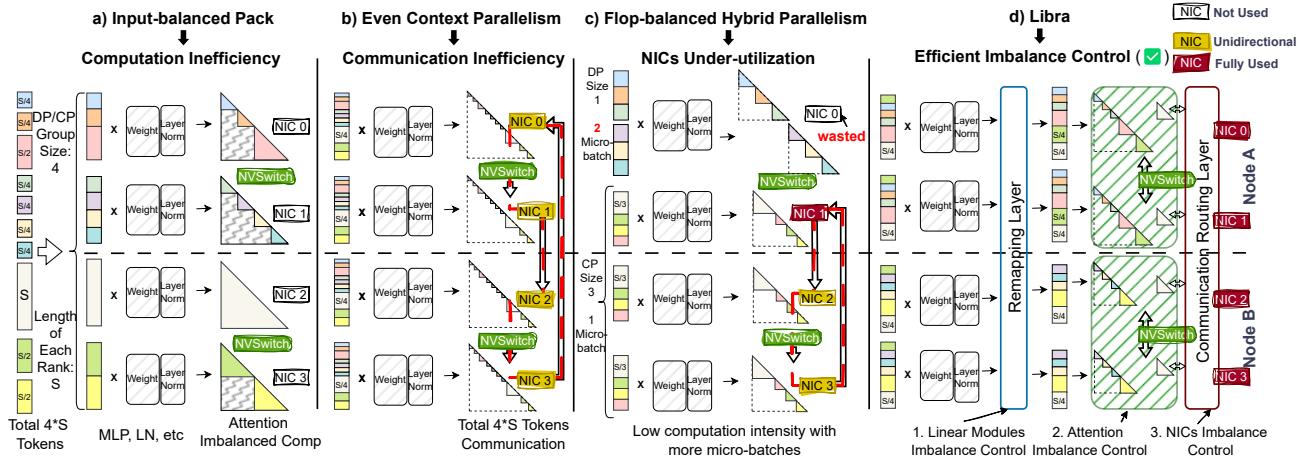


Figure 2. Balancing a complex system often propagates inefficiency internally. a) Input balancing [12, 50] struggles with quadratic scaling of attention, leading to computation inefficiency. b) Even sequence splitting [34, 39, 53] balances computation but incurs high communication overhead, especially for short sequences. c) Hybrid parallel methods [19, 52] lead to uneven hardware (NIC) utilization and computational intensity across ranks. d) *Zeppelin* addresses imbalance holistically from the model to hardware.

of training on mixtures of datasets with highly variable sequence lengths, as shown in Fig. 1. Longer sequences are crucial for capturing complex dependencies and improving model capabilities [3, 21, 34], but they impose a quadratic computational cost in the self-attention module. Variable-length sequences are essential for models to handle diverse inputs, from short queries to long documents [7, 17, 18, 30, 54]. Together, these long and variable-length inputs introduce severe load imbalance across DP ranks when the context window scales to hundreds of thousands of tokens, as seen in recent large-scale distributed training recipes [12, 19, 34, 50, 53].

As illustrated in Fig 2, recent methods attempt to mitigate load imbalance through data reorganization or hybrid data parallelism, typically optimizing for a balance metric. **Input-balanced pack** (Fig. 2.a) creates uniformly sized input tensors at the start of each iteration using techniques such as sequence packing or chunking [13, 14, 42, 58]. This approach is adopted in the Qwen [49, 50] and DeepSeek [11, 12] model families. While effective for balancing computation in linear modules, it often leads to redundant attention computation or imbalanced attention masking. **Even context parallelism (CP)** (Fig. 2.b) splits each sequence across devices to balance attention computation, employing distributed attention mechanisms method [4, 23, 26, 29, 32, 34]. This strategy is adopted in LLaMA 3 [34, 53] training and the Megatron-LM framework [9, 39, 47]. However, it introduces substantial communication overhead, particularly for the numerous short sequences common in diverse datasets. To reduce communication costs, **flop-balanced hybrid parallelism**

(Fig. 2.c) combines DP and CP, assigning short and long sequences to separate micro-batches [19, 20, 52]. While this balances end-to-end flop across devices, DP ranks become NIC-underutilized when processing more micro-batches of short sequences, whereas CP ranks remain computation-heavy but memory-underutilized.

Despite aiming to balance computation among devices, existing approaches often overlook the distinct scaling behaviors of LLM modules with respect to sequence length. First, different modules exhibit different complexities: linear modules (e.g., MLP) scale linearly, while attention scales quadratically with sequence length [5, 51]. An input distribution that is balanced for linear modules becomes inefficient for quadratic ones when length distribution is highly dynamic. Second, distributed attention incurs communication volume proportional to sequence length [26, 31, 34], leading to large discrepancies in computation-to-communication ratios across variable-length sequences. This indicates that such sequences place different demands on both computational resources and heterogeneous interconnects. However, in modern training systems each GPU is connected to a NIC [37, 38] through a PCIe switch, and these network resources are often underutilized in distributed attention. Given these insights, the performance of a training system can be significantly compromised by variable-length inputs, and optimizations based solely on a balance metric cannot deliver optimal performance.

We present *Zeppelin*, a balanced and efficient data-parallel LLMs training system for variable-length sequences. *Zeppelin* incorporates three key design innovations:

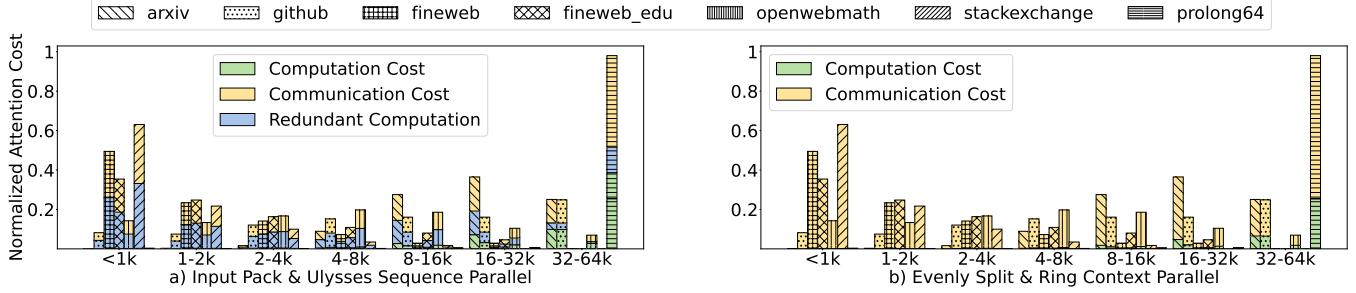


Figure 3. Multi-head attention cost distribution across different sequence length ranges in each dataset (normalized to the total attention cost per dataset). Evaluation on a 2-node system with 8 A800 GPUs per node, a total sequence length of 64k, and 4×200 Gbps NICs per node. (a) Packing-based input balancing introduces significant redundant computation, especially for short sequences. b) CP with even sequence splitting incurs substantial unnecessary communication for short sequences, and cross-node communication is difficult to fully overlap with computation.

First, mismatched scaling trends between computation and communication in distributed attention necessitate a fine-grained hybrid parallel strategy. To leverage the hierarchical bandwidth structure of modern interconnects, we categorize context parallel strategies into three types: local, intra-node, and inter-node, and propose a two-step, topology-aware sequence partitioning and placement method. Building on this partitioning, a flexible attention engine orchestrates the execution of these sequence types, ensuring efficient overlap between computation and communication.

Second, the dynamic communication patterns of variable-length sequences require disaggregating the fixed affinity between GPUs and NICs within each node. To fully utilize all NICs, we propose a three-step routing mechanism that substitutes direct inter-node communication with (1) intra-node dispatch, (2) multi-NIC inter-node transfer, and (3) intra-node combine. This design alleviates inter-node bandwidth bottlenecks and enables overlap of computation with both intra- and inter-node data transfers.

Third, balancing computation in linear modules requires a remapping mechanism aligned with the attention partitioning scheme. The remapping layer adjusts sequence layouts before and after linear modules, ensuring balanced workloads. We formalize this as an optimization problem to minimize remapping communication cost.

To conclude, this paper makes the following contributions:

- **Characterization of scaling behavior.** We analyze how key components scale with sequence length, including quadratic modules (attention), linear modules (MLPs), distributed attention with linear communication cost, and hardware-level GPU-NIC affinity.
- **System design.** We propose a bandwidth-aware partitioning strategy that categorizes sequences into local, intra-node, and inter-node types. Based on this, we design and implement a distributed attention engine to efficiently manage parallel execution. To accelerate dynamic communication workloads, we introduce a three-step routing

scheme that fully utilizes all NICs within a node, overlapping computation with intra- and inter-node communication. Additionally, we design a remapping layer that balances workloads in linear modules with minimal overhead.

- **Comprehensive evaluation.** We conduct extensive experiments across diverse models, sequence lengths, and training scales. Results demonstrate that *Zeppelin* achieves an average speedup of **2.80**× over state-of-the-art methods.

