PIPEFSDP: Efficient Pipeline Parallel under Fully Sharded Data Parallel for Large Language Model Training

Xinglin Pan[1*] **Mingji Han**[2] **Penghao Zhao**[2] **Lin Zheng**[2] **Ray Ying**[2] **Key Zhang**[2]
Shaohuai Shi[3] **Xiaowen Chu**[1†]
[1] The Hong Kong University of Science and Technology (Guangzhou)
[2] Tencent Hunyuan
[3] Harbin Institute of Technology (Shenzhen)
xwchu@hkust-gz.edu.cn

Abstract

It is now a common practice to use PyTorch's official Fully Sharded Data Parallel (FSDP) framework to train large language models (LLMs) on large-scale GPU clusters, typically together with pipeline parallelism (PP) to achieve better performance. However, existing popular systems like Torchtitan with FSDP and PP are inefficient due to the large bubbles and peak memory footprint caused by suboptimal task scheduling. In this paper, we propose PIPEFSDP, an efficient training framework for FSDP combined with PP that alleviates pipeline bubbles and resource contention, thus improve training efficiency. Specifically, we design four fine-grained optimizations: dynamic context-aware prefetching, communication contention mitigation, redundant reshard elimination, and communication order optimization. In addition, we develop a heuristic model that selects the FSDP and PP degrees based on the given hardware and model configurations, thereby enhancing the practicality of PIPEFSDP. Empirical evaluations on both dense and sparse LLMs (Llama3, Qwen3, and Qwen3-MoE) across 32-GPU and 64-GPU clusters demonstrate that PIPEFSDP significantly outperforms the Torchtitan system by up to 1.28× in Model FLOPs Utilization (MFU) and 1.87× in memory consumption.

1 Introduction

The escalating scale of Large Language Models (LLMs), exemplified by models such as Qwen3 MAX [27] and KIMI-K2 [2] with up to trillions of parameters, has established Fully Sharded Data Parallelism (FSDP) [28] as a standard component in distributed training infrastructures. Its integration into major frameworks like AutoModel [1] and Megatron-LM [22] reflects its efficacy in sharding model states to reduce per-GPU memory pressure.

However, the scalability of pure FSDP is constrained by collective communication overhead as cluster sizes expand. Our profiling indicates that scaling the sharding degree from 8 to 64 reduces effective interconnect throughput from 165.05 GB/s to 145.06 GB/s, a 12.1% degradation that narrows the window for communication-computation overlap. Because the computation duration remains constant for a fixed micro-batch size, the diminishing throughput progressively results in synchronization time exceeding computation time, leading to GPU idle periods and hardware under-utilization.

To arrest the efficiency decay, a promising architectural shift is to decouple the sharding degree from the total cluster size. By integrating FSDP with Pipeline Parallelism (PP) [15, 16], the communication

*Work done during an internship at Tencent Hunyuan.
†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

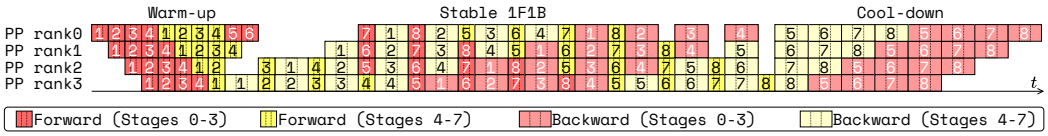


Figure 1: Illustration of the Interleaved 1F1B pipeline parallelism schedule. The model is divided into 8 stages, with each stage containing 3 layers. Each rank holds 2 stages. Each block is labeled with its corresponding micro-batch index (from MB #1 to MB #8).

domain can be restricted to localized sub-groups (e.g., 8 or 16), theoretically restoring the interconnect throughput to its peak values. Despite the theoretical potential of hybrid parallelism, existing implementations of FSDP and PP, such as those in Torchtitan [11], suffer from sub-optimal resource utilization due to a coarse-grained parameter lifecycle. These systems typically gather the full parameters for a pipeline stage during the initial micro-batch's forward pass and retain them until the final micro-batch's backward pass. Such eager synchronization escalates the peak memory footprint, as each GPU must concurrently host all parameters within its assigned pipeline stage. This intensified memory pressure necessitates a reduction in the micro-batch degree to maintain memory safety, which inherently amplifies pipeline bubbles and restricts the available computation volume for communication-computation overlapping. Consequently, the naive composition of FSDP and PP creates a systemic trade-off between memory efficiency and throughput, failing to achieve good scalability.

In this paper, we present PIPEFSDP, a high-performance framework that replaces coarse-grained parameter staging with a fine-grained parameter lifecycle. Unlike existing solutions that retain full stage parameters, PIPEFSDP immediately releases sharded parameters after their execution like ZeRO3 [19], significantly lowering the memory ceiling and enabling larger micro-batch sizes. To reconcile this aggressive memory reclamation with the need for high throughput, we introduce a suite of optimizations: (1) Dynamic Context-aware Prefetching adaptively aligns parameter fetching with the deterministic but non-monotonic layer-access order under PP; (2) Communication Contention Mitigation prioritizes latency-sensitive P2P transfers over collective operations to resolve bandwidth contention; (3) Redundant Reshard Elimination identifies and bypasses unnecessary sharding operations during contiguous execution phases; (4) Communication Order Optimization shifts communication load into underutilized bandwidth windows to maximize overlap during lightweight phases. By co-designing memory management with execution scheduling, PIPEFSDP effectively restores the communication-computation overlap that is typically lost in massive-scale hybrid parallelism.

