TawPipe: Topology-Aware Weight Pipeline Parallelism for Accelerating Long-Context Large Models Training

Houming Wu[1,2], **Ling Chen**[1,2,*]

[1]State Key Laboratory of Blockchain and Data Security, Zhejiang University
[2]College of Computer Science and Technology, Zhejiang University
{houmingwu, lingchen}@cs.zju.edu.cn

Abstract

Training large language models (LLMs) is fundamentally constrained by limited device memory and costly inter-device communication. Although pipeline parallelism alleviates memory pressure by partitioning models across devices, it incurs activation communication overhead that scales linearly with sequence length, limiting efficiency in long-context training. Recent weight-passing approaches (e.g., WeiPipe) mitigate this by transmitting model weights instead of activations, but suffer from redundant peer-to-peer (P2P) transfers and underutilized intra-node bandwidth. We propose **TawPipe**—**t**opology-**a**ware **w**eight **pipe**line parallelism, which exploits hierarchical bandwidth in distributed clusters for improved communication efficiency. TawPipe: (i) groups devices based on topology to optimize intra-node collective and inter-node P2P communication; (ii) assigns each device a fixed shard of model weights and gradients, avoiding redundant transfers; and (iii) overlaps communication with computation to hide latency. Unlike global collective operations used in fully sharded data parallelism (FSDP), TawPipe confines most communication within node boundaries, significantly reducing cross-node traffic. Extensive experiments on up to 24 GPUs with LLaMA-style models show that TawPipe achieves superior throughput and scalability compared to state-of-the-art baselines.

Code — https://github.com/wuhouming/TawPipe

Introduction

Transformer-based large language models (LLMs) have achieved impressive success across a wide range of natural language processing (NLP) tasks. Although empirical evidence consistently shows that scaling model size and training data leads to improved performance, training such large models remains fundamentally constrained by two key challenges: (1) limited device memory that restricts model capacity, and (2) high inter-device communication overhead that hampers efficiency in distributed training.

To overcome these challenges, various parallelization techniques have been proposed, including data parallelism (DP) (Li et al. 2014; Jiang et al. 2020), tensor parallelism (TP) (Shoeybi et al. 2019; Wang et al. 2022), sequence parallelism (SP) (Li et al. 2023; Wu et al. 2024), and pipeline parallelism (PP) (Huang et al. 2019; Guan et al. 2024). Among these, PP stands out for its relatively low memory footprint and high compute utilization, especially when combined with mixed-precision training (Micikevicius et al. 2017), activation checkpointing (Chen et al. 2016; Liu et al. 2025), and FlashAttention (Dao et al. 2022).

Conventional PP approaches (e.g., GPipe (Huang et al. 2019), DAPPLE (Fan et al. 2021), BitPipe (Wu, Chen, and Yu 2024) and Zero-Bubble PP (Qi et al. 2024)) divide models into pipeline stages and exchange intermediate activations across stages. This results in communication costs that scale with sequence length S, micro-batch size B, and hidden dimension H, yielding per-layer messages of size BSH. In long-context training scenarios, where S is large, activation communication becomes the dominant bottleneck, severely limiting scalability.

To mitigate this, recent work has explored weight-passing pipeline parallelism (e.g., WeiPipe (Lin et al. 2025)), which transmits model weights instead of activations. This strategy decouples communication volume from sequence length and batch size, offering theoretical benefits when the activation-to-weight size ratio exceeds one. However, WeiPipe faces practical inefficiencies due to its exclusive reliance on peer-to-peer (P2P) communication. Specifically: (1) it fails to exploit high-speed intra-node interconnects by ignoring bandwidth asymmetry between intra- and inter-node links, and (2) it incurs redundant data transfers and elevated memory overhead, as its ring-based communication pattern requires two full communication rounds per iteration and maintains two weight buffers per device, particularly during pipeline warm-up and cool-down phases.

To this end, we propose **t**opology-**a**ware **w**eight **pipe**line parallelism (**TawPipe**), a communication-efficient framework that exploits hierarchical bandwidth and device topology in distributed clusters. TawPipe unifies two weight-passing extremes—FSDP ((Rajbhandari et al. 2020; Zhao et al. 2023)) with global collectives and WeiPipe with pure P2P exchange, thereby reducing communication overhead and maximizing bandwidth utilization. Our main contributions are summarized as follows:

- We propose a group-based weight pipeline scheduler (GWPS) that partitions devices into topology-aware

*Corresponding Author

groups, integrating intra-node collective communication with inter-node P2P transfers to reduce cross-node traffic and improve bandwidth efficiency.

- We introduce a device-bound storage (DBS) strategy that assigns each device a fixed shard of model weights and gradients, avoiding redundant transfers and lowering memory consumption.
- We incorporate a communication-computation overlap (CCO) mechanism that asynchronously prefetches weights during computation, effectively hiding inter-node communication latency.
- We evaluate TawPipe on training long-context LLaMA-style (Touvron et al. 2023) models across up to 24 GPUs, demonstrating state-of-the-art throughput with a balanced and modest memory footprint.

Table 1 summarizes the key symbols used throughout this paper for ease of reference and analysis.