2 Background and Motivation

2.1 Transformer Architecture

LLMs based on the transformer architecture typically compromise a stack of transformer layers [5, 34, 50], each containing a quadratic self-attention module alongside several linear computation modules. The self-attention mechanism captures contextual dependencies across the entire sequence, introducing quadratic computational complexity with respect to sequence length [51]. Owing to causal dependencies among tokens, the attention mask typically follows a lower-triangular pattern. In contrast, operations such as normalization, linear projection, and activation functions are token-wise, enabling each token to be processed independently. These operations are referred to as "linear modules" collectively in this paper.

2.2 Data Parallel Variants

Traditional data parallel methods pack input tensors along the batch dimension. Before the era of large models, data parallelism primarily focused on increasing batch size to improve throughput [22, 48, 55, 56]. When inputs were small, simple padding could easily achieve load balance without harming training performance [15, 25].

In transformer-based large model training, however, both sequence length and model size have increased significantly.

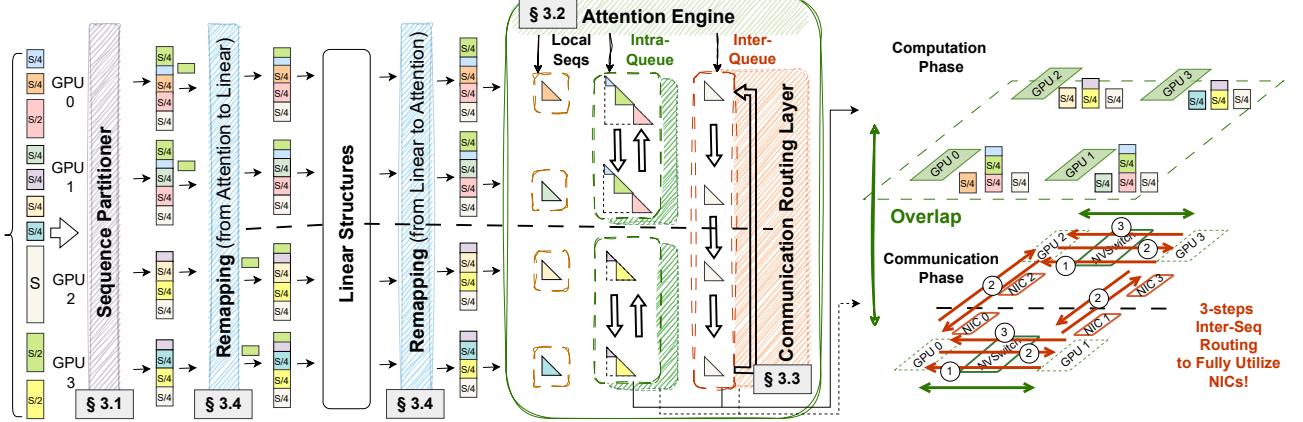


Figure 4. Overview of *Zeppelin*: The Sequence Partitioner assigns input sequences to minimize imbalance caused by quadratic attention. Before entering linear modules, sequences are remapped into a linearly balanced pattern and then restored after linear computation. The Attention Engine orchestrates three types of sequence queues, overlapping computation with communication. Meanwhile, the Communication Routing Layer optimizes inter-node data transfers by replacing direct cross-node communication with a three-step routing mechanism, ensuring full utilization of available NIC bandwidth.

To meet these demands, various distributed training strategies have been developed, broadly classified into two orthogonal types: Model Parallelism (splitting model weights, e.g., tensor and pipeline parallelism [35, 47]) and Data Parallelism variants. Key DP variants include ZeRO methods [45], Sequence Parallelism (SP) [29], and Context Parallelism (CP) [39]. Megatron-LM’s SP [29] inserts all-gather and reduce-scatter operations to synchronize intermediate activations, while Ulysses SP [26] in DeepSpeed [45] employs all-to-all communication to better manage communication and memory, though it requires the group size to be a factor of the number of attention heads. CP in Transformer Engine [39] utilizes ring attention, processing queries iteratively with different key-value chunks while overlapping send-receive operations via a ring topology [31, 32]. Many recent DP variants are flexible combinations of these base methods [16, 23].

These sequence-parallel approaches can be further adapted to dynamic-length input. Packing and padding strategies naturally integrate with SP, while evenly splitting each sequence aligns well with CP [9, 39]. Hybrid DP methods assign sequences of different lengths to appropriate variants and balance floating-point operations (flop) across ranks to reduce communication overheads [19, 52, 53].

2.3 Load Balance, at What Cost?

All DP variants discussed above aim to balance training with respect to a chosen metric, such as memory usage or computational flop. However, these “balanced” approaches inevitably introduce new source of overhead, as illustrated in Fig. 3. We identify three major inefficiencies underlying these methods:

- **Computation Inefficiency:** Input-balanced packing approaches successfully balance linear computations but introduce significant inefficiency in attention. As shown in Fig. 3.a, packing wastes a substantial portion of computation on cross-sequence attention and incurs high communication cost. In datasets with many short sequences, redundant cross-sequence computation and communication dominate attention overhead, reaching up to 60% for sequences shorter than 1k tokens in StackExchange [28], as shown in Fig. 3.a.
- **Communication Inefficiency:** Evenly partitioning sequences across devices balances both memory and computation but incurs excessive communication overhead, especially for batches with many short sequences. As shown in Fig. 3.b, the low computation-to-communication ratio of short sequences severely limits performance in distributed attention. Consequently, a single parallel strategy cannot effectively handle batches with diverse sequence lengths under heterogeneous inter- and intra-node bandwidth conditions.
- **Hardware Under-utilization:** Hybrid strategies that combine multiple DP variants attempt to reduce communication costs by splitting batches across different DP groups. However, this introduces imbalanced hardware utilization and computation intensity across parallelism strategies. As illustrated in Fig. 2.c, this often requires more micro-batches to balance memory and flop, reducing per-microbatch token count and overall compute intensity. Moreover, mixing different DP schemes frequently leads to uneven utilization of NICs. Achieving optimal performance thus requires simultaneously maintaining high compute intensity and high NIC utilization.

Table 1. Notations

S	Total Number of Sequences
s_i	i -th Sequence Length
P	Number of Devices per Node
N	Number of Nodes
L	Token Capacity of Each GPU
$b_{\text{inter}}, b_{\text{intra}}$	inter-, intra- inverse bandwidth cost

3 Libra Design

Building on the above analysis, we argue that achieving comprehensive and efficient load balance with respect to a single metric is inherently difficult. Instead, we identify structural inefficiencies in existing data-parallel training system and propose a new system design, *Zeppelin*, to address them holistically. As illustrated in Fig. 4, *Zeppelin* incorporates four key design components that collectively mitigate load imbalance across the training system:

- ① **Sequence Partitioner** (§ 3.1) splits and places sequences using a two-level hierarchical strategy, accounting for the differing computation and communication demands of variable-length sequences in distributed attention.
- ② **Attention Engine** (§ 3.2) dynamically manages sequence execution across local, intra-node, and inter-node tiers, adapting to the diverse execution characteristics of different sequence parallel types.
- ③ **Communication Routing Layer** (§ 3.3) addresses imbalances in communication patterns by decoupling GPU-NIC affinity. This enables better utilization of communication resources and improved overlap of computation and communication.
- ④ **Remapping Layer** (§ 3.4) improves performance by dynamically adjusting placement of sequences for linear structures with small data transfer overheads, guided by the attention partitioning pattern.

3.1 Sequence Partitioner

The end-to-end training cost compromises two main components: attention modules and linear modules. As discussed earlier, efficient sequence splitting and placement strategies differ between these components, particularly under dynamic input length distributions. Since quadratic attention constitutes the primary bottleneck in long-sequence training (Fig. 5), we begin our analysis with the attention module. The analysis of linear modules, together with the associated remapping strategy, is deferred to Section 3.4.

In practice, quadratic attention computation is distributed across a group of ranks to alleviate computation overhead. However, this distributed attention pattern requires transferring large volumes of key-value (KV) activations. The direct Allgather approach, adopted by LLaMA 3 training [34, 53], increases peak memory usage and places communication on the critical path. To improve overlap and reduce memory pressure, communication can instead be decomposed into

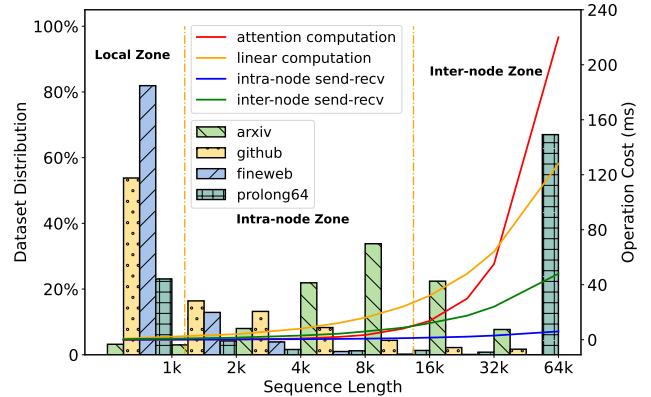


Figure 5. Attention computation cost on an A800 GPU and send-receive cost under 200 Gb/s inter-node and 400 GB/s intra-node bandwidths for different sequence lengths. The intersections of the three cost curves divide datasets into three zones: local, intra-node, and inter-node, representing the overlapping capabilities of sequences of different lengths.

rounds of send-receive operations, which interleave naturally with sharded attention computation. The send-receive-based ring attention mechanism [32] provides greater flexibility, offering a more adaptable foundation for per-sequence parallelism by supporting dynamic group sizes and fine-grained partitioning, compared to a single-synchronization Allgather across a rank group.