Through the integration of the aforementioned scheduling optimizations, PIPEFSDP provides a scalable framework for large-scale LLM training. Beyond the architectural design, we use a Configurator that selects the FSDP and PP degrees before training using a communication-hiding heuristic. We evaluate PIPEFSDP on dense and sparse LLMs, including Llama3 [24], Qwen3 [27], and Qwen3-MoE [27], across 32-GPU and 64-GPU clusters. Experimental results demonstrate that PIPEFSDP consistently outperforms state-of-the-art frameworks, achieving up to 1.28× improvement in Model FLOPs Utilization (MFU) and reducing peak memory consumption by 1.87×.

2 Background

Pipeline Parallelism. Pipeline parallelism (PP) [15, 6] is a form of model parallelism that partitions a model along its depth. In this scheme, the model is divided into N_{pp} stages, where each stage consists of a contiguous set of layers hosted on a single device. By distributing layers this way, each device stores only a fraction of the total training state. To ensure workload balance and minimize idle time during micro-batch processing, these stages should be roughly equal in size.

Fig. 1 illustrates the timeline of Interleaved 1F1B [16], a scheduling strategy commonly used in LLM training. The local batch is split into N_{mb} micro-batches, which are executed concurrently across stages. This strategy reduces the pipeline bubble ratio and decreases idle time. To maximize hardware utilization, N_{mb} is typically set larger than N_{pp}, allowing each of the N_{pp} stages to process a distinct micro-batch concurrently and independently. The *pipeline bubbles* inevitably occur in pipeline parallelism as stages wait for dependencies from previous ranks. For example, as shown in Fig. 1, PP rank 3 cannot begin its computation until it receives the output from stage 2. Interleaved

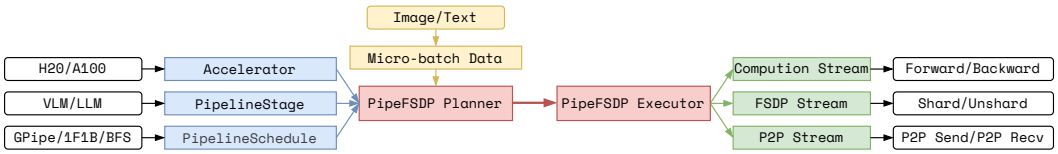


Figure 2: Overview of the PIPEFSDP framework.

1F1B introduces v virtual stages per PP rank, effectively partitioning the model into smaller segments and reduce the pipeline bubbles.

Fully Sharded Data Parallelism. Fully Sharded Data Parallelism (FSDP) partitions model states across a data-parallel group of size N_{fsdp}, sharding parameters, gradients, and optimizer states to reduce per-GPU memory overhead to $1/N_{fsdp}$. During execution, FSDP reconstructs required layers via all-gather collectives (unsharding) and immediately discards them after use (resharding) to maintain a minimal memory footprint. To hidden communication, a prefetching scheduler overlaps the all-gather of layer $i+1$ with the computation of layer i during the forward pass, and symmetrically fetches layer $i-1$ during the backward pass. In the backward pass, FSDP utilizes reduce-scatter to simultaneously synchronize and shard gradients, ensuring that each rank only retains the gradient portions corresponding to its local parameter shards. This cyclic orchestration of reconstruction, computation, and partitioning decouples peak memory consumption from the total model scale.

Motivation. Current integrations of FSDP and PP, such as Torchtitan [11], maintain full stage parameters throughout the pipeline execution cycle. This persistence creates significant memory pressure, which constrains the micro-batch degree. To stay within hardware memory limits, users must reduce the number of micro-batches, which extends the duration of pipeline bubbles and degrades overall throughput.

Conversely, an alternative strategy that releases parameters immediately after use to save memory introduces network contention. FSDP requires unsharding parameters and sharding gradients via All-Gather and Reduce-Scatter operations, which often coincide with PP's P2P activation transfers. Since P2P communication lies on the critical path, meaning subsequent steps cannot proceed without it, this resource competition can block the GPU and slow down the entire training process.

Because of these problems, we need a new framework that can save memory without blocking the important communication in the pipeline.