P	The number of devices
P_i	The ith device in a cluster
D	The number of groups ($P \bmod D = 0$)
L	The number of layers in neural network
H	The size of hidden dimension in Transformer
S	The sequence length in Transformer
B	Micro-batch size
N	The number of micro-batches in a mini-batch
A_j^i	Activation values of jth layer in ith micro-batch
W_j	Weights of jth layer
G_j	Gradients of W_j
M_W	Memory consumption for the weights of one stage
M_A	Memory consumption for the activations of one stage
T_C	Communication volume for the data of one stage

Table 1: Symbols used in the paper.

Related Work

Data Parallelism

Data parallelism (DP) replicates a model across multiple devices and distributes input data among them (Li et al. 2014; Jiang et al. 2020; Zhou, Chen, and Wu 2023). While effective for models of moderate size, DP becomes infeasible when the model parameters exceed the memory capacity of a single device. FSDP ((Rajbhandari et al. 2020; Zhao et al. 2023)) addresses this limitation by sharding parameters, gradients, and optimizer states across devices. However, its reliance on global collective communication introduces scalability bottlenecks, especially in bandwidth-constrained environments.

Pipeline Parallelism

Pipeline parallelism (PP) partitions a model into sequential stages distributed across multiple devices, with activations transmitted between stages. Synchronous PP approaches (e.g., GPipe (Huang et al. 2019)) enforce strict iteration-level synchronization to ensure convergence, but incur substantial activation memory overhead by scheduling all forward passes before backward passes. To mitigate this overhead, DAPPLE (Fan et al. 2021) adopts 1F1B scheduling, and Zero-Bubble (Qi et al. 2024) further introduces gradient decoupling. Asynchronous PP approaches (e.g., PipeDream (Narayanan et al. 2019), PipeMare (Yang et al. 2021), and PipeDream-2BW (Narayanan et al. 2021a)) improve hardware utilization by relaxing synchronization constraints, but risk gradient staleness and convergence instability. Critically, all these approaches incur communication overheads proportional to activation size, which scales with sequence length, micro-batch size, and hidden dimension, thereby limiting efficiency in long-context model training.

Weight-passing PP approaches (e.g., WeiPipe (Lin et al. 2025)) mitigate this limitation by transmitting model weights instead of activations. This design reduces communication volume to a constant level, independent of sequence length or batch size, and is theoretically advantageous when the activation-to-weight size ratio exceeds one. However, WeiPipe relies on a fixed ring-based P2P communication pattern that neglects hardware topology and computational dependencies, leading to redundant transfers and suboptimal use of high-speed intra-node interconnects (e.g., NVLink).

Topology-Aware Communication

Topology-aware communication improves bandwidth utilization by adapting data transfers to the underlying hardware topology. This concept is implemented in communication libraries (e.g., TACCL (Shah et al. 2023) and HiCCL (Hidayetoglu et al. 2024)) and further explored in distributed training systems. BytePS (Jiang et al. 2020) employs hierarchical parameter servers to decouple intra-node and inter-node communication, while TopoOpt (Wang et al. 2023) co-optimizes device placement and communication scheduling based on topology modeling. Although effective in enhancing collective communication for data-parallel training, these solutions lack native support for pipeline-parallelism and weight-passing schemes.

To address these gaps, we propose TawPipe. Instead of relying solely on P2P communication, TawPipe introduces hierarchical communication scheduling into weight-passing PP, enabling efficient use of both intra- and inter-node bandwidth. In addition, it integrates topology-aware device grouping with device-bound storage to align communication with hardware topology and eliminate redundant transfers.

Methodology

Overview

TawPipe comprises three tightly coupled components (as depicted in Figure 1): (i) Device-Bound Storage (DBS) assigns each device a fixed weight shard to eliminate redundant transfers and reduce memory overhead, (ii) Group-based Weight Pipeline Scheduler (GWPS) orchestrates topology-aware weight propagation to maximize intra-node bandwidth and computation efficiency, and (iii) Communication-Computation Overlap (CCO) asynchronously prefetches weights to hide inter-node communication latency. Together, these components form a unified system that minimizes communication overhead while improving throughput and scalability in distributed environments.

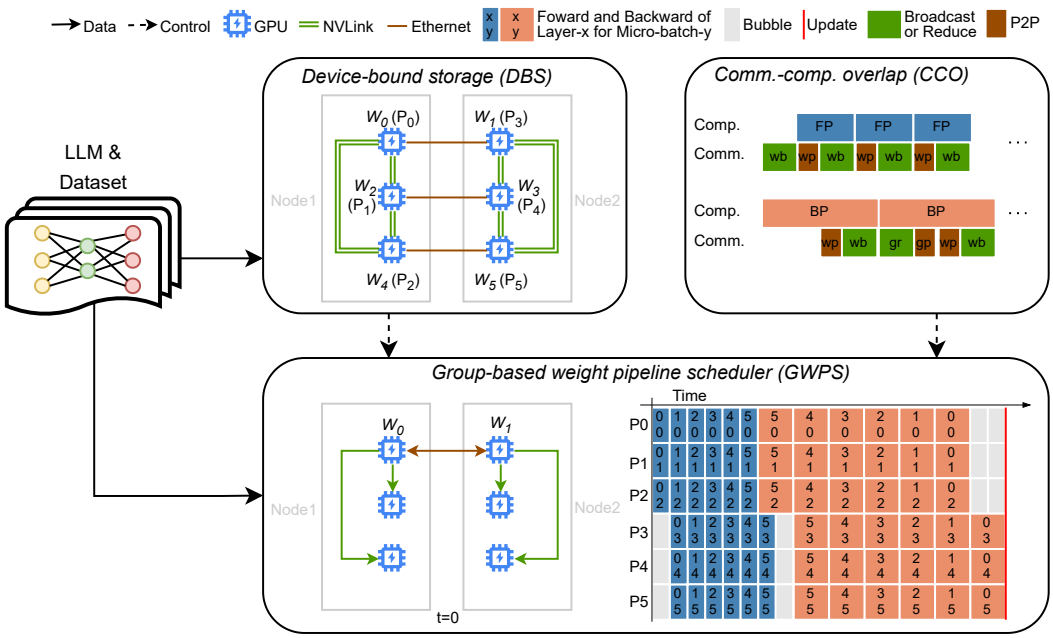
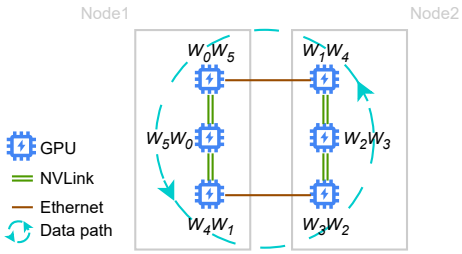


