WeiPipe: Weight Pipeline Parallelism for Communication-Effective Long-Context Large Model Training

Junfeng Lin[*]
Tsinghua University
Beijing, China
linjf21@mails.tsinghua.edu.cn

Ziming Liu[*]
National University of Singapore
Singapore
liuziming@comp.nus.edu.sg

Yang You
National University of Singapore
Singapore
youy@comp.nus.edu.sg

Jun Wang
CETHIK Group Co. Ltd.
Hangzhou, China
alfiede@163.com

Weihao Zhang[†]
Lynxi Technologies Co. Ltd
Beijing, China
weihao.zhang@lynxi.com

Rong Zhao[†]
Tsinghua University
Beijing, China
r_zhao@tsinghua.edu.cn

Abstract

Training large language models (LLMs) has become increasingly expensive due to the rapid expansion in model size. Pipeline parallelism is a widely used distributed training technique. However, as LLMs with larger context become prevalent and memory optimization techniques advance, traditional PP methods encounter greater communication challenges due to the increased size of activations and gradients of activations. To address this issue, we introduce weight-pipeline parallelism (WeiPipe) that transitions from an activation-passing pipeline to a weight-passing pipeline. WeiPipe reduces communication costs and achieves a more balanced utilization by transmitting only weights and their gradients between workers in a pipeline manner. WeiPipe does not rely on collective communication primitives, thus ensuring scalability. We present four variations of WeiPipe parallelism, including WeiPipe-Interleave, which emphasizes communication efficiency, and WeiPipe-zero-bubble, discussing the potential for minimal bubble ratios. Our implementation of WeiPipe-Interleave, performed on up to 32 GPUs and tested in various model configurations, including large-context LLM training, demonstrates a significant improvement in throughput compared to state-of-the-art pipeline parallelism and Fully Sharded Data Parallelism with different underlying infrastructures, including NVLink connections within cluster with Ethernet among cluster, and PCIe within cluster and Ethernet among cluster. Additionally, WeiPipe also shows greater scalability in communication-constrained scenarios compared to former pipeline methods.

CCS Concepts: • **Theory of computation** → **Parallel algorithms**; • **Computing methodologies** → *Neural networks.*

Keywords: distributed deep learning, pipeline parallelism, large-scale training, long-context training

1 Introduction

Advancements in LLMs with Transformer[38] architectures have revolutionized generative AI. The scaling law[19] reveals a positive correlation between model performance and parameter capacity. However, as these models grow, their computational resource requirements for training increase significantly. Diverse distributed training techniques are designed to optimize efficiency training across a large number of interconnected GPU workers, such as data parallelism (DP), tensor parallelism (TP)[5, 23, 35, 39, 40], pipeline parallelism (PP)[12, 15, 17, 22, 26, 32], and Sequence Parallelism (SP)[14, 16, 24, 25, 27]. DP and SP duplicate the model across multiple workers. TP splits matrix operations within a layer among workers, whereas PP segments the layers into stages processed by different workers. Compared to other strategies, PP reduces per worker's memory needs, requires less bandwidth, and only needs peer-to-peer (P2P) communication.

As the demand for long-context large language models (LLMs) grows rapidly, training these models with extended token length leads to a significant increase in activation values between layers. Moreover, memory optimization techniques such as mix-precision training[28], recomputation (gradient checkpointing), and flash attention are becoming the standard options for LLM training. These techniques reduce the peak memory occupation and enable large micro-batch sizes for PP. Both of these increase the communication burden of PP due to the larger volumes of activation and

[*]Both authors contributed equally to this research.
[†]Corresponding authors.

PPoPP '25, March 1–5, 2025, Las Vegas, NV, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1443-6/25/03
https://doi.org/10.1145/3710848.3710869

gradient values. Specifically, the output activation size of a single transformer stage in PP is about GSH (counted by number), where G is micro-batch size, S is the sequence length and H is the hidden dimension size in Transformer. In contrast, the weight size for one layer in a model like Llama-2 is about $12H^2$, including $4H^2$ for the attention module and $8H^2$ for the feed-forward network (FFN). When the ratio of output activation to weight, $\frac{GS}{12H}$, exceeds 1, the weight-passing method can be more communication-efficient than the activation-passing method.

Inspired by this observation, we propose the weight-pipeline parallelism (WeiPipe) to address the challenges posed by long-context models and larger micro-batch sizes. In conventional activation-passing pipelines, one worker holds the activations of multiple micro-batches and the weights of one stage, passing activations (As) and gradients of activations (Bs) to the next worker. In contrast, a worker in WeiPipe maintains the activations for all layers of a single micro-batch, passing weights (Ws) and gradients of weights (Ds) of one stage to the subsequent worker. Unlike DP and TP, activation-passing pipelines and WeiPipe rely solely on P2P communication to achieve high scalability. Except for optimizing pipeline bubbles, which is the primary focus of conventional PP strategies, WeiPipe additionally emphasizes the arrangement of data to ensure smooth flow of Ws and Ds between workers with minimal overhead. The contributions of this paper are as follows:

- We begin by introducing the basic WeiPipe strategy to demonstrate the fundamental concepts and mechanisms of the weight-passing pipeline.
- Next, we propose WeiPipe-Interleave that enhances the naive WeiPipe strategy by interleaving the forward and backward passes, which reduces the bubble ratio and halves the communication requirements compared to the naive approach.
- We investigate the potential of integrating WeiPipe with zero-bubble parallelism, demonstrating that a weight-passing pipeline can achieve near-zero bubble ratios.
- We implement WeiPipe-Interleave on PyTorch from scratch and apply WeiPipe-Interleave to mainstream LLama-style[37] models (also GPT-like). Experiments show that WeiPipe can improve training efficiency by about 30%-80% compared to state-of-the-art PP and maintain weak and strong scalability in long-context scenarios.