Based on the ring attention pattern, we identify a core objective in sequence splitting and placement: maximizing overlap between computation and communication over a hierarchical network. As shown in Fig. 5, computation scales more rapidly than communication, yielding a computation-to-communication ratio that increases linearly with sequence length. This trend indicates that longer sequences can more effectively hide communication costs, while shorter sequences suffer from inefficient communication. Leveraging this insight, we categorize sequences into three distinct zones, as depicted in Fig. 5.

- **Local zone:** Short sequences that require no communication and are most efficiently processed on a single device.
- **Intra-node zone:** Medium-length sequences that benefit from overlapping intra-node communication with computation.
- **Inter-node zone:** Long sequences that span multiple nodes, where high computation cost effectively overlaps inter-node communication.

This three-zone categorization aligns with the bandwidth hierarchy of modern GPU training infrastructures, where inter-node bandwidth is typically an order of magnitude lower than intra-node bandwidth. With this model, the optimization problem reduces to two key decisions: ① determining

Algorithm 1: Inter-Node Partitioning

Input: Input Sequences S , Devices per node P , Token capacity per device L , Number of Nodes N ,

- 1 Sort S in descending order by sequence length;
- 2 Initialize inter-node zone threshold $s_1 = P * L$;
- 3 **do**
- 4 $node_buckets = [[\cdot] \times N]$, flag = False ;
- 5 intra-node/local zone $z_{01} = \{|\mathbf{s}| < s_1 | \mathbf{s} \in S\}$;
- 6 inter-node zone $z_2 = \{|\mathbf{s}| \geq s_1 | \mathbf{s} \in S\}$;
- 7 per-node length budget $s_{avg} = \sum_{\mathbf{s} \in z_2} |\mathbf{s}| / N$;
- 8 **for** \mathbf{s} in z_2 **do**
- 9 | Split \mathbf{s} evenly to $\lceil |\mathbf{s}| / s_{avg} \rceil$ empty buckets;
- 10 **end**
- 11 **for** \mathbf{s} in z_{01} **do**
- 12 $idx = argmin_i(\sum_{ss \in node_buckets[i]} |ss|)$;
- 13 **if** $|\mathbf{s}| + \sum_{ss \in node_buckets[idx]} |ss| > P * L$ **then**
- 14 | Update threshold: $s_1 = max\{z_{01}\}$;
- 15 | flag = True; **break**;
- 16 **else**
- 17 | $node_buckets[idx].push(\mathbf{s})$;
- 18 **end**
- 19 **end**
- 20 **while** flag ;

Output: $node_buckets$

the appropriate zone for each sequence and ② partitioning and placing sequences within each zone at the proper granularity.

We propose a hierarchical partitioning strategy consisting of two steps: inter-node partitioning followed by intra-node partitioning. As illustrated in Alg. 1, the inter-node stage determines the boundary s_1 between the inter-node z_2 and intra-node/local z_{01} zones, and assigns them into N node-level buckets to optimize communication, which is the primary bottleneck at this level. The algorithm iteratively adjusts s_1 : it is initially set by the per-node token capacity (Line 2), and inter-node sequences z_2 are then chunked based on the average per-node budget s_{avg} to balance load. Instead of spreading sequences uniformly across all nodes, we increase the partition granularity for cross-node sequences and assign them to separate GPUs (Lines 7–10), improving communication efficiency. The remaining shorter sequences in z_{01} are assigned to the least-loaded node buckets (Lines 11–15). If any sequence in z_{01} exceeds node capacity (Lines 13–15), s_1 is reduced to the maximum of z_{01} and the process repeats. The iterative refinement guarantees that all sequences shorter than the final s_1 can be placed within node capacity.

At the intra-node stage, Alg. 2 further partitions the sequences within each node across P devices. Similar to Alg. 1, the algorithm determines the boundary s_0 between intra-node z_1 and local z_0 sequences. Inter-node sequences are

Algorithm 2: Intra-Node Partitioning

Input: Intra-node Sequences z_{01} , Inter-node Sequences z_2 (at current node), Devices per Node P , Token Capacity per Device L

- 1 Initialize threshold $s_0 = L$;
- 2 **do**
- 3 $device_buckets = [[\cdot] \times P]$, flag = False ;
- 4 **for** \mathbf{s} in z_2 **do**
- 5 | Split \mathbf{s} evenly to P devices;
- 6 **end**
- 7 $z_0 = \{|\mathbf{s}| < s_0 | \mathbf{s} \in z_{01}\}$, $z_1 = \{|\mathbf{s}| \geq s_0 | \mathbf{s} \in z_{01}\}$;
- 8 per-device budget $c_{avg} = \sum_{\mathbf{s} \in z_1} (|\mathbf{s}|^2) / P$;
- 9 **for** \mathbf{s} in z_1 **do**
- 10 | Split \mathbf{s} into $\lceil |\mathbf{s}|^2 / c_{avg} \rceil$ fragments;
- 11 | Assign fragments to buckets in round-robin fashion;
- 12 **end**
- 13 **for** \mathbf{s} in z_0 **do**
- 14 $idx = argmin_i(\sum_{ss \in device_buckets[i]} |ss|)$;
- 15 **if** $|\mathbf{s}| + \sum_{ss \in device_buckets[idx]} |ss| > L$ **then**
- 16 | Update threshold: $s_0 = max\{z_0\}$;
- 17 | flag = True; **break**;
- 18 **else**
- 19 | $device_buckets[idx].push(\mathbf{s})$;
- 20 **end**
- 21 **end**
- 22 **while** flag ;

Output: $device_buckets$

evenly distributed across P GPUs. Since intra-node communication overlaps more effectively with computation, intra-node sequences are split to balance their quadratic computation across devices, using c_{avg} as the per-device budget. Finally, local sequences are assigned to individual devices. If any local sequence exceeds device capacity L , s_0 is adjusted iteratively to the maximum of z_0 . This iterative adjustment guarantees feasible placement while balancing computation and communication within each node.

3.2 Attention Engine

With the sequence assignments, each device receives sequences categorized into three types: inter-node, intra-node, and local. Each type mapped to a distinct ring communication group. Compared to the evenly split pattern, where all sequences are evenly split across a single global ring [4, 39], the hierarchical ring-group mapping reduces the communication overhead from $b_{inter} \sum_{i \in S} s_i$ to $\max b_{inter}s + \max b_{intra}s$,

where b_{inter} and b_{intra} denote the inverses of inter- and intra-node bandwidths, respectively. Hybrid methods [19, 52] that partition sequences into multiple communication groups typically apply coarse-grained model-level parallelism, which often introduces imbalance among ranks. In contrast, *Zeppelin*

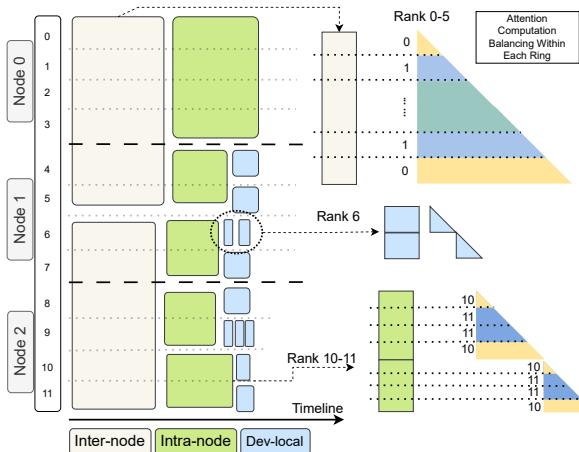


Figure 6. Attention Engine orchestrates partitioned sequences on each device, performing computations in the order of inter-node, intra-node, and local. Sequences are partitioned to balance the computational load, considering the triangular attention mask.

employs a fine-grained attention-level execution engine. It orchestrates three separate queues, ensuring balanced workload distribution while aligning with the underlying network hierarchy.

As shown in Fig. 6, execution proceeds in the order of inter-node, intra-node, and local sequences. This ordering is crucial for maximizing efficiency. Because the inter-node communication group spans and subsumes intra-node groups on participating nodes, completing inter-node tasks first enables immediate launching of intra-node queues without blocking. In contrast, executing intra-node tasks first would delay inter-node launches, as they must wait for all intra-node operations across nodes to finish, introducing idle time. Finally, local sequences, requiring no communication, are executed last to avoid interfering with communication-dependent tasks.