Figure 1: Overview of the TawPipe design. DBS fixes weight shards on each device, GWPS aligns communication with the underlying hardware topology, and CCO overlaps inter-group prefetching with computation. "wp", "wb", "gp", and "gr" denote weight passing, weight broadcasting, gradient passing, and gradient reduction, respectively.

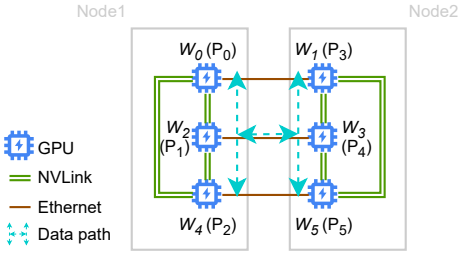
Device-Bound Storage

WeiPipe's ring-based exchange scheme requires each device to cyclically store and transmit multiple weight shards, leading to increased buffer usage and redundant data transfers. To address this, TawPipe introduces DBS, which statically assigns each layer's weights and gradients to a specific device. Communication is triggered only when a device needs to compute with a remote weight shard, ensuring that at most one shard is transferred per device at any given time.

Figure 2 illustrates the difference in weight initialization between the two strategies for a six-GPU setup. In the ring-based strategy, each device must maintain two distinct weight buffers (e.g., W_0 and W_5 in P_0) and complete two full communication cycles per iteration. In contrast, the device-bound strategy statically assigns a single weight shard to each device (e.g., W_0 to P_0), with communication initiated only as needed. This design eliminates redundant buffer allocation and reduces communication rounds by up to 50%. In addition, DBS integrates effectively with conventional pipeline scheduling and communication primitives (e.g., `Send/Recv` and `Broadcast/Reduce`), making it practical for scalable training for large-scale models.



(a) Ring-based initialization



(b) Device-bound initialization

Figure 2: Weight initialization under ring-based and device-bound strategies. Each set of three GPUs corresponds to a compute node. (a) Ring-based scheme needs to buffer and rotate two weight shards continuously. (b) Device-bound strategy stores one weight shard and initiates communication only when needed.

Group-Based Weight Pipeline Scheduler

GWPS improves communication locality by organizing devices into topology-aware groups, typically aligned with physical nodes (e.g., one group per node), and assigning model layers in an alternating pattern. Given P devices and $L \bmod P = 0$ layers, devices are evenly divided into D groups, where D generally corresponds to the number of nodes and P is divisible by D. Specifically,