To enable a more comprehensive analysis of state-of-the-art methods and WeiPipe, we have standardized the symbolic representation, which is presented in Table 1.

N	The number of micro-batches in an iteration
$Iter$	The number of iteration
G	Micro-batch size
P	The number of workers
L	The number of layers in neural network
A_j^i	Activation values of jth layer in ith micro-batch
B_j^i	Gradients of A_j^i
W_j	Weights of jth layer
D_j	Gradients of W_j
$M_{A/B/W/D}$	Memory consumption of A, B, W or D
$T_{F/B/W}$	Time cost for complete forward pass, B pass or W pass.
T_{BW}	$T_B + T_W$
α	The percentage of left memory consumption of activations after B pass.
H	The size of hidden dim in Transformer
S	The sequence length in Transformer

Table 1. Meaning of the symbols that are used in this paper.

2 Background and Related Work

2.1 Large Models and Long Context Training

Recently, Transformer-based models [4, 9, 10, 30, 36, 38] have become foundational in multiple fields, such as natural language processing (NLP) and computer vision (CV). As researchers strive for better performance, the scale of these models has increased significantly in two major dimensions. First, the size of the models has grown substantially. For example, the recent Llama-3.1 model [11] boasts an impressive 405 billion parameters, necessitating at least 6480 GB of GPU memory to accommodate the model parameters. This disparity presents a significant challenge in distributing such large models across a vast number of GPUs, with up to 16,000 NVIDIA H100 required in the case of Llama-3.1 [11].

Second, the activation size during training has also increased due to longer sequence lengths. In NLP, long-context capabilities are crucial for tasks such as multi-round conversations[1], summarization[2, 20], and processing large codebases[7, 21, 34]. In CV, generating high-resolution and long-duration videos[3, 31] results in sequences that expand both temporally and spatially. Furthermore, in scientific AI, models like AlphaFold [6, 18] must manage long sequences of amino acids. These long-context scenarios introduce new challenges in designing parallelism schemes that achieve efficient and balanced memory management.

2.2 Parallelism Technique for Training

To address the above challenges, various parallelization techniques have been developed. DP is the most straightforward approach, dividing the data among devices and using all-reduce operations at each iteration to synchronize gradients.

Figure 1. WeiPipe-Naive strategy. The text below workers denotes the retained data in memory. The text within the circle represents data counterclockwise circulating among workers. At t, each worker has the data at its position and the other side of the orange line. For instance, at $t = 5$, worker 1 has W_0, W_3, D_3 and receives W_1, W_2, D_2 from worker 0. At $t = 6$, worker 1 holds W_1, W_2, D_2. Green blocks indicate the worker is performing the forward pass and blue blocks indicate the backward pass.

Enhanced versions, such as fully sharded data parallelism (FSDP) [41] and zero redundancy optimizer(ZeRO)[33], further optimize by splitting model weights and optimizer states. TP divides tensors into chunks along specific dimensions, allowing each device to hold only a portion of the tensor without compromising the integrity of the computation graph. This technique requires frequent collective communication to synchronize the results. PP partitions the model at the layer level and divides the batch into micro-batches. Traditional PP assigns layers across multiple devices and uses P2P communication to pass activations between devices. Furthermore, SP [14, 16, 24, 25, 27] divides activations along the sequence dimension. Among these parallelization strategies, DP is widely used for its ease of use and PP is favored for its reliance on P2P communication, making it a viable option for devices with less robust connections. To better understand WeiPipe, we will first explore these parallelization techniques in detail.

2.2.1 Pipeline Parallelism. In this paper, we focus primarily on synchronous pipelines, which maintain the convergence of the training. One of the most known PP techniques is GPipe [15], which completes all forward propagation for all micro-batches before starting the backward propagation. Dapple [12], also known as 1F1B, enhances this approach by initiating backward propagation of a micro-batch immediately after its forward finishes, thereby reducing peak memory on certain devices. Chimera [22] further optimizes by combining multiple pipelines in different directions to reduce the bubble ratio. Hanayo [26] builds on this by utilizing a wave-shaped pipeline to decouple Chimera from model duplication, thereby improving efficiency. Zero-bubble PP [32] computes gradients for the weights and activations separately, allowing for a more flexible pipeline arrangement. This method offers two schemes: one minimizes peak memory, and the other reduces the bubble ratio. Further discussion of these approaches can be found in Section 4. Although current PPs have largely focused on reducing the bubble ratio, they face significant challenges in long-context scenarios where the overhead of activation and gradient communication can exceed the computational load of a single pipeline stage.

2.2.2 Fully Sharded Data Parallelism. FSDP has similar functions with ZeRO stage 3 (ZeRO-3), sharding the data, model weights and optimizer states on different GPUs. In FSDP, weights and gradients are only gathered when necessary for computation and are promptly discarded after the computation is completed. This sharding and unsharding process is facilitated by all-gather and reduce-scatter operations. FSDP also incorporates asynchronous communication to enhance overlap. Despite its ease of use and memory efficiency, FSDP requires a substantial amount of collective communication during both forward and backward propagation, which could be challenging in scaled scenarios.

In a word, existing techniques are not well equipped to reduce the communication overhead in long-context scenarios, particularly in clusters with less robust network connections.