3 System Design

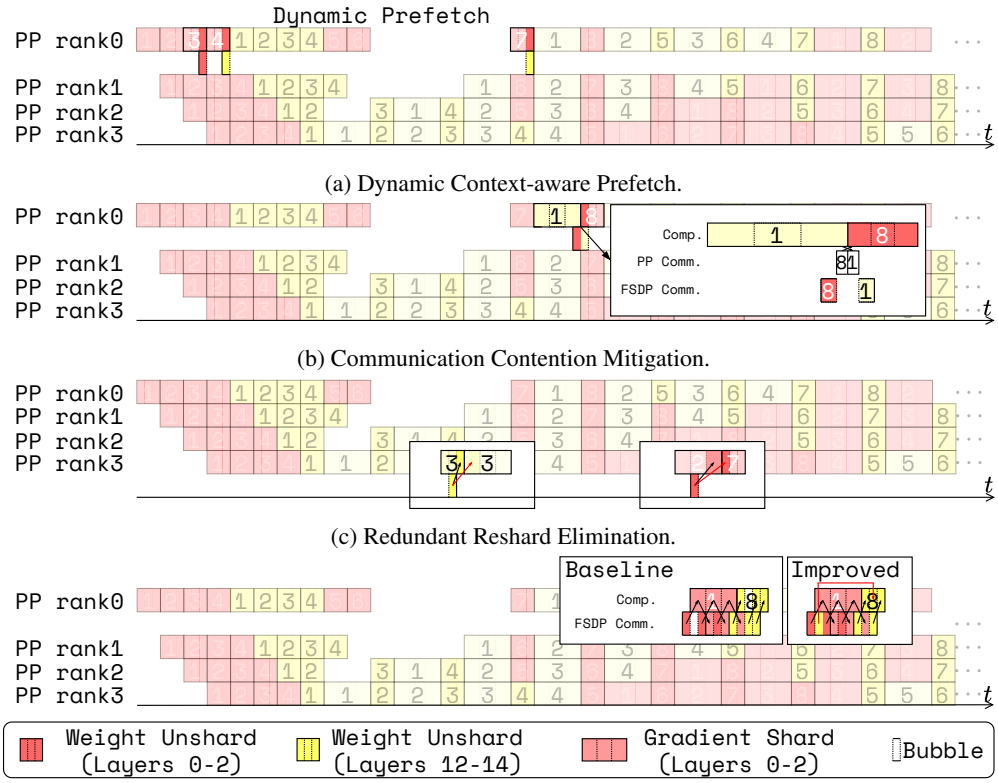
3.1 Overview of PIPEFSDP

Based on the observation above, we design and implement PIPEFSDP, comprising a static Configurator and a dynamic Executor. The Configurator selects the FSDP group size and the number of pipeline stages for a given model and GPU cluster, using the rule in Sec. 3.3. This configuration is then executed by the Executor, which orchestrates three parallel streams: the Computation Stream for forward and backward passes, the FSDP Stream for shard and unshard operations, and the P2P Stream for send and receive communications.

3.2 Basic Components

PIPEFSDP replaces stage-level parameter retention with immediate per-layer resharding to reduce parameter memory. When memory is the limiting factor, this can enable more micro-batches and reduce the structural pipeline bubble fraction. However, the additional FSDP communication can compete with critical-path P2P transfers and introduce communication stalls. The four optimizations below coordinate FSDP communication with the PP schedule to reduce these stalls while preserving the memory savings.

Dynamic Context-aware Prefetching Strategy. While FSDP effectively overlaps communication with computation through sequential prefetching (fetching layer $i+1$ during the execution of layer i), this static strategy can be further improved in settings that combine FSDP with PP. Specifically, in



(a) Dynamic Context-aware Prefetch.

(b) Communication Contention Mitigation.

(c) Redundant Reshard Elimination.

(d) Communication Order Optimization.

Figure 3: Basic Components in PIPEFSDP.

PP execution, the interleaving of forward and backward passes across multiple micro-batches results in a deterministic but non-monotonic layer-access sequence from the perspective of any single device. Static prefetching lacks the flexibility required in this context, as the next needed layer depends on the specific PP schedule (e.g., 1F1B) rather than a simple monotonic progression.

Fig. 3a illustrates a key scheduling challenge that motivates our dynamic prefetching design. For example, during the warmup phase, after completing the forward pass of micro-batch #3 at layer 2, the system immediately proceeds to compute the forward pass of micro-batch #4 at layer 0. In this scenario, a static prefetcher would erroneously attempt to fetch layer 3, whereas the system actually requires layer 0. The correct prefetch target thus changes dynamically with the schedule. This requirement persists into the stable phase, where forward and backward passes overlap. As shown in the figure, the forward pass of micro-batch #7 at layer 2 is immediately followed by the backward pass of micro-batch #1 at layer 14—a sudden switch that likewise demands on-demand fetching of the appropriate parameters. Backward prefetching follows a similar principle.

To realize this strategy, we introduce a lightweight, schedule-aware hook within the FSDP runtime. Specifically, we inject a custom hook just before the standard prefetching trigger. Because the pipeline execution schedule is deterministic, the precise order of layer accesses is known a priori. Here, *dynamic* refers to selecting the prefetch target according to the current position in this precomputed schedule, rather than changing the pipeline schedule online. We leverage this global knowledge to override the default prefetching behavior: instead of blindly fetching the next sequential layer, our hook queries the schedule to identify the true next required layer (e.g., wrapping around to layer 0). By dynamically aligning the prefetch target with the actual execution flow, we ensure that parameters are available exactly when needed, thereby maximizing the overlap between computation and communication.

Communication Contention Mitigation. We inject CUDA event-based synchronization barriers to dynamically coordinate bandwidth usage, ensuring that reduce-scatter operations yield to active P2P transfers. Crucially, rather than waiting for the entire iteration's P2P phase to complete, we trigger

the reduce-scatter operation immediately after the specific P2P input required by the corresponding layer is received—enabling earlier gradient reduction and significantly lowering peak memory consumption.

Redundant Reshard Elimination. Standard FSDP retains the full parameters of the final layer in memory after the forward pass, as they are immediately required for the backward pass. PIPEFSDP extends this optimization to the more complex execution schedules introduced by pipeline parallelism. Specifically, our scheduler identifies execution patterns within each pipeline stage where the forward pass of a layer is immediately followed by its corresponding backward pass. As illustrated in Fig. 3c, because the backward pass of micro-batch #3 directly follows its forward pass, the parameters can be reused in place, thereby eliminating unnecessary communication.

Furthermore, we generalize this principle to support Backward-to-Forward transitions, a scenario unique to interleaved PP schedules. As illustrated in Fig. 3c, the system detects when a forward pass (e.g., micro-batch #7) begins on the same layer (e.g., layer 9 in Stage 3) that has just completed a backward pass (e.g., micro-batch #2). Since backward execution terminates at the stage's entry layer, it naturally aligns with the starting point of the subsequent forward pass. In such cases, the unshard operations can be safely skipped. This optimization is enabled by a key insight: the sharding and resharding of model and optmizer states are agnostic to order of micro-batch. Unlike activations, which are data-dependent, the unsharded weights remain identical across micro-batches. Consequently, PIPEFSDP can reuse parameters between any contiguous operations on the same layer, independent of the order of micro-batch being processed.