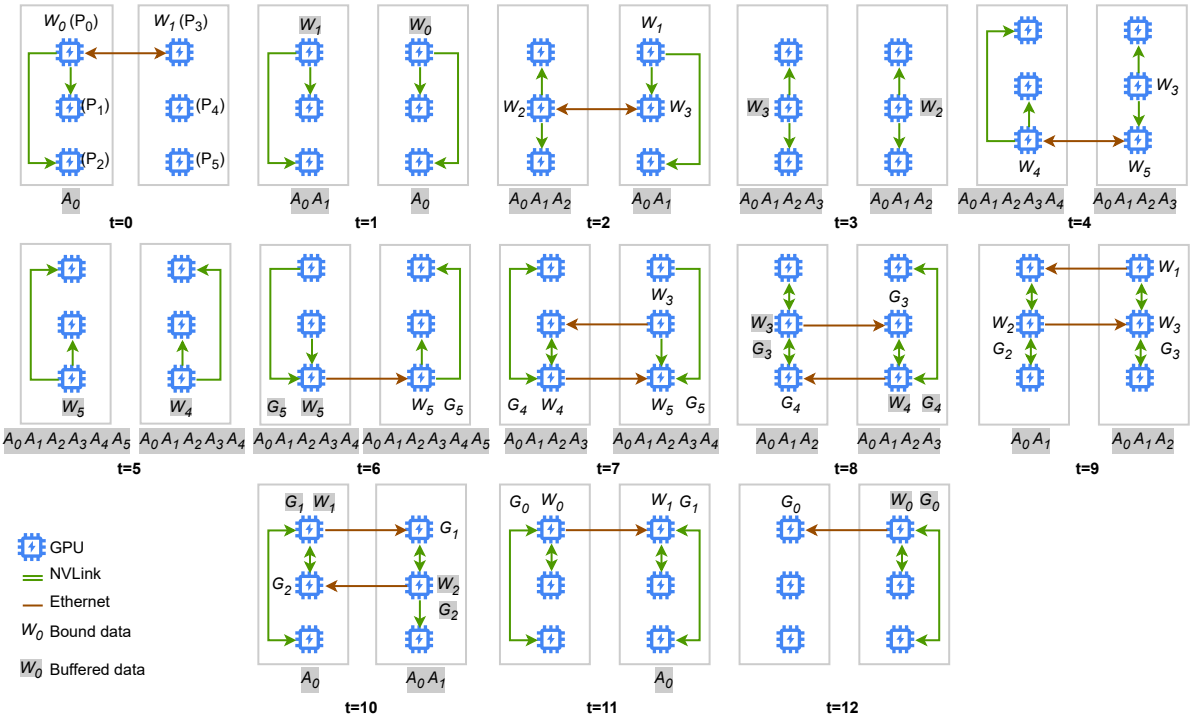


Figure 3: Group-based weight pipeline scheduling. The texts without background color represent device-bound data. The text with gray background color denotes buffered data in memory. For instance, at $t = 4$, P_2 holds W_4, and receives W_5 from P_5.

group g_0 includes devices $\{\mathrm{P}_0, \mathrm{P}_1, ..., \mathrm{P}_{P/D-1}\}$, group g_1 includes $\{\mathrm{P}_{P/D}, \mathrm{P}_{P/D+1}, ..., \mathrm{P}_{2P/D-1}\}$, and so on. Each device holds exactly one weight shard. Within group g_k ($k \in [0, D-1]$), device P_i ($i \in [0, P/D-1]$) holds weight shard $W_{(D \cdot i+k) \bmod P}$, resulting in a balanced and interleaved layer-to-device mapping across all groups. To support efficient pipelining, each group assigns two logical roles: the master device, which initially holds the weight shard required for the current computation step, and the staging device, which asynchronously prefetches the next weight shard from a remote group for the upcoming step.

Communication is structured hierarchically into intra- and inter-group phases. Intra-group communication exploits high-bandwidth collectives for weight broadcasting and gradient aggregation, where the master or staging device distributes weights and collects gradients within the group. Inter-group communication is restricted to lightweight P2P transfers of weights and aggregated gradients across groups. This hierarchical design localizes most traffic to intra-node links, substantially reducing cross-node communication overhead.

Training proceeds in three phases: forward, backward, and update. In the forward pass, at step $t = 0$, device P_0 broadcasts weight shard W_0 within group g_0, initiating parallel computation. Simultaneously, P_0 sends W_0 to $\mathrm{P}_{P/D}$ and receives W_1 in return. Devices in g_0 cache the resulting activations A_0 and proceed to the next layer using W_1, while $\mathrm{P}_{P/D}$ broadcasts W_0 within g_1 to start computation. This staggered communication and computation pipeline proceeds until all layers are processed. The forward steps 0 to 5 are illustrated at the top of Figure 3. The backward pass involves local gradient reductions within a group and inter-group transfers to the owner of the respective shard. For example, gradients for W_{L-1} are first computed by devices in group g_0, reduced to device $\mathrm{P}_{P/D-1}$, and then transferred to P_{P-1}—the owner of W_{L-1}, for final aggregation and update, as illustrated in steps 6 to 11 (bottom of Figure 3). Once gradients arrive at their corresponding shard owners, updates are applied locally using colocated optimizer states, avoiding additional communication during the update phase. For example, at $t = 7$, device P_5 updates W_5 using gradient G_5 without any inter-device synchronization.

By aligning the scheduling with hardware topology, GWPS minimizes cross-node communication and maximizes intra-node bandwidth utilization, leading to efficient, scalable training performance.

Communication-Computation Overlap

To further improve pipeline efficiency, TawPipe introduces CCO, which hides inter-group transfer latency by asynchronously prefetching remote weight shards required for the next step. Specifically, during execution at time step t, each staging device initiates non-blocking transfers of the weight needed for step $t+1$, provided it resides on another group.

CCO is implemented through dedicated memory buffers to decouple communication with computation, combined with synchronization mechanisms to ensure data con-