3 WeiPipe Framework

3.1 WeiPipe-Naive

To introduce the basic idea of WeiPipe, we focus on the commonly used ring topology PP. The initial step is to evenly distribute the model layers across all workers along the ring. Assume that the number of layers L is equal to the number of workers P and each worker holds the weight of one layer. The training procedure consists of three passes: the forward pass, the backward pass, and the update pass. During the forward pass, it is straightforward to achieve a weight-passing pipeline. Worker 0, holding the weights W_0 of the first layer, initiates the forward computation. Simultaneously, it passes W_0 to worker 1 and receives W_1 from worker $P-1$. Subsequently, worker 0 discards W_0 but retains the activations of the first layer A_0 in its memory and begins forwarding the second layer using W_1. Meanwhile, worker 1 computes the first layer with a new micro-batch input. This process continues until worker 0 completes the forward passes of all layers. Figure 1 illustrates this procedure from $t = 0$ to $t = 3$ with $P = 4$. In Figure 1, we use a circular representation to illustrate the arrangement and transmission of data in WeiPipe, which is particularly helpful for understanding the following complex strategies. For $P = 4$, the circle is divided into 8 positions, each filled with Ws or Ds. Each worker holds data in its current position and in the diagonal position (indicated by the orange line across the circle). Initially, worker 0 through worker 3 each hold W_0 to W_3 respectively (the black text in the circle). The circle rotates counterclockwise, with each turn denoted as $t + 1$. After each rotation, the workers update the data they hold, reflecting the transition of data between workers. From $t - 0$ to 3, each worker processes the forward pass for its corresponding layer (the green block in the circle) and accumulates activations from A_0 to A_3 in their own memory, as shown below for each worker.

The backward pass becomes more complex because the weights are required in reverse order. To maintain the weight-pipelining, we fill the opposite side of the circle with W_3 to W_0 (the blue text in Figure 1 from $t = 0$ to 3). After Worker 0 completes the forward pass, the circle continues to rotate counterclockwise. Worker 0 then receives W_3 to W_0 one by one and executes the backward computation from layer 3 to layer 0 (blue blocks). In WeiPipe-Naive, workers with lower index finish earlier than those with higher index. So, a smaller-indexed worker starts to perform the backward pass, a larger-indexes worker might still be engaged in the forward pass. Thus, the weights for both forward and backward need to be transferred simultaneously. To facilitate this, each worker holds weights for both forward and backward. Figure 1 from $t = 4$ to 8 presents the backward pass of worker 0 and worker 0 can initiate another cycle for a new micro-batch after $t = 8$.

During the update pass, each worker utilizes the weight gradients (Ds) to update the weights. However, since weights cycle among workers and each worker just generates a piece of Ds for their micro-batch, updating the weights of a layer requires gathering all corresponding Ds. In DP, the all-reduce primitive is used to collect Ds. WeiPipe aims to use only P2P communications, so Ds also circulate through the ring after it is generated. Taking worker 0 in Figure 1 as an example, at $t = 4$, worker 0 performs the backward pass for layer 3 with W_3 and generates D_3. Then D_3 travels together with W_3 around the circle. Each worker that processes layer 3 backward will average, sum, or normalize the newly generated D_3 with the existing D_3 and keep the communication volume unchanged. This circular process repeats until all micro-batches are processed, at which point each worker updates Ws and Ds it holds using the specified optimizer. Since each worker updates one layer of weights, it also stores the corresponding layer of optimization state, which does not need to be transmitted between workers.

3.2 WeiPipe-Interleave for Lower Bubble Ratio

There are major flaws of WeiPipe-Naive. There are two weight-flow circulating simultaneously, but only one is used for the current computation. The redundant transmission of weights increases transmission costs. Most crucially, the computational time for the backward pass of a layer is approximately twice that of the forward pass. When one worker executes the backward pass, it significantly increases the pipeline bubble for other workers currently engaged in the forward pass. To address these issues, we interleave the forward and backward passes and get the WeiPipe-Interleave strategy.

The idea is to utilize the weights at the diagonal positions of the circle for another micro-batch forward or backward computation. The initial forward process is the same as that of WeiPipe-Naive. Figure 2 illustrates the interleaving forward and backward from $t = 4$ to 11. At $t = 4$, worker 0 begins to perform the backward pass. In the meantime, it utilizes the W_0 to perform the forward pass of a new micro-batch's input. Therefore, in this position, worker 0 will execute one backward and one forward before the circle makes another turn. During this time, other workers can only perform one forward operation and wait for worker 0's computation to finish. In the subsequent turn, workers 0 to 3 also enter the forward-backward interleave stage when the computation workloads are balanced between workers.

There is no pipeline bubble during the forward-backward interleave stage, no matter how different the workloads of backward and forward are. Each worker can also decide their own execution order for the forward and backward passes based on their data transmission situation. In addition, idle memory during forward and backward passes is utilized to store activation values generated by new micro-batches, as shown in Figure 1, resulting in a balanced storage utilization across different workers and times. WeiPipe-Interleave further reduces communication overhead. For the LLama figures/fig2.png

Figure 2. WeiPipe-Interleave strategy. A^i_j denotes the activation values of jth layer in ith micro-batch.

figures/fig3.png

Figure 3. WeiPipe-zero-bubble 1. Green block for the forward pass, blue block for the B pass, and purple block for the W pass. B^i_j denotes the gradients of activations of jth layer in ith micro-batch. The intermediate gradients between layers are ignored.

structure that has $12H^2$ parameters in one layer, during the forward-backward interleave stage, each worker receives two layers of weights and one layer of D in each turn. The communication volume is $36H^2$, which doubles the compute/commute ratio compared to WeiPipe-Naive.

