Synergistic Tensor and Pipeline Parallelism

Mengshi Qi[*] [1], Jiaxuan Peng[1], Jie Zhang, Juan Zhu, Yong Li, Huadong Ma[1]
[1]State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, China
{qms, pjx, mhd}@bupt.edu.cn

Abstract

In the machine learning system, the hybrid model parallelism combining tensor parallelism (TP) and pipeline parallelism (PP) has become the dominant solution for distributed training of Large Language Models (LLMs) and Multimodal LLMs (MLLMs). However, TP introduces significant collective communication overheads, while PP suffers from synchronization inefficiencies such as pipeline bubbles. Existing works primarily address these challenges from isolated perspectives, focusing either on overlapping TP communication or on flexible PP scheduling to mitigate pipeline bubbles. In this paper, we propose a new synergistic tensor and pipeline parallelism schedule that simultaneously reduces both types of bubbles. Our proposed schedule decouples the forward and backward passes in PP into fine-grained computation units, which are then braided to form a composite computation sequence. This compositional structure enables near-complete elimination of TP-related bubbles. Building upon this structure, we further design the PP schedule to minimize PP bubbles. Experimental results demonstrate that our approach improves training throughput by up to 12% for LLMs and 16% for MLLMs compared to existing scheduling methods. Our source code is avaiable at https://github.com/MICLAB-BUPT/STP.

1 Introduction

Distributed systems [6, 23, 38, 11, 31] have become the cornerstone for the training of large-scale machine learning models [1, 35, 28, 4, 19, 26, 27, 20], particularly since model scales have grown to tens of billions of parameters. Data parallelism [12, 15, 5] is commonly employed to accelerate training and is most effective for relatively smaller models. As the scale of the model increases, researchers have turned to model parallelism [10, 32, 21], which partitions the model into smaller components and distributes them between multiple devices. Model parallelism primarily consists of two strategies: tensor parallelism (TP) [33] and pipeline parallelism (PP) [7, 23, 30]. Tensor parallelism partitions model weights across devices according to predefined rules, thereby reducing per-device memory requirements. While TP alleviates GPU memory pressure to some extent, it introduces additional collective communication operations within TP groups, referred to as TP bubbles that exhibit significant growth with increasing TP size, as shown in Figure 1. For instance, the TP bubbles account for 27.5% of the overall time in the configuration of TP=8 and sequence length of 6,144, which can significantly degrade the performance. In contrast, pipeline parallelism divides the model into chunks along the layer dimension, which are assigned to PP stages according to the PP strategy. Activations and gradients are then transmitted among PP stages during the forward and backward passes. This strategy enables efficient utilization of computational resources but requires careful synchronization between stages to minimize idle time, commonly referred to as PP bubbles.

To mitigate TP bubble issues, existing studies [37, 8, 2, 25] aim to overlap communication with computation at the kernel level, such as customized CUDA kernels [2], or at the hardware level,

[*]Corresponding author: qms@bupt.edu.cn.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

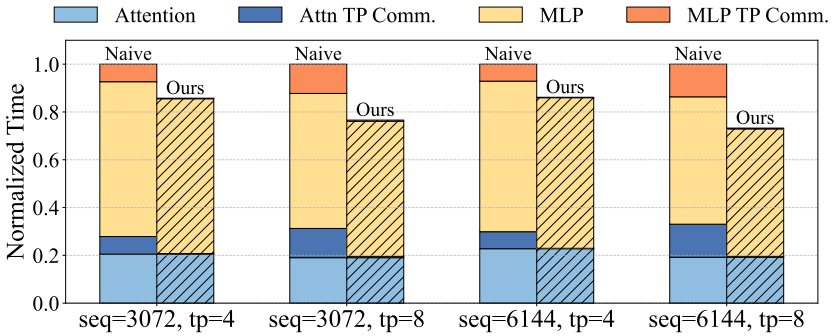


Figure 1: Speedup of overlapping TP communication with computation in PP within a Transformer layer of Qwen2 [39] in forward pass. The proportion of TP communications grows significantly with the increased TP size, which are effectively overlapped in our schedule compared with the naive implementation.

where specialized hardware mechanisms like the track-and-trigger system [25] are employed. By carefully scheduling paired decomposed communication and computation chunks, these methods effectively reduce overall execution time. As for PP bubbles, GPipe [7] reduces pipeline bubbles by splitting the global batch into microbatches. Subsequently, 1F1B [22] improves upon GPipe by scheduling backward passes earlier, thereby optimizing peak memory footprint. Furthermore, 1F1B-I [23] reduces the bubble rate by introducing virtual stages assigned to individual devices. More recently, Zero Bubble [30] decouples the full backward pass into separate activation and weight gradient computations, enabling more flexible scheduling and achieving a significantly lower PP bubble rate.