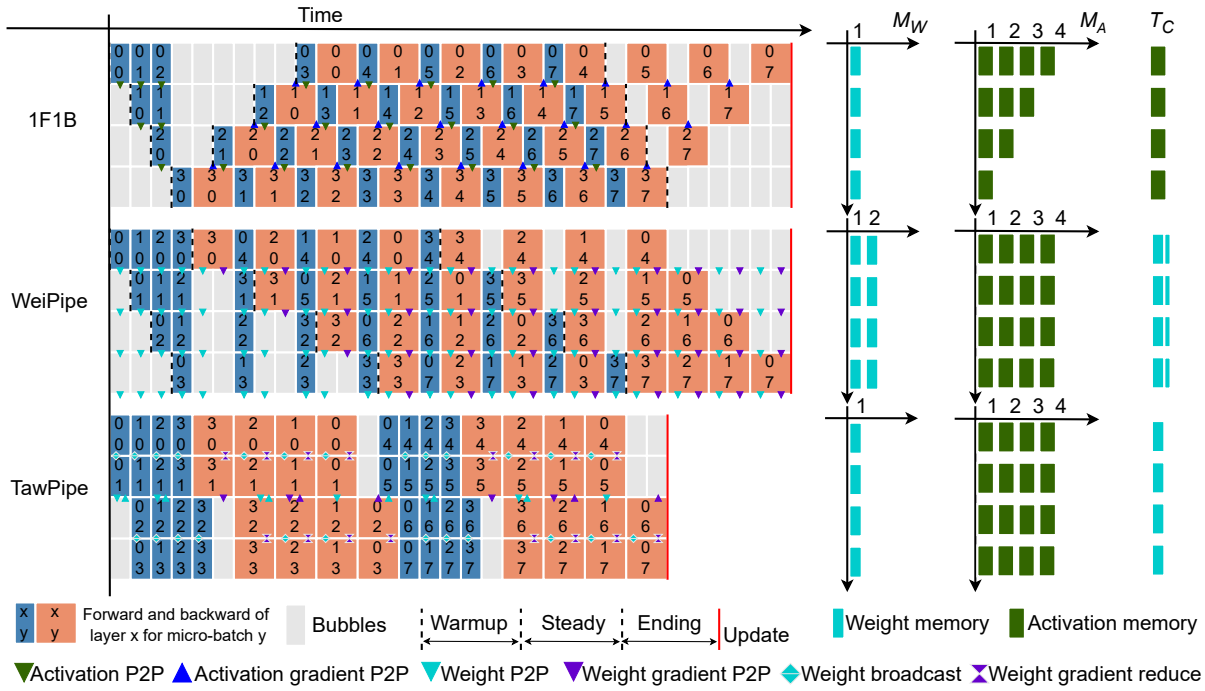


Figure 4: Comparison of TawPipe with 1F1B and WeiPipe. In the pipelining graph, the upper number in a block represents the layer index and the bottom number is the micro-batch index. TawPipe has a lower bubble ratio as the pipeline flush happens sooner in the timeline.

sistency and avoid resource contention. For example, by leveraging non-blocking communication APIs (e.g., `torch.distributed.isend/irecv`), weight transfers for the next step are launched in parallel with the current forward or backward computation, effectively masking communication delays and improving pipeline utilization.

Theoretical Analysis

We compare TawPipe against two representative baselines: 1F1B (Fan et al. 2021), a widely adopted activation-passing pipeline parallelism approach, and WeiPipe (Lin et al. 2025), a recent weight-passing variant. The analysis focuses on three dimensions: pipeline efficiency, memory usage, and communication volume, as illustrated in Figure 4.

Pipeline utilization is quantified by the bubble ratio, defined as the fraction of idle time over total execution time. TawPipe achieves a lower bubble ratio of $\frac{(D-1)\cdot P+N}{(3N+D-1)\cdot P+N}$, compared to $\frac{P-1}{N+P-1}$ for both 1F1B and WeiPipe, indicating more efficient pipeline scheduling with fewer idle slots.

Memory usage is dominated by activation buffers, with weight buffers contributing to a lesser extent. Although all three approaches have comparable peak activation memory usage (approximately BM_A), TawPipe achieves better memory balance. In addition, through the DBS strategy, TawPipe further reduces weight buffer overhead from $2M_W$ (in WeiPipe) to M_W, lowering memory footprint.

Communication volume is approximated by total data received per device per iteration. For a typical LLaMA-style layer with $12H^2$ parameters and activation volume BSH, 1F1B transfers $2PBSH$ activations per step. In contrast, TawPipe transfers a single weight shard and its corresponding gradients per step, totaling $24H^2$. This represents a 33% reduction compared to WeiPipe's $36H^2$, making TawPipe particularly communication-efficient in long-sequence bandwidth-constrained settings.

Experiments

Experimental Setup

Models and Environment. We evaluate TawPipe using models derived from the LLaMA-2 architecture (Touvron et al. 2023) on the C4 dataset (Raffel et al. 2020). To assess scalability, we systematically vary the hidden dimension H, sequence length S, and number of layers L. All experiments are conducted on a cluster with up to 24 NVIDIA A800 GPUs (80GB each). Within each node, GPUs are interconnected via NVLink, while inter-node communication is performed over 10Gb Ethernet (10GbE).

Baselines and Implementation. We compare TawPipe against several state-of-the-art pipeline parallelism and memory optimization approaches:

- **1F1B** (Fan et al. 2021). A widely used activation-passing pipeline parallelism baseline. We use the implementation from Megatron-LM project (Narayanan et al. 2021b).
- **Zero-Bubble (ZB-1 & ZB-2)** (Qi et al. 2024). A recent pipeline parallelism approach that decouples weight and