3.3 WeiPipe-zero-bubble

To further explore the potential of WeiPipe, we present a design that combines WeiPipe with the zero-bubble pipeline. Zero-bubble strategies are state-of-the-art PP that split the backward computation into a B pass and a W pass to achieve almost zero-bubble. In the zero-bubble pipeline, the B pass computes gradients for activations, and the W pass computes gradients for weights. The B pass is performed in a back-propagation manner, while the W pass can be delayed to fill the blanks in the pipeline. Based on this decouple method, we introduce WeiPipe-zero-bubble 1(WZB1), which slightly reduces the bubble ratio compared to WeiPipe-Interleave with relatively low storage and communication overhead, and WeiPipe-zero-bubble 2 (WZB2), which achieves a nearly zero bubble configuration.

Figure 3 shows the WZB1 procedure from $t = 4$. Blue blocks represent B passes, while purple blocks denote W passes. At $t = 4$, worker 0 performs two tasks: forward

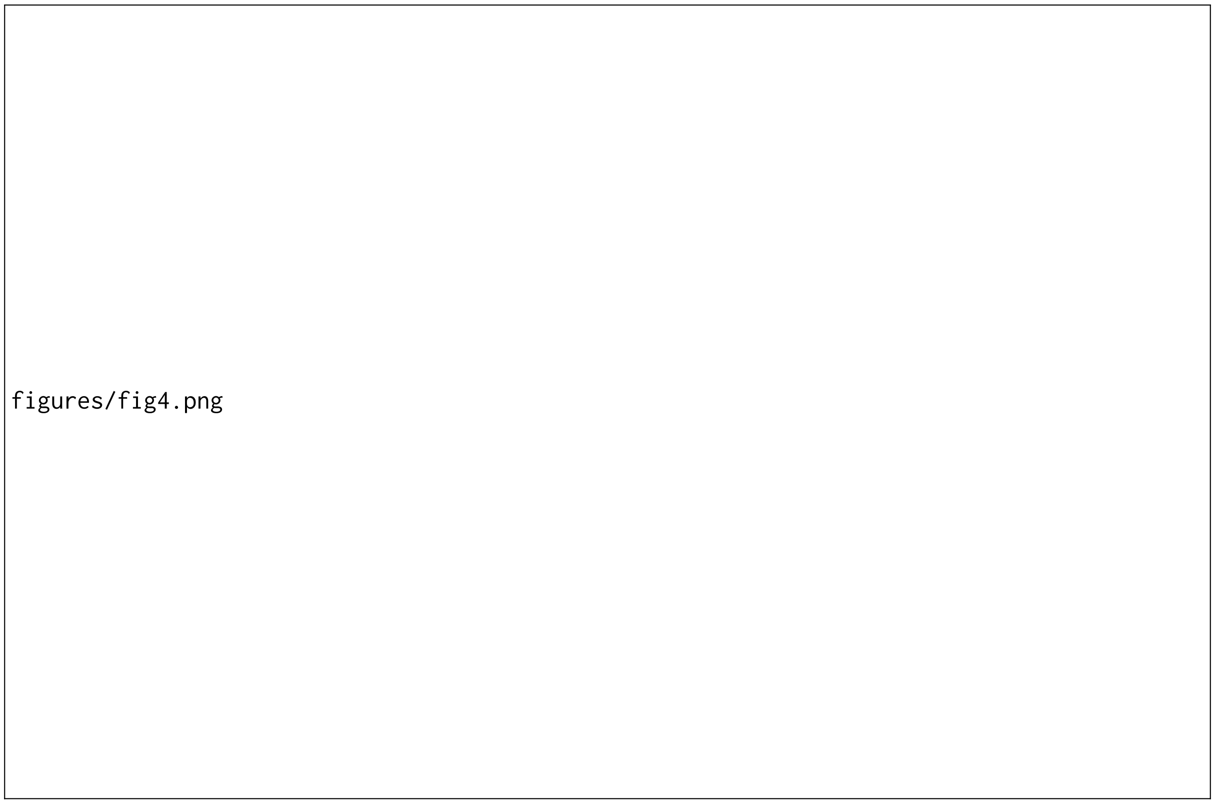

Figure 4. WeiPipe-zero-bubble 2. The worker 3 stages the updated weights at the end of each round and delivers these weights one by one to the worker 0 for the next round. This arrangement will not exceed the peak memory of worker 3.

of layer 0 on a new micro-batch (micro-batch 4) with W_0, generating A_0^4; the B pass of layer 3 with W_3 on the old micro-batch (micro-batch 0), producing the activation gradient B_3^0. Worker 0 still holds the portion of the A_3^0 after $t = 4$ to support the future W pass on W_3. Unlike WeiPipe-Interleave, W_1 for the backward pass is placed together with W_3 in the circle. The red text in Figure 3 indicates the data used by the worker at current. At $t = 5$, worker 0 performs the W pass, consuming A_3^0 and B_3^0 to generate D_3 in the circle, which will be sent to worker 1. Simultaneously, worker 0 also conducts the forward pass of micro-batch 4 with W_1. This process continues with alternating "one-forward-one-B" and "one-forward-one-W" until the forward pass of micro-batch 4 is completed. Then, at $t = 8$, worker 0 performs two B passes–one for micro-batch 0 with W_1 and one for micro-batch 4 with W_3. From $t = 8$ to 11, worker 0 alternates between two B passes and two W passes until $t = 12$ when all passes for micro-batch 0 are completed, and the forward pass for a new micro-batch (micro-batch 8) begins.

Figure 3 illustrates the arrangement of data in the ring. W_0 to W_3 for the forward pass are consistent with the previous strategy. The weights for the B pass and the gradients generated by the W pass are placed in pairs. Generally, for a network with L layers, W_{L-1} and $W_{\frac{L}{2}-1}$ are placed together, W_{L-2} and $W_{\frac{L}{2}-2}$ are placed together, and so forth. The same pairing applies to Ds. This arrangement ensures that each worker performs 2-chunk operations while transmitting 3 chunks of data to the next worker within one turn.