Communication Order Optimization. The interleaved execution of Pipeline Parallelism introduces a unique "Backward-then-Forward" transition that exposes a critical performance bottleneck: insufficient computation time for communication hiding. In standard FSDP sequences (Forward-then-Backward), communication is naturally overlapped with the backward pass. Since the backward pass computes gradients for both weights and inputs, it is computationally intensive (approximately $2\times$ the FLOPs of the forward pass), providing a generous time window to hide the latency of prefetching parameters and reduce-scattering gradients. In contrast, the B-to-F transition in PP forces communication to be overlapped with the forward pass, which is computationally lightweight. More critically, during this forward pass, the system must handle two simultaneous communication burdens: the reduce-scatter of gradients from the preceding backward pass, and the prefetch of parameters for the next layer. The brief duration of forward computation is often insufficient to fully hide this combined communication load, resulting in exposed latency and pipeline stalls.

To further improve overlap efficiency, we adjust the execution order of collective communication operations. We observe that the initial step of the backward pass (processing the last model layer) has lower communication demand compared to subsequent steps. Specifically, since this is the starting point of the backward propagation, there are no gradients from preceding layers that require a reduce-scatter operation. This results in a brief window of available network bandwidth. We utilize this interval to preemptively fetch parameters for the subsequent forward stage. As shown in Fig. 3d, the forward pass of micro-batch #8 is scheduled to follow the backward pass of micro-batch #1. Instead of waiting for the forward pass to start, we trigger the parameter prefetching for layer 12 (required by MB #8) during the backward computation of layer 2 (MB #1). By shifting the communication load into this available window, we reduce the bandwidth contention during the computationally lighter forward pass.

3.3 Tuning FSDP and PP Parallelism Degrees

In this section, we analyze the trade-offs involved in configuring the degrees of FSDP (N_{fsdp}) and PP (N_{pp}), given a fixed total number of GPUs $N_{total} = N_{fsdp} \times N_{pp}$ and a fixed model size. Specifically, we examine three key dimensions: memory footprint, batch size constraints, and communication overhead, and derive a practical rule for selecting appropriate values of N_{fsdp} and N_{pp}.

State Sharding Invariance. Both FSDP and PP shard model and optimizer states, albeit in different ways. Pipeline Parallelism partitions distinct layers across multiple GPUs, whereas FSDP shards parameters and optimizer states along the data parallel dimension. The memory cost associated with storing model parameters and optimizer states remains constant. Because both strategies ensure non-redundant storage of model states, varying the allocation between N_{fsdp} and N_{pp} does not change the total memory cost of model and optimizer states across all GPUs.

Batch Size Constraints. Although pipeline parallelism (PP) reduces model and optimizer states memory overhead by partitioning model layers across GPUs. However, this intuition holds only with careful consideration of the constraints imposed on batch size. To mitigate pipeline bubbles, the number of micro-batches (N_{mb}) is typically greater than $\geq N_{pp}$. Consequently, the minimum global batch size is bounded below by $N_{mb} \times N_{fsdp}$. Therefore, the effective global batch size under PP+FSDP often exceeds that of a standard FSDP baseline, where the minimum batch size can theoretically be as small as N_{total}. Combining FSDP with PP does not reduce the *minimum* memory cost for training. Moreover, in large-batch scenarios ($N_{mb} \gg N_{pp}$), standard FSDP can employ gradient accumulation to match the same effective batch size while maintaining minimal per-iteration memory pressure.

Communication Overhead. To rigorously quantify the communication overhead between different FSDP scales (e.g., $N_{fsdp} = 32$ vs. $N_{fsdp} = 64$) in an inter-node setting, we adopt a standard cost model for Ring-based All-Gather as an illustrative example.

Let x denote the total number of parameters to be gathered, B the network bandwidth, and α the point-to-point network latency (i.e., startup time). The completion time T_{ring} is modeled as:

$$T_{ring} = \underbrace{(N_{fsdp} - 1) \cdot \alpha}_{\text{Latency Term}} + \underbrace{\frac{N_{fsdp} - 1}{N_{fsdp}} \cdot \frac{x}{B}}_{\text{Bandwidth Term}}. \tag{1}$$

The bandwidth term $\frac{N_{fsdp}-1}{N_{fsdp}} \frac{x}{B}$ captures the transmission cost. As N_{fsdp} increases, the factor $\frac{N_{fsdp}-1}{N_{fsdp}}$ rapidly converges to 1 (e.g., $31/32 \approx 0.97$ versus $63/64 \approx 0.98$). Consequently, in large-scale clusters, the bandwidth cost becomes effectively constant and independent of the FSDP group size N_{fsdp}. Thus, doubling the group size from 32 to 64 yields negligible gains in per-link data transmission efficiency.

In contrast, the latency term $(N_{fsdp} - 1)\alpha$ scales linearly with the group size. In a ring topology, each data chunk must traverse $N_{fsdp} - 1$ hops, so increasing the group size directly increases synchronization latency. For instance, moving from $N_{fsdp} = 32$ to $N_{fsdp} = 64$ approximately doubles the cumulative latency. When message sizes are small, this linear growth in latency becomes the dominant performance bottleneck.

Optimization Strategy. Because static memory consumption is invariant to how the model is partitioned across these dimensions, we prioritize throughput over memory capacity constraints. The rule is as follows: if the FSDP communication time t_{comm}^{fsdp} does not exceed the computation time t_{comp}, FSDP can be favored to reduce the pipeline bubble and thus maximize throughput. Otherwise, the Configurator recommends a configuration by selecting the largest FSDP parallelism degree N_{fsdp} that satisfies the communication-hiding constraint $t_{comm}^{fsdp} \leq t_{comp}$, thereby achieving the shortest forward time while still minimizing the bubble. This rule can be directly applied to automate parallelism configuration.

Calibration and One-time Overhead. The Configurator measures computation and communication times before training. We run 50–60 training steps to obtain stable per-layer computation times, excluding communication. This calibration takes approximately 15 minutes, including checkpoint initialization and data loading. We then measure all-gather time with a microbenchmark for each candidate FSDP degree; each candidate takes less than one minute. The one-time calibration cost is therefore approximately 15 minutes plus less than one minute per candidate. Calibration is repeated when the model or hardware configuration changes.