Setup	H	1024			2048			4096		
	S	4096	8192	16384	4096	8192	16384	4096	8192	16384
	B (*)	16 (4)	8 (1)	4 (1)	16 (4)	8 (1)	4 (1)	16 (4)	8 (1)	4 (1)
Throughput (Tokens/ GPU/ second)	1F1B	7212.1	6636.4	5593.7	3028.5	2886.1	2900.7	1247.8	1365.8	1114.2
	ZB1	5893.3	5803.1	5883.4	2884.5	2839.8	2914.8	OOM	1323.3	OOM
	ZB2	6021.3	5918.8	5902.8	OOM	2860.5	OOM	OOM	OOM	OOM
	FSDP	10559.2	8826.3	6750.5	3860.3	3628.4	3165.2	1128.1	1086.5	956.1
	WeiPipe	12054.8	10663.1	8412.3	4008.1	4377.0	3841.7	1038.1	1158.9	1232.4
	TawPipe	**13629.2**	**11738.2**	**8913.7**	**5040.0**	**4636.6**	**4175.7**	**1567.7**	**1511.5**	**1377.6**
Memory (GB)	1F1B	**14.5**	**16.1**	20.4	27.6	28.9	34.8	55.1	54.9	62.3
	ZB1	34.3	19.0	36.7	67.6	35.9	69.3	OOM	71.1	OOM
	ZB2	64.5	34.2	67.8	OOM	66.8	OOM	OOM	OOM	OOM
	FSDP	19.4	19.4	**19.4**	27.8	27.8	27.8	**52.0**	**52.0**	**52.0**
	WeiPipe	22.0	22.0	22.0	29.0	29.0	29.0	57.8	57.8	57.8
	TawPipe	19.6	19.6	19.6	**27.5**	**27.5**	**27.5**	56.7	56.7	56.7

Table 2: Throughput and peak memory usage of training LLaMA-style models on 24 GPUs with NVLink and Ethernet connections. For ZB strategies, B is set to 4 when $S = 4096$ and $B = 1$ when $S \in \{8192, 16384\}$. The best results are **bolded**. The second-best results are underlined.

activation gradient computations. We use the implementation released by the authors.

- **FSDP** (Zhao et al. 2023). An enhanced data parallelism strategy that partitions weights, gradients, and optimizer states across devices. We use the implementation based on ZeRO-3 from DeepSpeed (Rasley et al. 2020).
- **WeiPipe** (Lin et al. 2025). A recent weight-passing pipeline parallelism approach. We use the implementation released by the authors.

TawPipe is implemented by extending WeiPipe, with modifications to the pipeline scheduler and communication engine to incorporate device-bound storage and group-based scheduling. All approaches are evaluated under identical settings, including model architecture, batch size, mixed-precision training (FP16), FlashAttention, and hardware configuration. Activation checkpointing is applied uniformly across all approaches, except for Zero-Bubble, where it offers no memory savings and incurs additional overhead (Lin et al. 2025). To ensure fair comparison, all approaches use the NCCL backend (Jeaugey 2017) for communication. All reported results are averaged over multiple runs to ensure statistical robustness.

Metrics. We evaluate all approaches in terms of end-to-end throughput (measured in tokens per second) and peak device memory usage. In addition, we conduct ablation studies to isolate and quantify the contribution of each design component in TawPipe.

Throughput and Memory Usage

We evaluate the throughput and memory efficiency of TawPipe using a 48-layer transformer model across 24 GPUs. Model complexity is scaled by varying the hidden dimension $H \in \{1024, 2048, 4096\}$ and sequence length $S \in \{4096, 8192, 16384\}$, resulting in model sizes ranging from 668 million to 10 billion parameters (i.e., 668M to 10B). The global batch size is fixed at 1536, with micro-batch size B adjusted according to memory constraints.

Table 2 summarizes the throughput and peak memory consumption across configurations. Key findings include: (1) TawPipe consistently achieves the highest throughput, particularly under communication-intensive settings. For the most demanding configuration $(H, S) = (4096, 16384)$, it outperforms WeiPipe, 1F1B, and FSDP by 11.8%, 23.6%, and 44.1%, respectively. (2) The performance advantage of TawPipe grows with model size. At sequence length $S = 16384$, TawPipe's throughput improvement over WeiPipe increases from 6.0% to 11.8% as hidden dimension H scales from 1024 to 4096. (3) While peak memory usage remains comparable across most approaches (excluding Zero-Bubble), TawPipe achieves better memory balance across devices and reduces weight buffer overhead through its device-bound storage strategy.

Scalability Study

We assess the scalability of TawPipe through both weak and strong scaling experiments. In weak scaling, we proportionally increase the number of GPUs and the global batch size to maintain a constant per-device workload. In strong scaling, we fix the global batch size while varying the number of GPUs. Considering the accommodation of a single node (i.e., 8 GPUs), all experiments use a fixed model configuration of $(S, H, L) = (16384, 1024, 48)$ with $B = 2$, except for ZB-2, which uses $B = 1$ due to higher memory usage.

Figure 5 presents weak scaling results. Key observations include: (1) Conventional activation-passing approaches (i.e., 1F1B and Zero-Bubble) perform well only in tightly coupled environments (e.g., ≤ 8 GPUs with NVLink), where communication overhead is less impactful. (2) As the system scales across nodes, weight-passing approaches (i.e., WeiPipe and TawPipe) show greater relative speedups, with TawPipe achieving the best performance. (3) TawPipe exhibits near-linear scaling, demonstrating efficient utilization of compute and communication resources with minimal idle time.

Figure 6 presents strong scaling results. Key findings in-

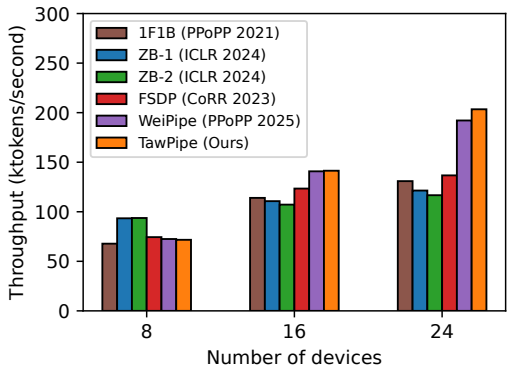


Figure 5: Weak scaling. The number of GPUs scales from 8 to 24 (8 GPUs in a node) with global batch size increasing proportionally from 512 to 1536.