WZB2 offers a more simple procedure with the same arrangement of Ws and Ds as WeiPipe-Interleave, as shown in Figure 4. In WZB2, the forward, B, & W passes for all layers are executed sequentially within a worker. In particular, the W pass progresses from layer 0 to layer 3, matching the forward order. During the B pass, old versions of the weights can be discarded to reduce transmission, as indicated by the blanks from $t = 8$ to 11. The last worker, worker 3, aggregates all Ds and updates the weights. In the case in Figure 4, at $t = 11$, worker 3 holds the aggregated D_0 updates W_0 with the specified optimizer. Worker 3 then sends the updated W_0, which initiates a new forward pass at $t = 12$. This seamless handover allows for frequent weight updates with fewer micro-batches and achieves almost zero bubble. WZB2 incurs higher communication and storage costs, performing one chunk operation while transmitting two chunks of data to the next worker.

figures/fig5.png

Figure 5. The comparison of WeiPipe with other pipeline strategies. In the pipelining graph, the upper number in a block represents the layer index and the bottom number is the micro-batch index. The distribution graph in the equation table shows the theoretical bandwidth and memory usage of these strategies along devices and times. The specific value is calculated based on Llama-structure with $H = 4096$, $S = 4096$, $N = 12$, $G = 32$, and 32 heads Transformer. Memory distribution is estimated with $\alpha = 0.6$ based on statistics without recomputation and flash attention. The practical distribution could be distinct depending on implementation.

4 Theoretical Analysis

Figure 5 compares WeiPipe with other PP. WeiPipe-Interleave has the same bubble ratio as 1F1B. Both zero-bubble strategies reduce the bubble by multiplying a factor greater than 1 to N in the bubble ratio equation. ZB2 and WZB2 have almost no bubbles along the iteration.

Communication efficiency is assessed by dividing the total number of data received by the computation time (bandwidth usage). Different zones of a pipeline strategy have different communication densities. Figure 5 presents the equation for the theoretical bandwidth usage of Zone 1, where passes are fully alternated (Zone 0 for WZB2). For activation-passing PP, the equation is derived from middle workers transmitting both As and Bs (so, there is a "2" in the equation). Communication for activation-passing strategies increases linearly with micro-batch size G and sequence length S. In contrast, WeiPipe's communication is dictated by the amount of weights, independent of G and S. In this paper, LLama-style models are selected as standard models for analysis, where the number of weights per layer is $12H^2$. WeiPipe-Interleave requires transmitting two chunks of W and one chunk of D, resulting in a communication volume of $36H^2$ at each turn with a time duration of $\frac{T_F+T_W}{P}$, where T_F and T_F is the time for a complete forward and backward respectively.

Memory consumption is significantly affected by the specific implementations. Here, we provide a rough estimate of M_A, M_B, M_W, and M_D based solely on the counts of these values, while ignoring factors such as communication buffers, embedding layers, and mixed precision training. For 1F1B and WeiPipe-Interleave, the dominant part is the storage of activations, measured as GM_A. WeiPipe-Interleave has similar memory consumption as 1F1B but is more balanced. In strategies that separate the B pass and the W pass, one stage of the B pass ($\frac{1}{P}$ of a complete B pass) will consume part of the activations M_A/P stored by the forward pass and generate gradients M_B/P. We assume that $\alpha M_A/P$ activations storage is left after one stage of the B pass. For the zero-bubble pipeline, we assess the peak storage required by the last worker. ZB1 and WZB2 both need to store $G(\alpha M_A + M_B)$ data. ZB2 nearly doubles this requirement. However, WZB1 can achieve a maximum memory consumption of approximately $1.5GM_A$, which is less than other zero-bubble strategies when $\frac{M_B}{M_A} > 1.5 - \alpha$. Note that calculating the peak memory for zero-bubble 1 could be tricky. With smaller α and specific memory management strategies in GPU, worker 0 may have the highest memory consumption.

5 Implementation

Current distributed training frameworks offer various activation-passing pipeline strategies, but few accommodate high-level modifications to achieve weight pipelining. Thus, we have implemented WeiPipe-Interleave from scratch in PyTorch[29]. The implementation is tailored for training LLMs, but our approach is not limited to Transformers. Due to the need for intricate and fine-grained control, we present WeiPipe-zero-bubble to demonstrate the potential of a weight-passing pipeline and to facilitate a deeper discussion, while leaving the implementation for future exploration. The following details are considered for implementation:

Mixed Precision. Unlike other pipeline research focusing mainly on full precision implementation, our implementable incorporates widely adopted mixed precision training to further reduce storage and communication. Specifically, A, W, and D are in fp16 precision, while B is in bf16. The optimizer state is in fp32, distributed among the workers.

Communication overlap. WeiPipe meticulously adjusts the arrangement of computation and weight-passing to balance communication and computation workload. To ensure WeiPipe's high-performance potential, communication hiding optimizations are implemented. During forward-backward interleaving, Ws and Ds are prefetched using asynchronous communication, which is realized by the batch_isend_irecv function provided by PyTorch distributed library.

Recomputation and Flash Attention. Recomputation (gradient checkpointing/rematerialization) is a strategy that saves storage by abandoning part of the activation values during the forward pass. Instead, these values are recomputed during the backward pass, thus significantly reducing memory consumption with redundant computation. Similarly, Flash Attention[8] is a widely used technique to optimize memory access and save memory in attention modules. By employing these techniques, we can increase micro-batch size and enhance the strengths of WeiPipe. We also implement these optimizations across other strategies for a fair comparison.