The lower-triangular computation pattern in causal self-attention [5, 34] requires a fine-grained sequence partitioning strategy for inter- and intra-node rings to achieve balanced load across participating ranks [53]. As illustrated in Fig. 6, sequences within a queue, e.g., an inter-node ring of size G_{inter} or an intra-node ring of size G_{intra} , are further divided into $2 \times G$ equal-length chunks, where G denotes the ring group size. Each rank i in the ring is assigned the i -th and the $(2G - i - 1)$ -th chunks, ensuring balanced computation among all ranks in the group. Execution follows the standard ring attention pattern: [4, 31, 32], the execution of each queue involves G rounds of computation overlapped with send-receive communication. In each round, rank i sends its current KV activations to rank $i + 1$ and receives the next round's KV activations from rank $i - 1$. Local sequences are executed directly on their assigned GPU using a

standard variable-length attention kernel without inter-rank communication.

3.3 Communication Routing

The hierarchical partitioning strategy, combining local, intra-node, and inter-node parallel patterns in distributed attention, helps reduce overall communication volume. However, the flexible combination of these methods introduces diverse communication patterns that can still result in hardware under-utilization. For example, devices processing local or intra-node sequences may leave their dedicated NICs idle while other devices on the same node engage in inter-node communication, as shown in Fig. 7. Within an inter-node ring, the structured send-receive pattern, e.g., rank i sends to $i + 1$ and receives from $i - 1$, can also lead to NIC under-utilization. NICs associated with GPUs not directly involved in the cross-node communication remain idle, and even active NICs are underused because ring attention only transfer data unidirectionally, leaving the opposite direction unused.

To fully utilize all NICs for flexible routing, *Zeppelin* introduces a Communication Routing Layer that disaggregates logical communication paths from fixed GPU-NIC affinities. We define proxy ranks as the ranks within a node responsible for inter-node transfers. Considering an inter-node ring: let x_1 and x_2 denote the number of proxy ranks at the current node for sending and receiving, respectively. In each round of send-receives, the inter-node communication for KV activation of total size n is decomposed into three-steps, as shown in Fig. 7:

- **Workload Dispatch (Intra-node):** The source rank scatters its n tokens to x_1 send proxy ranks using high-bandwidth intra-node links, with each proxy handling about n/x_1 tokens.
- **Inter-node Transfer (Multi-NIC):** The x_1 send proxy ranks deliver their data portions to the destination node, while x_2 receive proxy ranks simultaneously retrieve data from the source node.
- **Workload Combine (Intra-node):** Each of the x_2 receive proxy ranks forwards its n/x_2 tokens to the designated destination rank.

The direct transfer cost, $b_{\text{inter}} \cdot n$, is thus optimized to:

$$b_{\text{intra}} \frac{n(x_1 - 1)}{x_1} + b_{\text{inter}} \max\left(\frac{n}{x_1}, \frac{n}{x_2}\right) + b_{\text{intra}} \frac{n(x_2 - 1)}{x_2} \quad (1)$$

Given the typical $10\times$ bandwidth gap between b_{intra} , b_{inter} in modern GPU clusters, even a small number of proxy ranks can significantly reduce the inter-node communication bottleneck.

To implement this, GPUs are designated as send and receive proxy ranks in a balanced manner. GPUs already participating in ring communication act as proxy ranks for their groups, while GPUs handling local or intra-node sequences can also serve as proxies for inter-node transfer. For inter-node rings spanning multiple nodes, the number of proxy

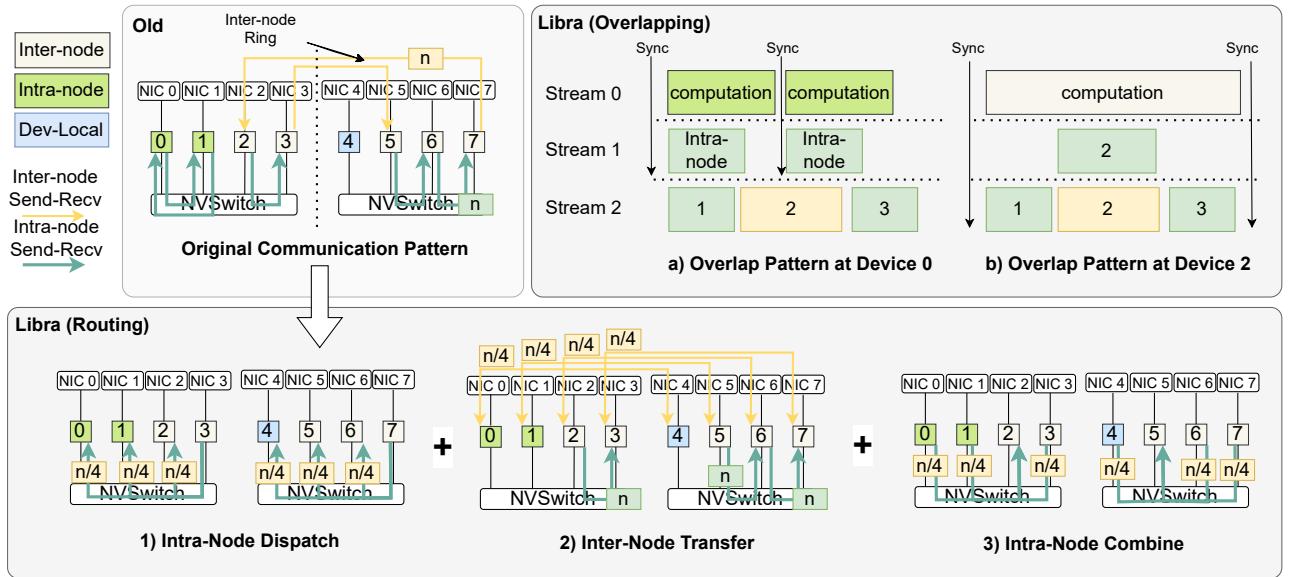


Figure 7. The send-receive operations of an inter-node ring for a data volume of n are decomposed into three steps: 1) local dispatch: the inter-node send workload is first distributed to designated proxy ranks within each node, 2) inter-node exchange: these proxy ranks perform the inter-node send-receives for the dispatched workload, overlapped with ongoing intra-node transfers, and 3) local gather: the data received from remote nodes is gathered on the target rank. Ranks assigned for intra-node and local attention computation also serve as proxy ranks to route the data.

GPUs involved in communication may differ across nodes. To ensure balanced communication, the number of send proxy ranks x_1 at each node is set to the minimum number of GPUs assigned to the current and destination nodes; the number of receive proxy ranks x_2 is set analogously, based on the current and source nodes. This pairing strategy ensures one-to-one matching of senders and receivers.

The optimized inter-node communication can be scheduled to overlap with local computation and intra-node transfers. As shown in Fig. 7.a, intra-node send-receives can proceed simultaneously with cross-node data transfer, improving hardware utilization. For ranks within a ring, their intra-node communication can also be scheduled to overlap the cross-node data transfer in Fig. 7.b.

3.4 Remapping Layer

After addressing attention imbalance, linear computation still contributes a non-negligible share of the total overhead, particularly when the sequence length distribution is highly right-skewed, with many short sequences and a few very long ones, as shown in Fig. 5. The partitioning strategy optimized for attention often leads to token distributions where some ranks are heavily loaded while others may be underutilized. In contrast, linear structures such as MatMul, LayerNorm, and Mixture-of-Experts (MoEs) achieve optimal efficiency when tokens are evenly distributed across ranks. Consequently, a strategy that balances quadratic attention computation may be misaligned with the optimal distribution for linear modules, leading to inefficiencies.

To bridge this gap, we introduce a Remapping Layer that dynamically adjusts the sequence distribution across ranks before and after the linear modules. Prior to linear computation, the remapping layer transforms the attention-optimized layout into a token-balanced layout. Afterwards, it performs the inverse transformation, incurring an equivalent cost, thereby aligning both attention and linear structures with their respective optimal execution patterns.