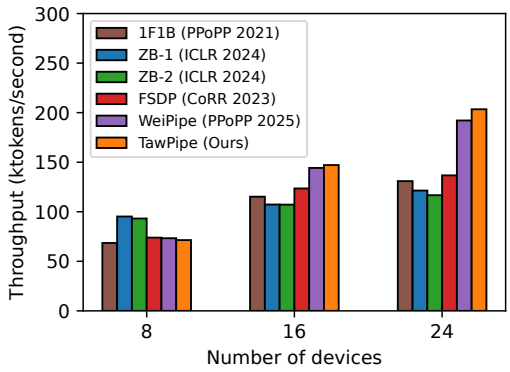


Figure 6: Strong scaling. The number of GPUs scales from 8 to 24 with the global batch size remaining 1536.

Approach	NCCL-Ratio	Duration	Throughput
1F1B (PPoPP 2021)	48.0%	105.1	5.59
ZB-1 (ICLR 2024)	77.6%	181.1	5.88
ZB-2 (ICLR2024)	77.5%	180.5	5.90
FSDP (CoRR 2023)	<u>33.7%</u>	<u>41.7</u>	6.75
WeiPipe (PPoPP 2025)	63.7%	194.0	<u>8.41</u>
TawPipe (Ours)	**24.1%**	**34.7**	**8.91**

Table 3: NCCL kernel time ratio, absolute duration (seconds), and throughput (kilo tokens/second). The best results are **bolded**. The second-best results are <u>underlined</u>.

clude: (1) Zero-Bubble strategies scale poorly due to higher memory consumption, which restricts the micro-batch size and reduces parallel efficiency. (2) Approaches relying on intensive inter-node communication (i.e., FSDP and 1F1B) suffer from degraded scalability as node number increases from 1 to 3. (3) TawPipe delivers the best strong scaling, which distributes fixed workloads efficiently without saturating communication bandwidth.

Communication Study

To evaluate the communication efficiency of TawPipe, we utilize NVIDIA Nsight Systems to capture GPU kernel traces and measure both the proportion of time spent on NCCL communication operations (e.g., `ncclBroadcast` and `ncclSend/Recv`), along with their absolute durations (Soytürk et al. 2021; Shah et al. 2023).

Table 3 presents results for a 48-layer model with $(S, H) = (16384, 1024)$ on 24 GPUs. TawPipe achieves the lowest NCCL kernel time ratio and the shortest overall communication time among all methods. It reduces NCCL execution time by up to 82.1% compared to WeiPipe (the second best in throughput) and by 16.8% compared to FSDP (the second best in absolute duration). These gains result from hierarchical communication that fully utilizes intra-node bandwidth and selective inter-node transfers that minimize cross-node traffic.

Setup	1024	2048	4096
w/o-GWPS	8.59 (-3.6%)	3.91 (-6.5%)	1.26 (-8.7%)
w/o-CCO	8.22 (-7.7%)	3.47 (-17.0%)	1.14 (-17.4%)
TawPipe	**8.91**	**4.18**	**1.38**

Table 4: The performance of TawPipe and the variants (in kilo tokens/second). The best results are **bolded**.

Ablation Study

To assess the contribution of each component in TawPipe, we conduct an ablation study on two modified variants: one without the group-based weight pipeline scheduler (replaced by WeiPipe's ring-based exchange) and another without communication–computation overlap (by disabling asynchronous prefetching).

Table 4 presents the throughput results on a 48-layer model with $S = 16384$ and $H \in \{1024, 2048, 4096\}$ across 24 GPUs. Key observations include: (1) TawPipe-w/o-CCO shows the largest throughput decline, confirming the effectiveness of overlapping inter-group transfers with computation. (2) TawPipe-w/o-GWPS also incurs a significant drop, underscoring the role of group-based scheduling in reducing inter-node communication and enhancing intra-node bandwidth utilization.

Conclusions

In this paper, we propose TawPipe, a topology-aware weight pipeline parallelism framework designed to accelerate the training of long-context large models. TawPipe introduces a group-based weight scheduling mechanism that exploits hierarchical hardware topology, a device-bound storage strategy that eliminates redundant data transfers, and a communication–computation overlap technique that hides inter-node latency. Experiments on LLaMA-style models across up to 24 GPUs show that TawPipe achieves state-of-the-art throughput with balanced and modest memory usage, demonstrating its scalability and efficiency for large-scale distributed training.

References

Chen, T.; Xu, B.; Zhang, C.; and Guestrin, C. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*.

Dao, T.; Fu, D.; Ermon, S.; Rudra, A.; and Ré, C. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35: 16344–16359.

Fan, S.; Rong, Y.; Meng, C.; Cao, Z.; Wang, S.; Zheng, Z.; Wu, C.; Long, G.; Yang, J.; Xia, L.; et al. 2021. DAPPLE: A pipelined data parallel approach for training large models. In *Proceedings of the ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, 431–445.

Guan, L.; Li, D.-S.; Liang, J.-Y.; Wang, W.-J.; Ge, K.-S.; and Lu, X.-C. 2024. Advances of pipeline model parallelism for deep learning training: An overview. *Journal of Computer Science and Technology*, 39(3): 567–584.