6 Evaluation

6.1 Experimental Setup

We evaluate our implementation on models based on the open-source LLama-2 structure[37] (also GPT-style models). In our experiments, we fix the head number as 32 and the layer number L as 32. We change the hidden dimension size H, token length S, micro-batch size G, and the number of micro-batches N in an iteration to obtain models with various scales. We compare the WeiPipe-Interleave against widely used 1F1B, state-of-the-art zero-bubble 1 & 2, and FSDP under different communication infrastructures in terms of throughput, memory consumption and scalability. The 1F1B and zero-bubble strategies are implemented through Megatron-LM, while FSDP is achieved by the ZeRO-3 optimization in DeepSpeed. All strategies are applied with the same model configuration, micro-batch size, mixed-precision settings, recomputation, flash-attention, and hardware environment.

We choose Colossal Cloud with A800 GPUs as the experimental hardware environment. The A800 has 80GB HBM

Table 2. Throughput and peak memory of training the LLama-style model on 16 GPUs with NVLink and Ethernet connections. Due to the limitation of memory, for ZB strategies, G is set to 4 if S=4096 and G=1 if S=8192 or 16384.

Model Config			Throughput(Tokens/second/GPU)					Memory(GB)				
H	S	G	1F1B	ZB1	ZB2	FSDP	WeiPipe	1F1B	ZB1	ZB2	FSDP	WeiPipe
1024	4096	16	8581.7	7547.0	7638.5	11525.9	**15138.8**	13.0	20.4	39.3	8.6	9.4
	8192	8	7403.8	6739.6	6768.1	9424.4	**12122.3**	9.9	10.7	20.5	8.6	9.4
	16384	4	5641.2	5651.6	5651.9	6973.6	**8188.3**	9.1	21.6	42.2	8.6	9.4
2048	4096	16	4163.2	3823.3	OOM	4104.8	**6499.7**	18.7	44.3	OOM	17.9	19.9
	8192	8	3791.3	3517.8	OOM	3706.8	**6033.2**	19.6	22.3	OOM	17.9	19.9
	16384	4	3146.3	3050.1	OOM	3087.2	**4607.8**	22.9	42.9	OOM	17.9	19.9
4096	4096	16	1662.7	OOM	OOM	1110.5	**2023.1**	40.5	OOM	OOM	39	44.5
	8192	8	1556.2	OOM	OOM	1063.2	**2059.4**	41.6	OOM	OOM	39	44.5
	16384	4	1331.6	OOM	OOM	944.2	**1684.9**	45.1	OOM	OOM	39	44.5

Table 3. Throughput and peak memory of training the LLama-style model on 16 GPUs with PCIe and Ethernet connections. Due to the limitation of memory, for ZB strategies, G is set to 4 if S=4096 and G=1 if S=8192 or 16384.

Model Config			Throughput(Tokens/second/GPU)				
H	S	G	1F1B	ZB1	ZB2	FSDP	WeiPipe
1k	4k	16	8193	7708	7952	11545	**13847**
	16k	4	5394	4583	4630	6764	**7551**
2k	4k	16	4030	3701	OOM	4205	**5587**
	16k	4	2907	2638	OOM	3150	**4151**
4k	4k	16	**1530**	OOM	OOM	1186	1402
	16k	4	1232	OOM	OOM	966	**1505**

and 312 TFlops fp16/bf16 tensor cores. However, compared to the A100 GPUs, the A800 limits NVLink bandwidth from 600GB/s to 400GB/s. To evaluate the communication effectiveness of WeiPipe, we conducted experiments using two different communication infrastructures. The first setup involves 16 A800 GPUs in two clusters with NVLink[13] connections. The second setup features 32 A800 GPUs across 4 clusters with PCIe connections within each cluster. The clusters are connected by 10Gb Ethernet. We utilized the NCCL as the underlying communication library. The NCCL chooses ring-based implementation for collective primitives including reduce-scatter and all-gather that are used in FSDP by default, and tree algorithms are not adopted in our experiments. Therefore, we maintain a ring topology for all parallel strategies that appeared in the experiments.

6.2 Throughput and Memory Consumption

For the NVLink environment, the performance and maximum memory consumption of WeiPipe and other strategies are shown in Table 2. In this experiment, the hidden dimension size ranges from 4096 to 16384 combined with sequence length of 4096, 8192 and 16384 to get 9 different models, covering scenarios with long context. The model size ranges from 384M to 6.1B, unifying the batch size to 64. Among all configurations, WeiPipe demonstrates higher performance compared to other strategies. Specifically, when $H = 4096$ and $S = 16384$, WeiPipe achieves 22.3% and 78.4% improvement of throughput compared to 1F1B and FSDP respectively.

Table 4. Throughput of training the LLama-style model on 8 GPUs with NVLink connections with layer number as 16. For ZB strategies, G is set to 4 if S=4096 and G=1 if S=8192 or 16384.