To minimize the communication overhead of remapping operations, we formalize the problem as an optimization task. Let A be a d -dimension vector representing the current token distribution across d ranks in a remapping group. The target distribution B is a d -dimensional vector in which each element B_i equals the average number of tokens, i.e., $\sum A_i/d$. The goal is to find a transfer matrix $M \in \mathbb{R}^{d \times d}$, which minimizes the data transfer cost required to transform A into B :

$$\begin{aligned} & \arg \min_M \| (T * M) \mathbf{1} \|_\infty \\ & \text{s.t. } \sum_j (M_{ij}) = \max\{A - B, 0\}_i, \forall i; \\ & \quad \sum_i (M_{ij}) = \max\{B - A, 0\}_j, \forall j; \\ & \quad M \geq 0; \end{aligned} \tag{2}$$

Here, M_{ij} denotes the volume of tokens transferred from rank i to rank j , and T is a symmetric cost matrix determined by network bandwidth: $T_{ij} = b_{\text{inter}}$ for inter-node remapping,

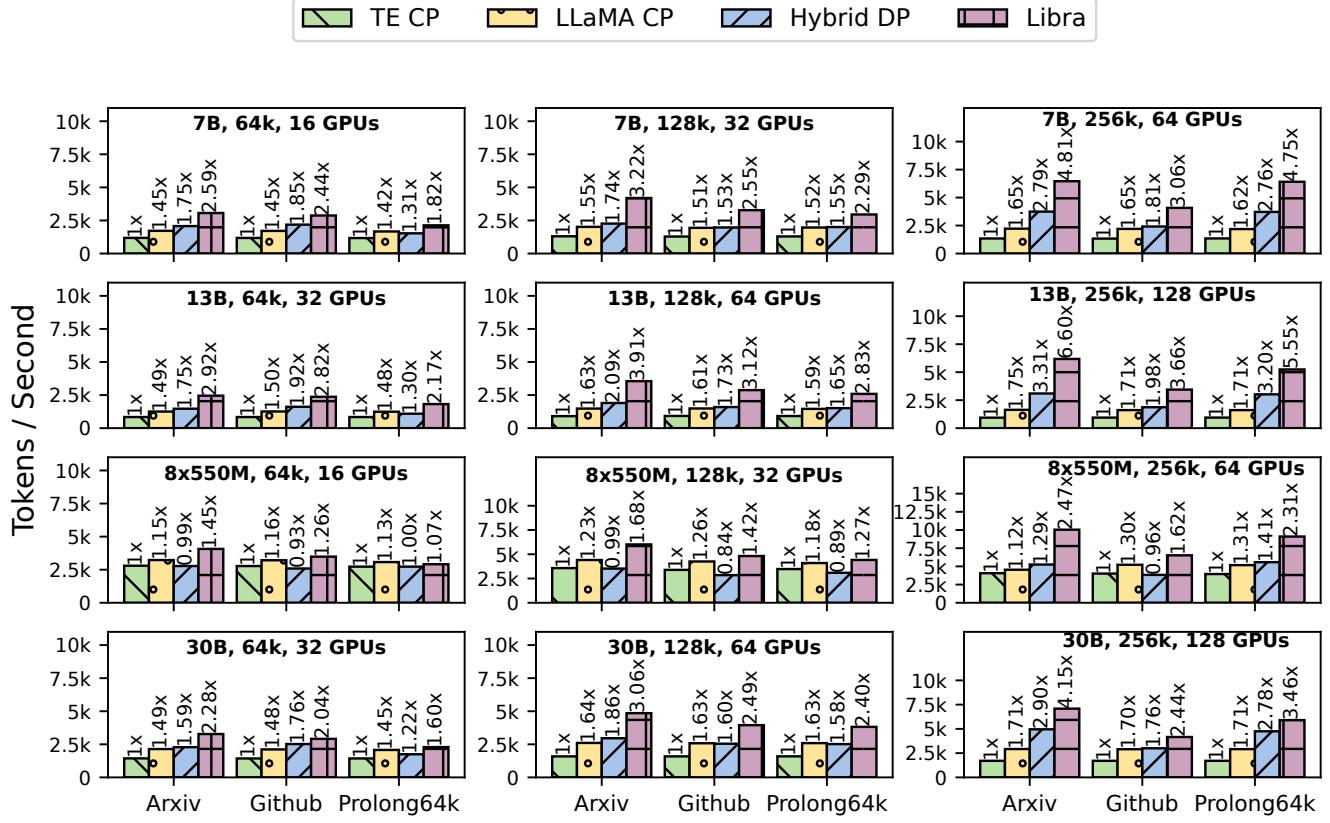


Figure 8. Throughputs for three model configurations—7B, 13B, 30B, and $8 \times 550M$ —evaluated on three dataset types, across total context lengths ranging from 64k to 256k (with 4k per GPU), on Cluster A & C.

and $T_{ij} = b_{\text{intra}}$ otherwise. The first constraint ensures that ranks only send surplus tokens, while the second constraint ensures that deficits are satisfied. The objective minimizes the maximum communication cost incurred by any rank, thereby balancing the transfer overheads. This formulation corresponds to a minimum-cost flow problem and can be solved efficiently using standard solvers such as Gurobi [1].

4 Implementation

We implemented the core of *Zeppelin* with 2k lines of C++/CUDA and 6k lines of Python, extending Megatron-LM [47] and Transformer Engine [39], two widely adopted open-source LLM training frameworks. The communication routing layer includes custom NCCL kernels for the three-step routing procedure, which flexibly assign inter-node send and receive ranks. These kernels are scheduled across multiple CUDA streams with a stream manager that prioritizes kernel launches to maximize execution overlap. The attention engine, integrated into Transformer Engine, manages the execution of three sequence queues (local, intra-node, and inter-node) on a dedicated computation stream. A global context-parallel process group is dynamically partitioned into multiple logical ring, and the engine asynchronously launches routing-layer communication kernels or intra-node

Table 2. Sequence length distribution of three datasets. Values represent the proportion of sequences within each length bin (lengths in thousands of tokens).

Dataset	<1	1-2	2-4	4-8	8-16	16-32	32-64	64-128	128-256
Arxiv	0.032	0.03	0.08	0.219	0.338	0.224	0.077	0	0
Github	0	0.34	0.095	0.104	0.107	0.102	0.088	0.064	0.045
Prolong64k	0.231	0.042	0.021	0.012	0.013	0.008	0.673	0	0

send-receive operations to overlap communication with computation. The sequence partitioner and remapping layers are integrated into Megatron-LM. The sequence partitioner leverages the global batch length distribution to execute the hierarchical partitioning algorithm and generate per-rank placement plans. The remapping layer computes optimal data transfer strategies via a standard solver [1] and executes them with a dynamic-shape `alltoallv` primitive that supports both forward and backward passes.

5 Evaluation

Experimental Setup We conduct experiments on three GPU clusters: Cluster A, B, and C. In Cluster A, each node is equipped with 8 NVIDIA A800-80G GPUs interconnected via NVSwitch, providing 400 GB/s of intra-node bandwidth. Each node also has 4 RoCE NICs, with each NIC shared by 2 GPUs, yielding an aggregate cross-node bandwidth of 4×200 Gb/s. In contrast, Cluster B features nodes equipped with 8

NVIDIA H800 GPUs and 8 RoCE NICs. Cluster C nodes contain 8 H200 GPUs and 8 CX7 NICs of 8×400 Gb/s bandwidth, enabling significantly higher cross-node bandwidth with one-to-one GPU-NIC mapping. All clusters run a uniform software stack: CUDA 11.8, cuDNN 8.9.6, NCCL 2.14 [36], PyTorch 2.4.0 [2], FlashAttention 2.4.3 [10], Megatron-LM 0.8.0rc0 [47], and Transformer Engine v1.8 [39].

Model and Dataset We use LLaMA [33] with multi-head attention as the baseline architecture, given its competitiveness and widespread adoption. Five representative configurations are evaluated: 3B, 7B, 13B, 30B dense, and 8×550 M MoE.

For datasets, we evaluate on three representative long-context datasets: ArXiv [46], GitHub [19], and Prolong64k [18]. Synthetic datasets are generated to match the length distributions of these benchmarks, with batch sequence lengths sampled proportionally to dataset distributions. The detailed length distributions are shown in Table 2. Throughput is reported as processed tokens per second, averaged over steps 50–150.

Baseline We compare *Zeppelin* against several state-of-the-art methods:

- **Transformer Engine CP [39]**: evenly splits sequences across devices and applies balanced ring attention.
- **LLaMA CP [34, 53]**: replicates the CP approach in LLaMA training, where KV activations are all-gathered across devices prior to attention computation.
- **Hybrid DP [19]**: combines standard DP for short sequences with ring-based CP for long sequences to balance FLOPs. When short sequences exceed memory limits, they are further chunked into smaller micro-batches.

Evaluation Scope We evaluate *Zeppelin* across four aspects: ① end-to-end performance across models, datasets, and scales; ② scalability and adaptability to different network architectures; ③ ablation studies of key design points; ④ case studies with execution timeline analysis and length distribution effects.

5.1 End-to-End Throughput

We evaluate end-to-end training throughput across diverse combinations of four model architectures, three datasets, and three context length scales, as shown in Fig. 8. *Zeppelin* consistently outperforms other SOTA baselines, achieving up to $6.60\times$ speedup and an average speedup of $2.80\times$ over the TE baseline.

For 7B model training, throughput improvements are consistent across different context lengths and dataset distributions. At each context length, datasets with shorter length distributions (e.g., ArXiv) are partitioned efficiently to reduce more communication costs than skewed distributions (e.g., GitHub) with longer sequence lengths, resulting in larger speedups. For the 13B model on Cluster A and 30B model on Cluster C, context parallel training is combined with tensor parallelism of size 2. The speedup trend mirrors the 7B case. An interesting observation is that the speedups achieved

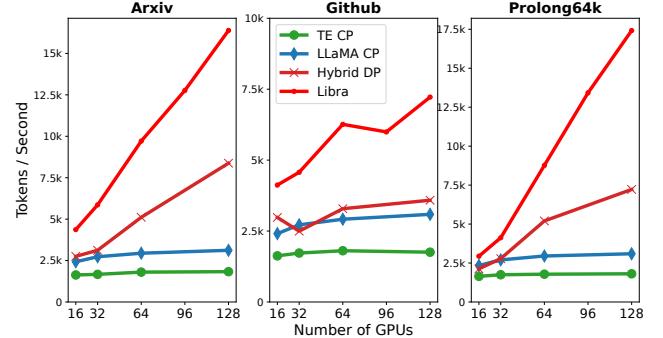


Figure 9. Scalability on the LLaMA 3B (Cluster A). Throughput is plotted against the number of GPUs (16–128), with context length set fixed at 4K tokens per GPU.