In practice, hybrid model parallelism, particularly the combination of PP and TP, has become the dominant approach for large-scale model training. However, most of the aforementioned works primarily focus on optimization from isolated perspectives, generally yielding limited enhancements. Inspired by studies [17, 14] that integrate two kinds of parallelism strategies for better optimization, our work aims to integrate PP and TP from a synergistic perspective. Specifically, our method leverages the abundant computation operations inherent in PP to overlap the TP bubbles and employs precise scheduling to mitigate the PP bubbles. Firstly, we decouple the forward and backward passes of the model chunk within each stage of PP into fine-grained computation units. By interleaving forward and backward computation units to construct execution blocks, we achieve near-zero TP idle time as shown in Figure 1, where TP communication and the fine-grained computation units in PP execute in parallel. Building upon these blocks, we further design a PP schedule with a low bubble rate and balanced memory footprints across stages. Additionally, we provide an enhanced variant of our schedule that incorporates activation offloading, serving as a preliminary attempt under limited memory conditions. Our main contributions are summarized as follows:

- We rethink hybrid model parallelism from a synergistic perspective and decouple the forward and backward passes of PP into fine-grained computation units, which are interleaved together to form braided execution blocks that effectively eliminate TP communication bubbles at the scheduling level.
- We design a novel PP schedule that synergistically integrates with TP based on the braided execution blocks. This schedule features a "V"-shape dataflow that achieves balanced memory footprints and a significantly reduced PP bubble rate.
- We conduct extensive experiments on popular LLMs and MLLMs of various scales to validate the universality and effectiveness of our method. The results demonstrate that our proposed schedule improves throughput by up to 16% compared to the existing methods.

2 Related Work

Tensor Parallelism. Tensor parallelism [33, 23, 14, 37, 8, 2] partitions tensors across devices to reduce per-device memory usage but introduces communication overhead due to synchronization requirements. This problem becomes more pronounced with TP sizes increase, where the limited

inter-device bandwidth (*e.g.*, systems without NVLink [24]) can severely degrade training efficiency. Several prior works [37, 8, 2] optimize this kind of collective communication by breaking down both communication and corresponding computation operations into sequences of smaller chunks. Subsequently, they carefully coordinate the execution of these granular units to achieve acceleration, often through the intricate fusion of computation and communication kernels, along with complex hardware control. In contrast, our proposed schedule optimizes TP communication at the scheduling or software level, providing a more accessible and user-friendly approach.

Pipeline Parallelism. Pipeline parallelism [16, 11, 34, 9, 18, 13, 17] divides the entire model into smaller chunks using a variety of strategies. GPipe [7] reduces PP bubbles by splitting the global batch into smaller micro-batches. Building on this, PipeDream [22] improves upon GPipe by advancing the backward computation to further reduce memory usage. 1F1B-I [23] and Hanayo [18] further introduce additional virtual stages to lower the bubble rate. More recently, Zero Bubble [30] decouples the backward computation into activation gradient and weight gradient computation to achieve a near-zero PP bubble schedule. However, these existing schedules tend to address PP bubbles in isolation. In this paper, we design an integrated approach that concurrently minimizes both TP and PP bubbles by synergistically combining these two parallelism techniques.