Model Config			Throughput(Kilo Tokens/second/GPU)				
H	S	G	1F1B	ZB1	ZB2	FSDP	WeiPipe
1k	4k	16	32.0	45.8	**46.5**	37.9	31.3
	16k	4	15.9	22.0	**22.1**	17.8	16.9
2k	4k	16	15.0	**22.4**	OOM	17.0	14.2
	16k	4	9.4	12.8	OOM	10.1	9.7
4k	4k	16	5.2	OOM	OOM	**6.0**	4.9
	16k	4	3.7	OOM	OOM	**3.8**	3.6

Table 2 also presents the maximum memory usage of each strategy. 1F1B has the smallest memory usage among all pipeline strategies. Recomputation and Flash Attention notably reduce memory usage for 1F1B and the earlier workers in the zero-bubble strategy. However, the zero-bubble method incurs significantly higher memory consumption compared to the 1F1B method, making it prone to out-of-memory (OOM) errors in long context scenarios, partially contradicting the conclusions reported in [32] that ZB1's memory consumption is not higher than 1F1B, and ZB2's memory consumption is twice that of 1F1B. This discrepancy arises because the original method employs neither flash-attention nor recomputation; activations in the forward pass are mainly generated by attention computations. During the B Pass, this portion of the memory is released while gradients of the remaining activations are generated. Compared to the size of the eliminated activations, this memory overhead is minimal, so the overall peak memory still occurs before the first backward pass of the first Rank begins. However, after enabling flash-attention, the activations generated by the attention process are greatly reduced. At this point, the forward activations are mainly produced by the computations of the FFN, and the size of the gradients produced during the B pass is approximately equal to the size of the activations generated in one forward pass. Consequently, the peak memory appears before the first W Pass of the last Rank and is about twice the peak of the first Rank. Therefore, although zero-bubble pipeline parallelism can theoretically significantly reduce the computation bubble rate, the high memory consumption necessitates the use of smaller batch sizes, which to some extent compromises computational efficiency.

Conventional activation-passing PP typically use small micro-batch sizes due to the potentially large size of GM_A. However, with the adoption of Flash Attention and recomputation, memory consumption can be significantly reduced, allowing for larger micro-batch sizes and highlighting the advantages of WeiPipe. The memory usage for WeiPipe is slightly higher compared to FSDP due to the additional buffer requirements for communication. In WeiPipe-Interleave, each worker allocates three buffers to receive Ws and Ds and three buffers to send. In contrast, FSDP's implementation creates buffers on an operator-wise granularity, which may result in smaller and more fragmented buffers.

Table 3 presents the throughput in PCIe and Ethernet environment with 16 A800 GPUs from 2 clusters. Compared to NVLink, this environment further increases the communication pressure of large-context training. From Table 3, we can see that WeiPipe-Interleave achieves 31.7% and 22.2% improvement of throughput compared to the best performance of other strategies when $S = 16384$ and $H = 2048, 4096$.

The two experiments above are both in the presence of communication bottlenecks, where the low bandwidth of Ethernet makes a significantly impact. The results show that WeiPipe allows LLM training to be less dependent on high-bandwidth techniques. Similarly to DP, WeiPipe is well-suited for optimizations that overlap communication and computation. In our experiments, we implement layer-wise latency hiding in WeiPipe. However, in activation-passing methods, the previous layer must complete its computations before passing activations during the forward pass, and similarly during the backward pass. Communication hiding is constrained by data dependencies, while WeiPipe can transmit weights concurrently with computations. As a result, even in scenarios without long contexts, WeiPipe can still outperform current PP methods. We also conducted an experiment using eight A800 GPUs connected solely via NVLink, where communication constraints are less significant. The results in Table 4 indicate that in this environment, conventional methods may have advantages.

6.3 Weak Scaling

Weak scaling assesses how well a strategy preserves efficiency as both the number of workers and the size of tasks grow. We examine WeiPipe's ability of weak scaling by keeping the workload per worker constant while progressively increasing the number of workers from 8 to 32, and batch size from 128 to 512. The scalability experiments are performed on the 32 A800 GPUs from 4 clusters with PCIe connections within each cluster. We fixed the sequence length as 16384 and the layer number as 32. The results are shown in Figure 6, which reveals that the overall performance of WeiPipe scales proportionally to the increase in GPUs.

Figure 6. Weak scaling of different strategies. The number of GPUs scales from 8 to 32, and the batch size increases proportionally. The left coordinates correspond to the overall throughput as represented by the bar chart, while the coordinates on the right indicate tokens per second per GPU.

It can be seen there is a lower performance with PCIe connections of WeiPipe-Interleave, especially compared to FSDP. Through profiling, we found that the primary inefficiency in WeiPipe-Interleave occurs during the tail of the last backward pass. Worker 0 has to wait for other workers to complete the pipeline while still transferring large blocks of data. In contrast, FSDP divides communication into many fine-grained blocks, and high-bandwidth NVLink does not effectively reduce the frequency of interruptions. This result also suggests that further optimization of communication in WeiPipe is necessary to achieve ideal performance.

6.4 Strong Scaling

Unlike weak scaling, strong scaling measures scalability by keeping the task size constant while expanding the number of workers. In our experiments, the model configuration is the same as weak-scaling experiments, also increasing the number of GPUs from 8 to 32 but fix the batch size to 128. The result can be found in Figure 7, which shows that WeiPipe exhibits superior scalability compared to other strategies. This result reveals that WeiPipe has the potential to utilize more GPUs to achieve greater speed-up. The large token length and low-bandwidth communication infrastructure cause serious troubles for conventional PP approaches with increasing communication pressure.

`figures/exp_fig4.png`

Figure 7. Strong scaling of different strategies. The number of GPUs scales from 8 to 32, but the batch size remains 128.