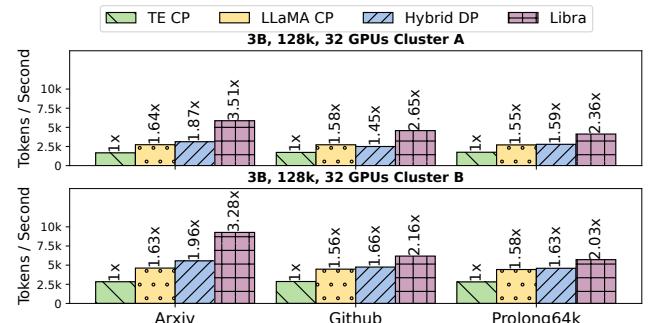


Figure 10. Speedup comparison on Cluster A & B.

by the 13B model are greater than those of the 7B model. This is because, in the 13B setting, two ranks within a TP group share the same NIC on Cluster A, whereas in the 7B setting (TP size 1), the workload is handled by a single rank. Consequently, bottleneck communication is eliminated with better speedups.

For MoE models, flop cannot be accurately estimated prior to routing, which undermines Hybrid DP’s flop-based token assignment and often leads to imbalanced expert computation. At shorter context length (e.g., 64k), where expert computation dominates, the balanced LLaMA CP method achieves the highest throughput. However, as context length grows, the primary bottleneck shifts to attention computation. In this regime, *Zeppelin*’s attention optimizations dominate, yielding superior performance.

5.2 Scalability

We evaluate the scalability of *Zeppelin* across different training scales and datasets, and also test its performance under a different GPU-NIC affinity architecture on Cluster B.

Experiments are conducted with the LLaMA 3B dense model on cluster A, using a context length of 4k tokens per GPU. As shown in Fig. 9, the TE baseline maintains nearly constant throughput across scales due to the cross-node communication bottleneck of ring attention. LLaMA CP achieves better performance by leveraging optimized all-gather collectives, but its communication overhead grows linearly with total sequence length. Hybrid DP reduces communication

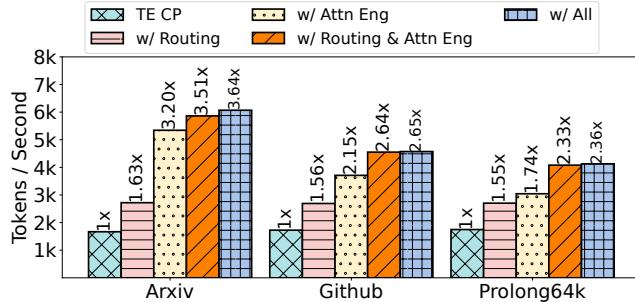


Figure 11. Ablation study on *Zeppelin* component performance. Comparison of TE CP baseline versus configurations with Routing Layer, Attention Engine, and Remapping Layer on a 3B model (32 GPUs, Cluster A) across three datasets.

costs using coarse-grained model-level parallelism, yet fails to capture the scaling characteristics of individual components. Consequently, it fails to outperform LLaMA CP even at small scales (16–32 GPUs), where group size remains limited. In contrast, *Zeppelin* incorporates per-sequence parallelism and leverages the hierarchical bandwidth structure of the hardware, yielding significantly better scalability across all settings.

For the ArXiv dataset, which exhibits a balanced sequence length distribution at 64k, performance scales smoothly across all methods. In contrast, the GitHub dataset contains long sequences exceeding 64k tokens. As context length grows, such sequences become more prevalent, leading to slower throughput gains for Hybrid DP and *Zeppelin* compared to balanced datasets. For both GitHub and ProLong64k, Hybrid DP’s FLOP-balanced metric is particularly sensitive to the dominance of long sequences: they occupy most ranks and force short sequences to be split into additional micro-batches. By contrast, *Zeppelin* mitigates this bottleneck through its fine-grained, per-sequence execution and hierarchical partitioning strategy.

We further evaluate performance on Cluster B, which features a different GPU-NIC affinity architecture and stronger GPU processing capabilities. As shown in Fig. 10, *Zeppelin* achieves significant throughput improvements on Cluster B over all baselines. The overall throughput is higher than that of Cluster A, primarily due to the enhanced computational power of the Hopper architecture. Both clusters exhibit similar speedup trends across datasets. However, the relative speedup of *Zeppelin* is greater on Cluster A. This is because Cluster A has a larger computation-to-communication ratio, allowing *Zeppelin* to more effectively overlap communication costs, consistent with its design principles.

5.3 Ablation Study

We evaluate the efficacy of key design components in *Zeppelin* by comparing TE CP with ablated variants of *Zeppelin* across three datasets, as shown in Fig. 11. Integrating the Communication Routing Layer into the TE CP baseline enables full utilization of all NICs in a global ring. Since the

total communication volume remains proportional to token count, the routing layer alone provides a consistent $\sim 1.6\times$ speedup across datasets. Adding efficient sequence partitioning and the attention engine, which coordinate the execution of local, intra-node, and inter-node queues, further reduces communication volume. On balanced datasets such as ArXiv, this yields up to $3.2\times$ speedup. Combining routing with the attention engine amplifies these gains by simultaneously reducing communication volume and improving NIC utilization. Building on attention-level partitioning, introducing the Remapping Layer balances workloads for linear modules. On right-skewed datasets such as ArXiv, remapping increases speedup from $3.51\times$ to $3.64\times$. In contrast, on long-sequence-dominated datasets such as GitHub, the incremental benefit of remapping is minimal, since attention computation dominates the cost.

5.4 Case Study

5.4.1 Timeline Analysis We profile three execution traces of a 3B model on 16 GPUs with a total sequence length of 64k on Cluster A in Fig. 12. These traces illustrate how *Zeppelin* improves performance and reveal potential overheads in its implementation. In the TE CP baseline, each round of ring attention overlaps attention computation of local Query activations and inter-node communication of KV activations for the next round. However, limited inter-node bandwidth makes inter-node communication the dominant contributor to attention overhead.

With communication routing in *Zeppelin* (Fig. 12.b), inter-node communication is decomposed into three steps. The local dispatch step overlaps directly with attention computation, but the inter-node transfer phase often stalls while waiting for Streaming Multiprocessor (SM) resources, since attention kernels dominate compute capacity. This creates bubbles—idle periods between communication kernels in the forward phase. Despite these bubbles, inter-node communication cost is reduced substantially from 2.18 ms to 411 μ s (a reduction proportional to NIC count). The forward phase communication cost is nearly halved, decreasing from 2.18 ms to 1.3 ms. In the backward phase, both computation and communication roughly double in duration, but overlap improves.

When the total context consists of multiple small sequences (Fig. 12.c), inter-node communication is avoided by assigning sequences to separate node buckets. Intra-node communication overlaps more effectively with computation, as observed in the timeline of Rank 0. Ranks 8–11 handle both intra-node and local sequences, with backward computation executed in reverse order. For small intra-node communications, performance fluctuations appear at Rank 8, in contrast to the stable but heavier communication observed in TE CP. Nonetheless, the per-round cost is dramatically reduced from $16 \times (2.18 \text{ ms} + 4.41 \text{ ms}) = 105.44 \text{ ms}$ in TE CP to $8 \times (738 \mu\text{s} + 1.95 \text{ ms}) = 21.504 \text{ ms}$ in *Zeppelin*.