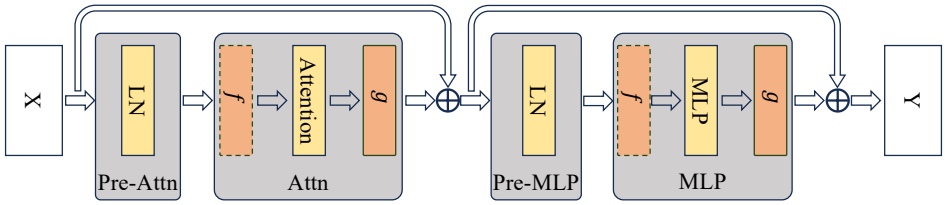
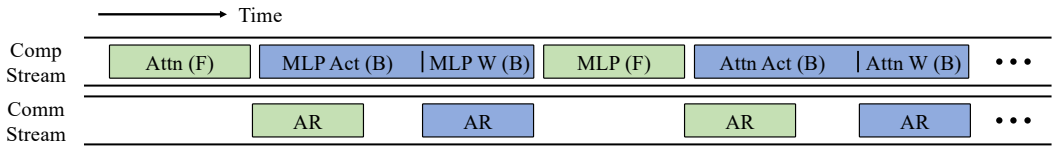
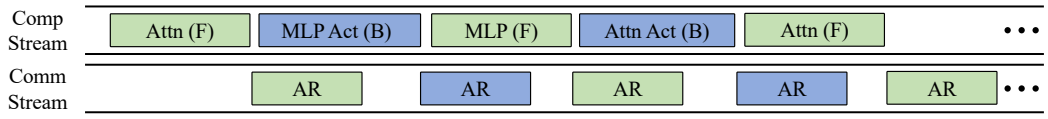


Figure 2: Illustration of the main computation and communication operation in a single TP Transformer layer. f is an identity operation in forward while All-Reduce in backward. g is opposite to f.



(a) Execution block with forward and full backward.



(b) Execution block with forward and activation backward.

Figure 3: Two types of execution blocks that braid the forward (F) and backward (B) computation units to overlap the TP communications (All-Reduce), denoted as AR.

3 Braided Execution Block

Although TP reduces memory consumption, it introduces additional communication overhead after the Attention and MLP computations in both the forward and backward passes, as illustrated in the orange blocks in Figure 2. As shown in the blue blocks of Figure 3a, the communication represented by the blocks in the communication stream can be naturally overlapped with the weight gradient computation during the backward pass, where TP communication (All-Reduce) and weight gradient computation run in parallel. However, the communication in the solid blocks during the forward pass cannot be easily overlapped with other operations at the software level due to data dependencies, leading to massive TP bubbles. To address this TP bubble issue, we design two types of execution blocks that overlap TP communication with PP computation through braiding forward and backward computation units, as shown in Figure 3. These blocks serve as the foundation of our schedule.

Specifically, we decompose the entire Transformer layer into several fine-grained computation units: the Pre-Attn unit, Attn unit, Pre-MLP unit, and MLP unit. The units from the forward and backward passes are then interleaved sequentially to form a braided execution block, as shown in Figure 3a. Moreover, both the Attn and MLP units in backward passes are further split into two components: activation gradient computation and weight gradient computation, following Zero Bubble [30]. When necessary, we activate the separation of weight gradient computations to balance the workload and reduce pipeline bubbles. This separation does not disrupt the execution blocks, as the subsequent forward computation can fill the bubbles introduced by the separation, as shown in Figure 3b. The Pre-Attn and Pre-MLP units are inserted into the computation stream according to their computational dependencies. All these operations only involve model-level modifications, making our approach more user-friendly compared to kernel-level implementations.

Furthermore, considering the residual computations after the Attn and MLP units that introduce additional data dependencies and increase engineering complexity for backward passes, we modify the forward and backward computations to fuse the residual computation into the Attn and MLP units before applying operation g. This modification preserves computational equivalence and does not affect model convergence. The modified forward computation of the Attn unit for a single TP rank can be formulated as follows:

$$X_{ln} = LayerNorm(X), \quad X_{attn} = AR(Attention(X_{ln}) + \frac{detach(X)}{t}), \tag{1}$$

where t denotes the tensor parallel size, $detach(\cdot)$ is an operation that stops gradient computation of X, $Attention(\cdot)$ is the attention computation performed without any communication, and $AR(\cdot)$ represents the All-Reduce communication operation. The gradient computation $g(X)$ of the hidden state X is then formulated as the following:

$$g(X) = AR\left(\frac{\partial X_{attn}}{\partial X_{ln}}\right) \cdot \frac{\partial X_{ln}}{\partial X} + 1, \tag{2}$$

where the term $+1$ accounts for the gradient contribution from the residual connection. Similarly, the MLP unit adopts a comparable computation fusion strategy.

4 Pipeline Parallelism Schedule

We begin by introducing several key principles that facilitate the construction of the overall schedule in Section 4.1 and then introduce the proposed schedule in Section 4.2. Following this, we present a theoretical analysis of the PP and TP bubbles and peak memory in Section 4.3. Finally, we propose an enhanced variant of our schedule to accommodate scenarios with limited memory in Section 4.4.