7 Conclusions

In this work, we introduce a novel pipeline parallelism strategy aimed at reducing communication overhead in LLM training by shifting from activation-based to weight-based communication. We have developed and implemented the WeiPipe-Interleave to minimize the weight-passing overhead, reducing the bubble ratio and balancing memory usage. To further explore the potential of weight-passing pipelines, we discussed WeiPipe-zero-bubble methods to achieve nearly no-vacancy pipelining. Our experiments demonstrate that WeiPipe-Interleave can outperform current PP approaches and FSDP. WeiPipe extends the prospect of training large-context models with PP and reduces the reliance on expensive high-bandwidth communication infrastructure.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, et al. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[2] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer. arXiv:2004.05150 [cs.CL]

[3] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. 2024. https://openai.com/research/video-generation-models-as-world-simulators

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

[5] Shenggan Cheng, Ziming Liu, Jiangsu Du, and Yang You. 2023. ATP: Adaptive Tensor Parallelism for Foundation Models. arXiv:2301.08658 [cs.DC] https://arxiv.org/abs/2301.08658

[6] Shenggan Cheng, Xuanlei Zhao, Guangyang Lu, Jiarui Fang, Zhongming Yu, Tian Zheng, Ruidong Wu, Xiwen Zhang, Jian Peng, and Yang You. 2023. FastFold: Reducing AlphaFold Training Time from 11 Days to 67 Hours. arXiv:2203.00854 [cs.LG]

[7] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311 [cs.CL] https://arxiv.org/abs/2204.02311

[8] 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]

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

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV] https://arxiv.org/abs/2010.11929

[11] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan

Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[12] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, Lansong Diao, Xiaoyong Liu, and Wei Lin. 2020. DAPPLE: A Pipelined Data Parallel Approach for Training Large Models. https://doi.org/10.48550/ARXIV.2007.01045

[13] Denis Foley and John Danskin. 2017. Ultra-Performance Pascal GPU and NVLink Interconnect. *IEEE Micro* 37, 2 (2017), 7–17. https://doi.org/10.1109/MM.2017.37

[14] 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. LoongTrain: Efficient Training of Long-Sequence LLMs with Head-Context Parallelism. arXiv:2406.18485 [cs.DC] https://arxiv.org/abs/2406.18485

[15] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2018. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. https://doi.org/10.48550/ARXIV.1811.06965

[16] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. 2023. DeepSpeed Ulysses: System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models. arXiv:2309.14509 [cs.LG]

[17] Arpan Jain, Ammar Ahmad Awan, Asmaa M. Aljuhani, Jahanzeb Maqbool Hashmi, Quentin G. Anthony, Hari Subramoni, Dhableswar K. Panda, Raghu Machiraju, and Anil Parwani. 2020. GEMS: GPU-Enabled Memory-Aware Model-Parallelism System for Distributed DNN Training. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15. https://doi.org/10.1109/SC41405.2020.00049

[18] John M. Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David A. Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray

Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596 (2021), 583 – 589. https://api.semanticscholar.org/CorpusID:235959867

[19] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. arXiv:2001.08361 [cs.LG] https://arxiv.org/abs/2001.08361

[20] Huan Yee Koh, Jiaxin Ju, Ming Liu, and Shirui Pan. 2022. An Empirical Survey on Long Document Summarization: Datasets, Models, and Metrics. *Comput. Surveys* 55 (2022), 1 – 35. https://api.semanticscholar.org/CorpusID:250118028

[21] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! arXiv:2305.06161 [cs.CL] https://arxiv.org/abs/2305.06161

[22] Shigang Li and Torsten Hoefler. 2021. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[23] Shengwei Li, Zhiquan Lai, Yanqi Hao, Weijie Liu, Keshi Ge, Xiaoge Deng, Dongsheng Li, and Kai Lu. 2023. Automated Tensor Model Parallelism with Overlapped Communication for Efficient Foundation Model Training. arXiv:2305.16121 [cs.DC] https://arxiv.org/abs/2305.16121

[24] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. 2022. Sequence Parallelism: Long Sequence Training from System Perspective. arXiv:2105.13120 [cs.LG]

[25] Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring Attention with Blockwise Transformers for Near-Infinite Context. arXiv:2310.01889 [cs.CL]

[26] Ziming Liu, Shenggan Cheng, Haotian Zhou, and Yang You. 2023. Hanayo: Harnessing Wave-like Pipeline Parallelism for Enhanced Large Model Training Efficiency. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 56, 13 pages. https://doi.org/10.1145/3581784.3607073

[27] Ziming Liu, Shaoyu Wang, Shenggan Cheng, Zhongkai Zhao, Xuanlei Zhao, James Demmel, and Yang You. 2024. WallFacer: Guiding Transformer Model Training Out of the Long-Context Dark Forest with N-body Problem. arXiv:2407.00611 [cs.DC] https://arxiv.org/abs/2407.00611

[28] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. arXiv:1710.03740 [cs.AI] https://arxiv.org/abs/1710.03740

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. https://doi.org/10.48550/ARXIV.1912.01703

[30] William Peebles and Saining Xie. 2023. Scalable Diffusion Models with Transformers. arXiv:2212.09748 [cs.CV] https://arxiv.org/abs/2212.09748

[31] William Peebles and Saining Xie. 2023. Scalable Diffusion Models with Transformers. arXiv:2212.09748 [cs.CV]

[32] 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

[33] 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

[34] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL] https://arxiv.org/abs/2308.12950

[35] 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. *arXiv preprint arXiv:1909.08053* (2019).

[36] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[37] 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. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] https://arxiv.org/abs/2307.09288

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[39] Boxiang Wang, Qifan Xu, Zhengda Bian, and Yang You. 2022. Tesseract: Parallelize the Tensor Parallelism Efficiently. In *Proceedings of the 51st International Conference on Parallel Processing*. 1–11.

[40] Qifan Xu, Shenggui Li, Chaoyu Gong, and Yang You. 2021. An efficient 2d method for training super-large deep learning models. *arXiv preprint arXiv:2104.05343* (2021).

[41] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. arXiv:2304.11277 [cs.DC] https://arxiv.org/abs/2304.11277