Hidayetoglu, M.; de Gonzalo, S. G.; Slaughter, E.; Surana, P.; Hwu, W.-m.; Gropp, W.; and Aiken, A. 2024. HiCCL: A hierarchical collective communication library. *arXiv preprint arXiv:2408.05962*.

Huang, Y.; Cheng, Y.; Bapna, A.; Firat, O.; Chen, D.; Chen, M.; Lee, H.; Ngiam, J.; Le, Q. V.; Wu, Y.; et al. 2019. GPipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in Neural Information Processing Systems*, 32.

Jeaugey, S. 2017. NCCL 2.0. In *GPU Technology Conference (GTC)*, volume 2, 23.

Jiang, Y.; Zhu, Y.; Lan, C.; Yi, B.; Cui, Y.; and Guo, C. 2020. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*, 463–479.

Li, M.; Andersen, D. G.; Smola, A.; and Yu, K. 2014. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27.

Li, S.; Xue, F.; Baranwal, C.; Li, Y.; and You, Y. 2023. Sequence Parallelism: Long sequence training from system perspective. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2391–2404.

Lin, J.; Liu, Z.; You, Y.; Wang, J.; Zhang, W.; and Zhao, R. 2025. WeiPipe: Weight pipeline parallelism for communication-effective long-context large model training. In *Proceedings of the ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, 225–238.

Liu, W.; Li, M.; Tan, G.; and Jia, W. 2025. Mario: Near zero-cost activation checkpointing in pipeline parallelism. In *Proceedings of the ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, 197–211.

Micikevicius, P.; Narang, S.; Alben, J.; Diamos, G.; Elsen, E.; Garcia, D.; Ginsburg, B.; Houston, M.; Kuchaiev, O.; Venkatesh, G.; et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740*.

Narayanan, D.; Harlap, A.; Phanishayee, A.; Seshadri, V.; Devanur, N. R.; Ganger, G. R.; Gibbons, P. B.; and Zaharia, M. 2019. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the ACM Symposium on Operating Systems Principles*, 1–15.

Narayanan, D.; Phanishayee, A.; Shi, K.; Chen, X.; and Zaharia, M. 2021a. Memory-efficient pipeline-parallel DNN training. In *Proceedings of the International Conference on Machine Learning*, 7937–7947.

Narayanan, D.; Shoeybi, M.; Casper, J.; LeGresley, P.; Patwary, M.; Korthikanti, V.; Vainbrand, D.; Kashinkunti, P.; Bernauer, J.; Catanzaro, B.; et al. 2021b. 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*, 1–15.

Qi, P.; Wan, X.; Huang, G.; and Lin, M. 2024. Zero bubble (almost) pipeline parallelism. In *Proceedings of the International Conference on Learning Representations*.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140): 1–67.

Rajbhandari, S.; Rasley, J.; Ruwase, O.; and He, Y. 2020. ZeRO: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–16.

Rasley, J.; Rajbhandari, S.; Ruwase, O.; and He, Y. 2020. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 3505–3506.

Shah, A.; Chidambaram, V.; Cowan, M.; Maleki, S.; Musuvathi, M.; Mytkowicz, T.; Nelson, J.; Saarikivi, O.; and Singh, R. 2023. TACCL: Guiding collective algorithm synthesis using communication sketches. In *USENIX Symposium on Networked Systems Design and Implementation*, 593–612.

Shoeybi, M.; Patwary, M.; Puri, R.; LeGresley, P.; Casper, J.; and Catanzaro, B. 2019. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Soytürk, M. A.; Akhtar, P.; Tezcan, E.; and Unat, D. 2021. Monitoring collective communication among GPUs. In *European Conference on Parallel Processing*, 41–52.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Wang, B.; Xu, Q.; Bian, Z.; and You, Y. 2022. Tesseract: Parallelize the tensor parallelism efficiently. In *Proceedings of the International Conference on Parallel Processing*, 1–11.

Wang, W.; Khazraee, M.; Zhong, Z.; Ghobadi, M.; Jia, Z.; Mudigere, D.; Zhang, Y.; and Kewitsch, A. 2023. TopoOpt: Co-optimizing network topology and parallelization strategy

for distributed training jobs. In *USENIX Symposium on Networked Systems Design and Implementation*, 739–767.

Wu, B.; Liu, S.; Zhong, Y.; Sun, P.; Liu, X.; and Jin, X. 2024. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, 640–654.

Wu, H.; Chen, L.; and Yu, W. 2024. BitPipe: Bidirectional interleaved pipeline parallelism for accelerating large models training. *arXiv preprint arXiv:2410.19367*.

Yang, B.; Zhang, J.; Li, J.; Ré, C.; Aberger, C.; and De Sa, C. 2021. PipeMare: Asynchronous pipeline parallel DNN training. *Proceedings of Machine Learning and Systems*, 3: 269–296.

Zhao, Y.; Gu, A.; Varma, R.; Luo, L.; Huang, C.-C.; Xu, M.; Wright, L.; Shojanazeri, H.; Ott, M.; Shleifer, S.; et al. 2023. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*.

Zhou, X.; Chen, L.; and Wu, H. 2023. ABS-SGD: A delayed synchronous stochastic gradient descent algorithm with adaptive batch size for heterogeneous gpu clusters. *arXiv preprint arXiv:2308.15164*.