Tab. 1 lists PIPEFSDP alongside existing parallelization strategies by quantifying how model states are mapped onto the N_{total} GPUs. Compared to standard PP, PIPEFSDP reduces the pipeline bubble from $(N_{total} - 1)$ to $(N_{pp} - 1)$. When PP is combined with data parallelism (DP, e.g., ZeRO-1), each node must store a full replica of the parameters for its assigned pipeline stage, requiring M_{total}/N_{pp} memory per node, where M_{total} denotes the total memory footprint of the model parameters. In contrast, PIPEFSDP eliminates this replication through its unified device view, reducing per-node parameter memory to the theoretical minimum of M_{total}/N_{total}.

Compared to pure FSDP with configuration $(1, N_{total})$, PIPEFSDP groups devices into N_{fsdp} sub-groups. This design partitions the global communication cost t_{comm}^{total} (both all-gather and reduce-scatter operations) into localized communications t_{comm}^{fsdp} within each sub-group. As a result,

Method ($N_{pp}, N_{dp/fsdp}$)	Parameter Memory	Forward Time	Bubble	Activation Memory	P2P Communication
Standard PP $(N_{total}, 1)$	M_{total}/N_{total}	t_{comp}	$(N_{total}-1)$	$N_{total} \times$ act.	$N_{mb} \times 2(N_{total}-1) \times$ act.
PP with DP (N_{pp}, N_{dp})	M_{total}/N_{pp}	t_{comp}	$(N_{pp}-1)$	$N_{total} \times$ act.	$N_{mb} \times N_{dp} \times 2(N_{pp}-1) \times$ act.
Standard FSDP $(1, N_{total})$	M_{total}/N_{total}	$\max(t_{comp}, t_{\text{comm}}^{total})$	0	$N_{mb} \times$ act.	0
PIPEFSDP (N_{pp}, N_{fsdp})	M_{total}/N_{total}	$\max(t_{comp}, t_{comm}^{fsdp})$	$(N_{pp}-1)$	$N_{total} \times$ act.	$N_{mb} \times 2(N_{pp}-1) \times$ act.

Table 1: Comparison of parallel training methods.

synchronization latency is minimized, and the effective execution time per training step converges to $\max(t_{comp}, t_{comm}^{fsdp})$. Thus, by strategically combining pipeline and fine-grained data parallelism, PIPEFSDP simultaneously achieves near-optimal memory efficiency and reduced communication overhead.

4 Implementation

In this section, we briefly discuss the implmentation details of PIPEFSDP.

Eliminate Unstable Memory Peak. Standard FSDP2 uses asynchronous CUDA streams to overlap computation and communication. However, although the CPU issues resharding (freeing current layer) and unsharding (allocating next layer) kernels sequentially, their GPU execution can overlap. This may start allocation for the next layer before the previous layer is fully deallocated, causing transient memory spikes and out-of-memory errors. We eliminate this instability by inserting CUDA event barriers that force resharding to complete before unsharding begins. This serialization is applied in both forward and backward passes, yielding a stable, predictable memory footprint.

Avoid FSDP Sharding for MoE. Expert Parallelism (EP) adds dispatch/combine communication overhead; we evaluate configurations both with and without EP. When EP is disabled, multiple experts reside on one device. We use a grouped GEMM operator to fuse disjoint expert computations into a single kernel, mitigating load imbalance and reducing kernel launch overhead. When EP is enabled, decoupled dispatch/combine can delay the MoE layer and create cascading bottlenecks. To avoid this, we align the FSDP sharding degree with the EP degree so that expert parameters are already sharded by EP, removing the need for separate all-gather and reduce-scatter in FSDP synchronization.

An alternative is to use the `exclude_parameters` option in `fully_shard`; this applies EP to the MoE module without additional FSDP sharding and prevents a single Transformer layer from being split into two FSDP submodules (which would require two all-gather operations).

Finally, PIPEFSDP overlaps attention computation with the subsequent all-gather in the forward pass. In the backward pass, it defers the all-gather of attention parameters and the reduce-scatter of gradients to the attention backward phase, shifting collective communication away from the MoE layer and alleviating bottlenecks.

Explicit Unsharding of the Initial Layer. All GPUs except those in the first pipeline stage experience idle bubble time before the warm-up phase. PIPEFSDP exploits this slack by explicitly unsharding the first layer's parameters in advance, so that the first forward pass is ready to execute without delay.

5 Experiments

Testbeds. To evaluate the performance of PIPEFSDP, we conduct experiments on two GPU clusters. Testbed A consists of four nodes, each equipped with eight NVIDIA H20 GPUs (32 GPUs total), while Testbed B comprises eight nodes with eight NVIDIA H800 GPUs each (64 GPUs total). All nodes within each cluster are interconnected via InfiniBand. The software environment includes CUDA 12.9.1, PyTorch 2.9.1, and Python 3.10. We implement our system on top of Torchtitan [11].

	Testbed A					Testbed B				
		Torchtitan		PIPEFSDP			Torchtitan		PIPEFSDP	
Backbone	N_{pp}	**TPS** ↑	**Memory(GiB)** ↓	**TPS** ↑	**Memory(GiB)** ↓	N_{pp}	**TPS** ↑	**Memory(GiB)** ↓	**TPS** ↑	**Memory(GiB)** ↓
Llama3	2	480.14	82.78	516.02	93.01	4	1531.49	73.32	1929.07	67.76
	4	501.77	80.92	564.27	86.86	8	1287.37	74.20	1517.90	68.95
Qwen3	2	414.90	85.31	446.61	68.00	4	1364.83	74.12	1562.29	69.27
	4	416.51	86.14	424.15	61.62	8	1224.29	71.00	1542.43	75.02
Qwen3-MoE	2	1975.14	81.25	2532.52	91.80	4	4296.86	70.94	4970.73	69.75
	4	2254.57	79.49	2413.08	82.32	8	3568.95	70.78	4047.27	72.16

Table 2: TPS comparison between PIPEFSDP and Torchtitan across diverse architectures and hardware environments.