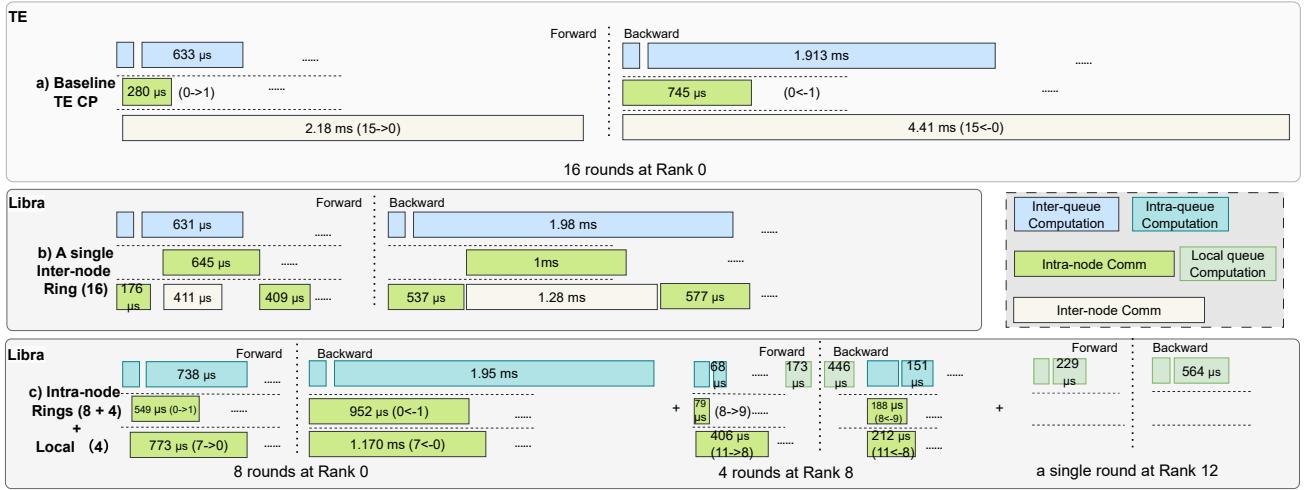


Figure 12. Forward and backward phase timelines of the attention component in a 3B model on 16 GPUs with a total context length of 64k tokens. a) TE CP: For a single sequence of length 64k, cross-node communication between rank 0 and rank 15 becomes a bottleneck during each round of ring attention. b) *Zeppelin* (single sequence): For a 64k-length sequence, the cross-node communication workload is split into eight chunks, enabling local dispatch, inter-node transfers, and local combination in each ring attention round. c) *Zeppelin* (multiple sequences): For multiple sequences with a total length of 64k, sequences are distributed to separate nodes without partitioning across nodes. Within each node, sequences are further partitioned or placed with balanced computation, improving parallel efficiency.

Table 3. Cost distribution under two length distribution

Components (ms)	Balanced	Skewed
Forward	316 - 817	1000 - 1002
Forward Quadratic Attention	161 - 670	801 - 854
Forward Linear Modules	93 - 97	91 - 105
Forward Remapping Layer	14 - 16	14 - 45
Forward Sequence Partition	3 - 10	3 - 12
Backward	422 - 1108	1605 - 1608

5.4.2 Length Distribution We evaluate the cost distribution across ranks under two different input length distributions, as shown in Table 3. Experiments are conducted on the 7B model with four nodes of Cluster C and a total context length of 128K. The Balanced distribution samples sequences from each bucket in Table 2, while the Skewed distribution contains one very long sequence and several short ones. End-to-end overhead under the Skewed distribution is higher than under the Balanced distribution, as the long sequence dominates attention computation. By contrast, remapping overheads are negligible, since our partitioning algorithm already accounts for token balance. The complexity of sequence partitioning is polynomial and incurred only once per iteration, making it negligible compared to the end-to-end training cost.

6 Related Work

Training LLMs efficiently at scale and enabling long-context capability have driven the development of advanced distributed learning techniques. Our work builds upon and

extends prior research in parallelization strategies, long-sequence processing, and communication optimization.

Basic Parallelisms in LLM Training In Data Parallelism [45], multiple workers train identical model replicas on different data subsets. ZeRO [44] reduces memory redundancy in DP. For models too large to fit on a single device, Tensor Parallelism (TP) [29, 47] shards weights within layers, while Pipeline Parallelism (PP) [8, 24, 35, 43] partitions layers across devices. Hybrid approaches [6, 27, 57] combine these techniques to scale to extremely large models.

Sequence Length Scaling The growing demand for long-context processing stresses traditional parallelisms due to the quadratic cost of self-attention mechanism. FlashAttention [10] reduces memory I/O overheads for single-device attention. For distributed training, multiple schemes have been proposed: DeepSpeed Ulysses [26] utilizes all-to-all operations to switch between sequence- and head-wise partitions; WLB-LLM [53] allgathers KV cache before local attention; Ring Attention [32] adopts ring-based communication to distribute attention; Striped Attention [4] and DIST-FLASHATTN [31] optimize load balance under causal masks; USP [16] integrate DeepSpeed-Ulysses and Ring Attention for better scalability; and LoongTrain [23] applies a double ring algorithm to reduce communication overhead.

Variable-length Training Fixed-length padding is inefficient as it wasted computation and memory. Packing variable-length sequences [42] reduces waste but leads to complex attention masks and load imbalance in SP approaches. Transformer Engine [39] balance computation by

evenly splitting sequences. FlexSP [52] adapts grouping and parallelism dynamically. HotSpa [20] mitigates skewness via hot switching among parallel strategies. ByteScale [19] reduces redundant communication for short sequences using hybrid data parallel approaches.

7 Conclusion

In this paper, we proposed *Zeppelin*, a novel system design engineered to address the severe imbalance issues encountered during data-parallel training of LLMs with long and variable-length data. *Zeppelin* enhances training performance through three key components: an efficient attention engine, a flexible communication routing layer, and a remapping layer. Comprehensive evaluations demonstrate that *Zeppelin* achieves an impressive average speedup of $2.80\times$ over state-of-the-art methods.

References

- [1] 2025. GUROBI. <https://docs.gurobi.com/projects/optimizer/en/current/index.html>
- [2] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshitijee Kalambarakar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathew, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) (*ASPLOS '24*). Association for Computing Machinery, New York, NY, USA, 929–947. doi:10.1145/3620665.3640366
- [3] Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R. Gormley. 2023. Unlimiformer: Long-Range Transformers with Unlimited Length Input. In *Thirty-seventh Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=IJWUJWLCo>
- [4] William Brandon, Aniruddha Nrusimha, Kevin Qian, Zachary Ankner, Tian Jin, Zhiye Song, and Jonathan Ragan-Kelley. 2023. Striped Attention: Faster Ring Attention for Causal Transformers. arXiv:2311.09431 [cs.LG] <https://arxiv.org/abs/2311.09431>
- [5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020). arXiv:2005.14165 <https://arxiv.org/abs/2005.14165>
- [6] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. 2024. 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* (La Jolla, CA, USA) (*ASPLOS '24*). Association for Computing Machinery, New York, NY, USA, 178–191. doi:10.1145/3620666.3651379
- [7] Mayee F. Chen, Michael Y. Hu, Nicholas Lourie, Kyunghyun Cho, and Christopher Ré. 2025. Aioli: A Unified Optimization Framework for Language Model Data Mixing. arXiv:2411.05735 [cs.LG] <https://arxiv.org/abs/2411.05735>
- [8] Tiancheng Chen, Ales Kubicek, Langwen Huang, and Torsten Hoefer. 2025. CrossPipe: Towards Optimal Pipeline Schedules for Cross-Datacenter Training. arXiv:2507.00217 [cs.DC] <https://arxiv.org/abs/2507.00217>
- [9] Yukang Chen, Fuzhao Xue, Dacheng Li, Qinghao Hu, Ligeng Zhu, Xiuyu Li, Yunhao Fang, Haotian Tang, Shang Yang, Zhijian Liu, Yihui He, Hongxu Yin, Pavlo Molchanov, Jan Kautz, Linxi Fan, Yuke Zhu, Yao Lu, and Song Han. 2025. LongVILA: Scaling Long-Context Visual Language Models for Long Videos. In *The Thirteenth International Conference on Learning Representations*. <https://openreview.net/forum?id=wCXAlfvCy6>
- [10] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv:2205.14135 [cs.LG] <https://arxiv.org/abs/2205.14135>
- [11] DeepSeek-AI. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. arXiv:2405.04434 [cs.CL] <https://arxiv.org/abs/2405.04434>
- [12] DeepSeek-AI. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] <https://arxiv.org/abs/2412.19437>
- [13] Mostafa Dehghani, Basil Mustafa, Josip Djolonga, Jonathan Heek, Matthias Minderer, Mathilde Caron, Andreas Steiner, Joan Puigcerver, Robert Geirhos, Ibrahim M Alabdulmohsin, Avital Oliver, Piotr Padlewski, Alexey Gritsenko, Mario Lucic, and Neil Houlsby. 2023. Patch n' Pack: NaViT, a Vision Transformer for any Aspect Ratio and Resolution. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 2252–2274. https://proceedings.neurips.cc/paper_files/paper/2023/file/06ea400b9b7fcce6428ec27a371632eb-Paper-Conference.pdf
- [14] Hantian Ding, Zijian Wang, Giovanni Paolini, Varun Kumar, Anoop Deoras, Dan Roth, and Stefano Soatto. 2024. Fewer Truncations Improve Language Modeling. arXiv:2404.10830 [cs.CL] <https://arxiv.org/abs/2404.10830>
- [15] Mahidhar Dwarampudi and N V Subba Reddy. 2019. Effects of padding on LSTMs and CNNs. arXiv:1903.07288 [cs.LG] <https://arxiv.org/abs/1903.07288>
- [16] Jiarui Fang and Shangchun Zhao. 2024. USP: A Unified Sequence Parallelism Approach for Long Context Generative AI. arXiv:2405.07719 [cs.LG] <https://arxiv.org/abs/2405.07719>
- [17] Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. 2024. Data engineering for scaling language models to 128K context. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) (*ICML '24*). JMLR.org, Article 564, 10 pages.
- [18] Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. 2025. How to Train Long-Context Language Models (Effectively). arXiv:2410.02660 [cs.CL] <https://arxiv.org/abs/2410.02660>
- [19] Hao Ge, Junda Feng, Qi Huang, Fangcheng Fu, Xiaonan Nie, Lei Zuo, Haibin Lin, Bin Cui, and Xin Liu. 2025. ByteScale: Efficient Scaling of LLM Training with a 2048K Context Length on More Than 12,000 GPUs. arXiv:2502.21231 [cs.DC] <https://arxiv.org/abs/2502.21231>
- [20] Hao Ge, Fangcheng Fu, Haoyang Li, Xuyan Wang, Sheng Lin, Yujie Wang, Xiaonan Nie, Hailin Zhang, Xupeng Miao, and Bin Cui. 2024. Enabling Parallelism Hot Switching for Efficient Training of Large Language Models. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles* (Austin, TX, USA) (*SOSP '24*). Association for Computing Machinery, New York, NY, USA, 178–194.