	Testbed A					Testbed B				
		Torchtitan		PIPEFSDP			Torchtitan		PIPEFSDP	
Backbone	N_{pp}	**TPS** ↑	**Memory(GiB)** ↓	**TPS** ↑	**Memory(GiB)** ↓	N_{pp}	**TPS** ↑	**Memory(GiB)** ↓	**TPS** ↑	**Memory(GiB)** ↓
Llama3	2	424.7	81.39	438.9	48.64	4	1570.2	50.14	1881.3	42.39
	4	396.4	57.76	407.9	45.12	8	1509.7	42.93	1844.7	37.52
Qwen3	2	341.9	83.55	349.8	44.66	4	1287.1	50.48	1397.5	33.18
	4	315.7	57.45	318.8	35.78	8	1262.4	43.34	1414.7	36.86
Qwen3-MoE	2	226.3	80.63	250.3	43.62	4	316.7	46.75	372.0	32.81
	4	214.2	52.62	230.2	39.38	8	306.0	39.40	375.4	34.65

Table 3: Memory comparison between PIPEFSDP and Torchtitan across diverse architectures and hardware environments.

Settings. We employ the Adam optimizer with gradient clipping and enable mixed-precision training [14]. Under this configuration, model parameters, gradients, and optimizer states are maintained in reduced precision, along with an additional FP32 master copy to ensure numerical stability. Memory consumption is profiled using the PyTorch CUDA memory management API (*torch.cuda.memory*). The reported peak activation memory corresponds to the maximum value observed across all devices during a full training iteration, including both forward and backward passes. Throughput is measured using using Token Per Second (TPS) and Model FLOPs Utilization (MFU), computed according to the standard methodology described in Narayanan et al. [16]. We fix the sequence length to 4096 on both Testbed A and Testbed B. While we adopt Interleaved 1F1B [16] as the default pipeline schedule, our framework seamlessly supports alternative schedules such as BFS [8] to accommodate diverse model architectures and network topologies.

		Standard FSDP		PIPEFSDP	
Top_k	N_{total}	**MFU** ↑	**Memory(GiB)** ↓	**MFU** ↑	**Memory(GiB)** ↓
2	32	3.29%	49.39	4.73%	66.58
	64	3.25%	41.09	5.93%	51.20
4	32	4.08%	49.39	5.74%	73.14
	64	4.01%	41.09	7.12%	50.68
8	32	5.81%	49.53	7.74%	72.62
	64	5.72%	40.70	9.16%	50.68
16	32	9.19%	50.41	10.60%	74.46
	64	9.08%	42.11	11.72%	50.21
32	32	14.10%	50.77	14.03%	73.72
	64	14.26%	42.46	14.09%	49.71

Table 4: Performance comparison between PIPEFSDP and Standard FSDP using Qwen3-MoE across diverse node configurations and Top_k settings on Testbed B.

Models. We evaluate our proposed system on transformer-based architectures following the designs of Llama3 [24] and Qwen3-Dense and -MoE [27]. To mitigate the well-known pipeline imbalance introduced by the embedding and output layers, we adopt the established practice of introducing a virtual pipeline stage dedicated to these components, consistent with prior work [26, 4, 24]. The Llama3 dense model is implemented by scaling down Llama3-70B from 80 layers to 30 layers, while the Qwen dense and MoE models follow their original specifications.

5.1 TPS Comparison

Tab. 2 summarizes the TPS performance across various architectures and hardware configurations, where we meticulously tuned all hyperparameters for both frameworks to maximize throughput provided they remain within hardware memory limits. PIPEFSDP consistently achieves higher TPS than the baseline in all evaluated scenarios. Notably, for the Llama3 backbone, our framework improves TPS from 1531.49 to 1929.07 on Testbed B, representing a 1.26× throughput gain. For

communication-intensive workloads like Qwen3-MoE, PIPEFSDP maintains a performance lead, outperforming the state-of-the-art by up to 1.28×. These empirical results validate that our fine-grained scheduling and sharding optimizations effectively translate into practical speedup across both dense and MoE-based architectures.

5.2 Memory Comparison

Tab. 3 presents the peak memory consumption across diverse architectures and hardware environments, with the micro-batch size fixed to one. PIPEFSDP consistently demonstrates a significantly lower memory footprint compared to the baseline. This advantage is particularly pronounced in configurations with low pipeline degrees. Notably, for the Qwen3 backbone on Testbed A, our framework reduces peak memory usage from 83.55 GiB to 44.66 GiB, achieving a 1.87× reduction. These results confirm that PIPEFSDP provides substantial headroom for scaling to larger models or increasing the micro-batch size to improve throughput.

5.3 Comparison with Torchtitan pure 1D FSDP

Tab. 4 compares standard FSDP with the FSDP+PP hybrid paradigm using Qwen3-MoE-235-A22B (94 to 14 experts), a micro-batch size of 1, and a sequence length of 2048 on Testbed B. Experiments vary the total node count ($N_{total} = 32$ or 64) and the expert Top_k value (2, 4, 8, 16, 32) to modulate the computation-to-communication ratio. Top_k denotes the number of active experts selected for each token.

As analyzed in Sec. 3, the FSDP+PP baseline increases memory usage, aligning with theoretical expectation. The efficiency of the hybrid approach depends on the ratio $t^{total}_{comm}/t^{fsdp}_{comm}$. At $N_{total} = 32$, this ratio is modest, yielding conservative gains. At $N_{total} = 64$, the communication overhead dominates, allowing the hybrid strategy to better demonstrate its scalability advantage. However, when $t_{comp} \geq t^{total}_{comm}$ (e.g., at $Top_k = 32$), pure FSDP becomes more efficient, as its computation can more fully overlap with the lighter communication.

5.4 Comparison with Megatron-FSDP

Megatron-FSDP [22] optimizes individual communication operations, whereas PIPEFSDP coordinates these operations with the pipeline schedule. Megatron-FSDP supports zero-copy all-gather through NCCL user-buffer registration, native FP8 parameter communication, fused gradient reduce-scatter with scaling, persistent double-buffered memory pools, and coalesced NCCL kernel launches. These mechanisms reduce communication overhead within each operation. PIPEFSDP complements them by controlling which operations execute and when they execute.

PIPEFSDP addresses four scheduling opportunities that Megatron-FSDP leaves open. Megatron-FSDP prefetches sequential buckets and stops at chunk boundaries; DCP instead queries the pre-computed pipeline schedule for the next required layer. Megatron-FSDP also runs P2P transfers and FSDP collectives independently. CCM coordinates these streams: it delays reduce-scatter until the corresponding P2P receive completes to protect critical-path transfers. Megatron-FSDP skips resharding during activation recomputation; RRE additionally reuses full parameters across adjacent forward–backward or backward–forward tasks on the same layer. Finally, COO moves upcoming all-gathers into the underutilized bandwidth window at backward entry. Together, our four mechanisms coordinate prefetching, parameter reuse, and communication timing around the PP schedule.

We integrate these mechanisms into Megatron-FSDP to test their benefit beyond its existing communication optimizations. We align parameter bucket boundaries with individual layers to support per-layer scheduling. On Testbed A with Qwen3-MoE and $N_{pp} = 2$, PIPEFSDP increases throughput from 1,191 to 1,227 TPS (3.0%). Peak memory remains unchanged. Megatron-FSDP preallocates its memory buffers, so the scheduling changes do not reduce the reported peak footprint. PIPEFSDP thus improves throughput within this Megatron-FSDP configuration without increasing peak memory.

5.5 Ablation Study

We isolate weight sharding and each optimization in Tab. 5. Weight sharding alone reduces peak memory from 52.62 to 44.53 GiB but lowers throughput from 1,583 to 1,548 TPS. DCP further

Weight Sharding	DCP	RRE	COO	CCM	TPS ↑	Memory (GiB) ↓
					1,583	52.62
✓					1,548	44.53
Individual optimizations						
✓	✓				1,547	**29.37**
✓		✓			1,583	44.53
✓			✓		1,622	44.53
✓				✓	1,564	44.53
Cumulative optimizations						
✓	✓	✓			1,594	**29.37**
✓	✓	✓	✓		1,689	37.27
✓	✓	✓	✓	✓	**1,701**	39.38

Table 5: Ablation study on Qwen3-MoE using Testbed A. Weight Sharding partitions model weights across the FSDP group. DCP: Dynamic Context-aware Prefetching; RRE: Redundant Reshard Elimination; COO: Communication Order Optimization; CCM: Communication Contention Mitigation. The DCP-only row also starts the cumulative comparison.

reduces memory to 29.37 GiB with negligible throughput change. RRE, COO, and CCM individually improve throughput without increasing peak memory. DCP thus primarily saves memory; the other components improve throughput.

We then combine the components to measure their cumulative benefit. The full system reaches 1,701 TPS with 39.38 GiB peak memory, improving throughput by 7.5% and reducing memory by 25.2% relative to the baseline without weight sharding. PIPEFSDP trades some of DCP's memory savings for higher throughput while retaining a lower peak footprint than either unoptimized baseline.

6 Related Work

A number of research efforts (e.g., Interleaved-1F1B [16], Zero Bubble [18], DualPipe [4], Breadth-First Pipeline Parallelism [8], DAPPLE [5], PipeDream [15], ZeroPP [23], Chimera [9], BPipe [7], AutoPipe [12] and PipeOffload [26]) have sought to enhance PP's efficiency and reduce its memory overhead [17, 3, 25].

FSDP [28] lowers the memory footprint per GPU by sharding model parameters, gradients, and optimizer states across all GPUs, following the principle of the Zero Redundancy Optimizer (ZeRO) [19–21]. Recent works, including Torchtitan [11], VeOmni [13], veScale [10], AutoModel [1], and Megatron-LM [22], have built native PyTorch LLM training systems based on FSDP. Current training frameworks primarily focus on optimizing kernels. However, how to effectively combine FSDP with PP, and the resulting trade-offs, remain underexplored.

7 Limitations

PIPEFSDP has three main limitations, which we plan to address. First, we evaluate only the 1F1B schedule because PyTorch lacks a native zero-bubble pipeline (ZeroPP) implementation; we will build a toy example to show potential gains. Second, the system is not yet ported to Megatron-FSDP, as we prioritized a clean native-FSDP prototype; this port and a performance comparison are planned. Third, all experiments are on NVIDIA GPUs; we will extend to other accelerators (e.g., Ascend NPUs) to verify generality.

8 Conclusion

In this work, we present PIPEFSDP, a system that implements a detailed integration of FSDP with PP atop the Torchtitan framework. We further analyze the performance characteristics and inherent trade-offs of combining FSDP with PP in comparison to other parallelism strategies. Evaluations conducted on LLMs across two distinct testbeds show that PIPEFSDP improves MFU by up to 1.28× and reduces peak memory consumption by up to 1.87× compared to existing frameworks.