- doi:10.1145/3694715.3695969
- [21] Google. 2025. Gemini 2.5: Our most intelligent AI model. <https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking>
- [22] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2018. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677 [cs.CV] <https://arxiv.org/abs/1706.02677>
- [23] Diandian Gu, Peng Sun, Qinghao Hu, Ting Huang, Xun Chen, Yingtong Xiong, Guoteng Wang, Qiaoling Chen, Shangchun Zhao, Jiarui Fang, Yonggang Wen, Tianwei Zhang, Xin Jin, and Xuanzhe Liu. 2024. Loong-Train: Efficient Training of Long-Sequence LLMs with Head-Context Parallelism. arXiv:2406.18485 [cs.DC] <https://arxiv.org/abs/2406.18485>
- [24] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. PipeDream: Fast and Efficient Pipeline Parallel DNN Training. arXiv:1806.03377 [cs.DC] <https://arxiv.org/abs/1806.03377>
- [25] huggingface. 2025. Padding and truncation. https://huggingface.co/docs/transformers/pad_truncation
- [26] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Reza Yazdani Aminadabi, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. 2024. System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing* (Nantes, France) (PODC '24). Association for Computing Machinery, New York, NY, USA, 121–130. doi:10.1145/3662158.3662806
- [27] Ziheng Jiang, Haibin Lin, Yimin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. 2024. MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs. arXiv:2402.15627 [cs.LG] <https://arxiv.org/abs/2402.15627>
- [28] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2022. The Stack: 3 TB of permissively licensed source code. arXiv:2211.15533 [cs.CL] <https://arxiv.org/abs/2211.15533>
- [29] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2022. Reducing Activation Recomputation in Large Transformer Models. arXiv:2205.05198 [cs.LG] <https://arxiv.org/abs/2205.05198>
- [30] Mosh Levy, Alon Jacoby, and Yoav Goldberg. 2024. Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikanth (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 15339–15353. doi:10.18653/v1/2024.acl-long.818
- [31] Dacheng Li, Rulin Shao, Anze Xie, Eric P. Xing, Xuezhe Ma, Ion Stoica, Joseph E. Gonzalez, and Hao Zhang. 2024. DISTFLASHATTN: Distributed Memory-efficient Attention for Long-context LLMs Training. arXiv:2310.03294 [cs.LG] <https://arxiv.org/abs/2310.03294>
- [32] Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring Attention with Blockwise Transformers for Near-Infinite Context. arXiv:2310.01889 [cs.CL] <https://arxiv.org/abs/2310.01889>
- [33] Meta. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] <https://arxiv.org/abs/2307.09288>
- [34] Meta. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] <https://arxiv.org/abs/2407.21783>
- [35] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. 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* (St. Louis, Missouri) (SC '21). Association for Computing Machinery, New York, NY, USA, Article 58, 15 pages. doi:10.1145/3458817.3476209
- [36] NVIDIA. 2022. NVIDIA Collective Communication Library (NCCL) Documentation. <https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/index.html>
- [37] NVIDIA. 2025. DGX A100 System Topology. <https://docs.nvidia.com/dgx/dgxa100-user-guide/introduction-to-dgxa100.html>
- [38] NVIDIA. 2025. DGX H100/200 System Topology. <https://docs.nvidia.com/dgx/dgxh100-user-guide/>
- [39] NVIDIA. 2025. TransformerEngine. <https://github.com/NVIDIA/TransformerEngine>
- [40] Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2023. OpenWebMath: An Open Dataset of High-Quality Mathematical Web Text. arXiv:2310.06786 [cs.AI] <https://arxiv.org/abs/2310.06786>
- [41] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. arXiv:2406.17557 [cs.CL] <https://arxiv.org/abs/2406.17557>
- [42] Hadi Pouransari, Chun-Liang Li, Jen-Hao Rick Chang, Pavan Kumar Anasolu Vasu, Cem Koc, Vaishaal Shankar, and Oncel Tuzel. 2024. Dataset Decomposition: Faster LLM Training with Variable Sequence Length Curriculum. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 36121–36147. https://proceedings.neurips.cc/paper_files/paper/2024/file/3f9bf45ea04c98ad7cb857f951f499e2-Paper-Conference.pdf
- [43] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. 2023. Zero Bubble Pipeline Parallelism. arXiv:2401.10241 [cs.DC] <https://arxiv.org/abs/2401.10241>
- [44] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. arXiv:1910.02054 [cs.LG] <https://arxiv.org/abs/1910.02054>
- [45] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) (KDD '20). Association for Computing Machinery, New York, NY, USA, 3505–3506. doi:10.1145/3394486.3406703
- [46] Zhiqiang Shen, Tianhua Tao, Liquan Ma, Willie Neiswanger, Zhengzhong Liu, Hongyi Wang, Bowen Tan, Joel Hestness, Natalia Vassilieva, Daria Soboleva, and Eric Xing. 2024. SlimPajama-DC: Understanding Data Combinations for LLM Training. arXiv:2309.10818 [cs.CL] <https://arxiv.org/abs/2309.10818>
- [47] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *CoRR* abs/1909.08053 (2019). arXiv:1909.08053 <http://arxiv.org/abs/1909.08053>
- [48] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. 2018. Don't Decay the Learning Rate, Increase the Batch Size. arXiv:1711.00489 [cs.LG] <https://arxiv.org/abs/1711.00489>
- [49] Qwen Team. 2024. Qwen2 Technical Report. arXiv:2407.10671 [cs.CL] <https://arxiv.org/abs/2407.10671>
- [50] Qwen Team. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] <https://arxiv.org/abs/2412.15115>
- [51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017.

- Attention is All you Need. In *Advances in Neural Information Processing Systems*. I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- [52] Yujie Wang, Shiju Wang, Shenhan Zhu, Fangcheng Fu, Xinyi Liu, Xuefeng Xiao, Huixia Li, Jiashi Li, Faming Wu, and Bin Cui. 2025. FlexSP: Accelerating Large Language Model Training via Flexible Sequence Parallelism. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Rotterdam, Netherlands) (ASPLOS '25). Association for Computing Machinery, New York, NY, USA, 421–436. doi:10.1145/3676641.3715998
- [53] Zheng Wang, Anna Cai, Xinfeng Xie, Zaifeng Pan, Yue Guan, Weiwei Chu, Jie Wang, Shikai Li, Jianyu Huang, Chris Cai, Yuchen Hao, and Yufei Ding. 2025. WLB-LLM: Workload-Balanced 4D Parallelism for Large Language Model Training. arXiv:2503.17924 [cs.DC] <https://arxiv.org/abs/2503.17924>
- [54] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bharava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, Madiam Khabsa, Han Fang, Yashar Mehdad, Sharan Narang, Kshitiz Malik, Angela Fan, Shruti Bhosale, Sergey Edunov, Mike Lewis, Sinong Wang, and Hao Ma. 2024. Effective Long-Context Scaling of Foundation Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 4643–4663. doi:10.18653/v1/2024.nacl-long.260
- [55] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Large Batch Training of Convolutional Networks. arXiv:1708.03888 [cs.CV] <https://arxiv.org/abs/1708.03888>
- [56] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Sri-nadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. arXiv:1904.00962 [cs.LG] <https://arxiv.org/abs/1904.00962>
- [57] Tailing Yuan, Yuliang Liu, Xucheng Ye, Shenglong Zhang, Jianchao Tan, Bin Chen, Chengru Song, and Di Zhang. 2024. Accelerating the Training of Large Language Models using Efficient Activation Rematerialization and Optimal Hybrid Parallelism. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 545–561. <https://www.usenix.org/conference/atc24/presentation/yuan>
- [58] Liang Zhao, Tianwen Wei, Liang Zeng, Cheng Cheng, Liu Yang, Peng Cheng, Lijie Wang, Chenxia Li, Xuejie Wu, Bo Zhu, Yimeng Gan, Rui Hu, Shuicheng Yan, Han Fang, and Yahui Zhou. 2024. LongSkywork: A Training Recipe for Efficiently Extending Context Length in Large Language Models. arXiv:2406.00605 [cs.CL] <https://arxiv.org/abs/2406.00605>