Acknowledgments and Disclosure of Funding

This work was partial support from the National Natural Science Foundation of China (Grant No. 62272122), the Guangzhou Municipal Joint Funding Project with Universities and Enterprises (Grant No. 2024A03J0616), Guangzhou Municipality Big Data Intelligence Key Lab (2023A03J0012), and Hong Kong CRF (Grant Nos. C7004-22G and C6015-23G).

References

[1] Nemo automodel: Dtensor-native spmd library for scalable and efficient training. `https://github.com/NVIDIA-NeMo/Automodel`, 2025. GitHub repository.

[2] Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, and Haiming Wang. Kimi K2: open agentic intelligence. *CoRR*, abs/2507.20534, 2025.

[3] Weiwei Chu, Xinfeng Xie, Jiecao Yu, Jie Wang, Amar Phanishayee, Chunqiang Tang, Yuchen Hao, Jianyu Huang, Mustafa Ozdal, Jun Wang, Vedanuj Goswami, Naman Goyal, Abhishek Kadian, Andrew Gu, Chris Cai, Feng Tian, Xiaodong Wang, Min Si, Pavan Balaji, Ching-Hsiang Chu, and Jongsoo Park. Scaling llama 3 training with efficient parallelism strategies. In *ISCA*, pages 1703–1716. ACM, 2025.

[4] DeepSeek-AI. Deepseek-v3 technical report. *CoRR*, abs/2412.19437, 2024.

[5] 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. DAPPLE: a pipelined data parallel approach for training large models. In *PPoPP*, pages 431–445. ACM, 2021.

[6] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, pages 103–112, 2019.

[7] Taebum Kim, Hyoungjoo Kim, Gyeong-In Yu, and Byung-Gon Chun. Bpipe: Memory-balanced pipeline parallelism for training large language models. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 16639–16653. PMLR, 2023.

[8] Joel Lamy-Poirier. Breadth-first pipeline parallelism. In *MLSys*. mlsys.org, 2023.

[9] Shigang Li and Torsten Hoefler. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *SC*, page 27. ACM, 2021.

[10] Youjie Li, Cheng Wan, Zhiqi Lin, Hongyu Zhu, Jiacheng Yang, Ziang Song, Xinyi Di, Jiawei Wu, Huiyao Shu, Wenlei Bao, Yanghua Peng, Haibin Lin, and Li-Wen Chang. vescale: Consistent and efficient tensor programming with eager-mode SPMD. *CoRR*, abs/2509.07003, 2025.

[11] Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, Sanket Purandare, Gokul Nadathur, and Stratos Idreos. Torchtitan: One-stop pytorch native solution for production ready LLM pretraining. In *ICLR*. OpenReview.net, 2025.

[12] Weijie Liu, Zhiquan Lai, Shengwei Li, Yabo Duan, Keshi Ge, and Dongsheng Li. Autopipe: A fast pipeline parallelism approach with balanced partitioning and micro-batch slicing. In *CLUSTER*, pages 301–312. IEEE, 2022.

[13] Qianli Ma, Yaowei Zheng, Zhelun Shi, Zhongkai Zhao, Bin Jia, Ziyue Huang, Zhiqi Lin, Youjie Li, Jiacheng Yang, Yanghua Peng, Zhi Zhang, and Xin Liu. Veomni: Scaling any modality model training with model-centric distributed recipe zoo. *CoRR*, abs/2508.02317, 2025.

[14] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *ICLR (Poster)*. OpenReview.net, 2018.

[15] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for DNN training. In *SOSP*, pages 1–15. ACM, 2019.

[16] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using megatron-lm. In *SC*, page 58. ACM, 2021.

[17] Penghui Qi, Xinyi Wan, Nyamdavaa Amar, and Min Lin. Pipeline parallelism with controllable memory. In *NeurIPS*, 2024.

[18] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble (almost) pipeline parallelism. In *ICLR*. OpenReview.net, 2024.

[19] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In *SC*, page 20. IEEE/ACM, 2020.

[20] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. Zero-infinity: breaking the GPU memory wall for extreme scale deep learning. In *SC*, page 59. ACM, 2021.

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

[22] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.

[23] Ding Tang, Lijuan Jiang, Jiecheng Zhou, Minxi Jin, Hengjie Li, Xingcheng Zhang, Zhilin Pei, and Jidong Zhai. ZeroPP: unleashing exceptional parallelism efficiency through tensor-parallelism-free methodology. *arXiv preprint arXiv:2402.03791*, 2024.

[24] Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

[25] Yeung Man Tsung, Penghui Qi, Min Lin, and Xinyi Wan. Balancing pipeline parallelism with vocabulary parallelism. In *MLSys*. mlsys.org, 2025.

[26] Xinyi Wan, Penghui Qi, Guangxing Huang, Min Lin, and Jialin Li. Pipeoffload: Improving scalability of pipeline parallelism with memory optimization. In *ICML*. OpenReview.net, 2025.

[27] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jian Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025.

[28] 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. Pytorch FSDP: experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, 16(12):3848–3860, 2023.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction were clearly state the claims made, including the contributions made in the paper and important assumptions and limitations.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations of PIPEFSDP in the paper.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The paper provided the full set of assumptions and a complete (and correct) proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have included all necessary details to ensure the reproducibility of our results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Due to company policy, the code cannot be released at submission time, but we are committed to releasing it upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer:[Yes]

   Justification: The experimental setting was presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Standard deviations are not reported for the full-scale model training due to the extreme computational cost.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have read and ensured that our research fully conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification:

    - Llama3, Meta Llama 3 Community License (`https://www.llama.com/llama3/license/`).
    - Qwen3, Apache 2.0 (`https://github.com/QwenLM/Qwen3`).
    - Qwen3-MoE, Apache 2.0.
    - torchtitan, BSD 3-Clause License (`https://github.com/pytorch/torchtitan`).

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Due to company policy, the code cannot be released at submission time, but we are committed to releasing it upon acceptance.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.