4.1 Building Principles

To ensure an effective pipeline schedule, we outline several guiding principles aimed at optimizing memory efficiency and minimizing bubbles.

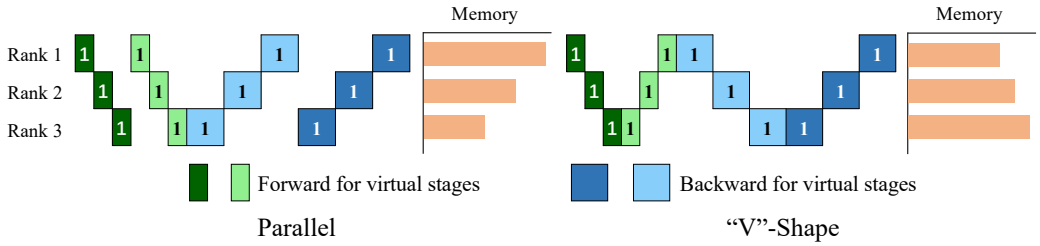


Figure 4: Comparison between parallel and "V"-shape pipeline flow for microbatch 1, which shows improved memory balance across stages. That is attributed to the early backward pass on device 0.

Balanced Memory Footprint: As discussed in [29, 18], the 1F1B-I [23] often leads to a memory bottleneck on the first device due to the parallel dataflow of virtual stages, while subsequent stages often have underutilized memory resources. This imbalance results in inefficient resource usage. To

tackle this issue, current scheduling methods [18, 29, 17] utilize a “V”-shape scheduling strategy to achieve a more balanced distribution of peak memory across all devices (see Figure 4). We follow this principle by implementing a "V"-shape dataflow for our virtual stages to optimize memory efficiency, which also supports the scheduling of braided execution blocks for improved overall performance.

Bubble Minimization: The braided computation-communication execution blocks, as detailed in Section 3, illustrate the arrangement of computation units. The practical construction of these blocks follows two primary patterns: (1) pairing the forward units of one virtual stage with the backward units of another, or (2) braiding the forward and backward units of the same model chunk across different microbatches. The selection of the building pattern is closely tied to the characteristics of virtual stages in PP. Specifically, when partitioning models into virtual stages, it is essential to maintain approximately equal execution times across these stages to mitigate pipeline bubbles. For MLLMs, a significant disparity often exists between the vision encoder (*e.g.*, ViT) and the Language Model (LM) due to differences in their hidden sizes. This presents a key challenge for multimodal models, where earlier virtual stages typically contain more ViT layers, while later stages consist of fewer LM layers to achieve balanced execution times. However, the imbalance in the number of computation units across virtual stages limits the effectiveness of pattern (1), as mismatched unit counts prevent efficient hiding of TP communication during cross-virtual-stage pairings. In contrast, pattern (2), which involves overlapping the forward and backward units of the same chunk, more effectively minimizes the TP bubbles. Therefore, we adopt pattern (2) to construct the basic blocks for universality, especially considering the multimodal models.

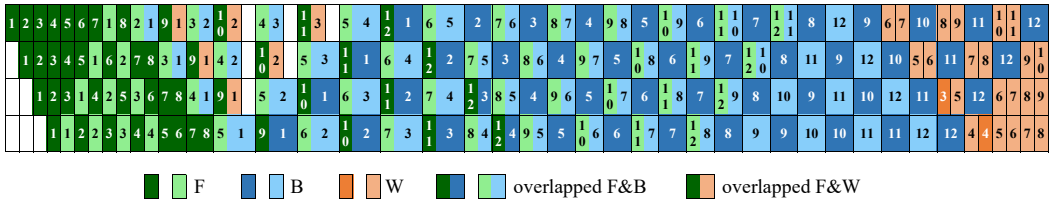


Figure 5: Synergistic tensor and pipeline parallel schedule with the setting of 4 devices and 12 microbatches. The dark and light pieces indicate the computation of model chunks 0 and 1, respectively. F, B, and W represent the forward, backward, and backward for weight gradients, respectively.

4.2 Synergistic Tensor and Pipeline Parallel Schedule

Based on the above building principles, we construct a synergistic tensor and pipeline schedule, which implements a “V”-shape dataflow across stages for a single microbatch, as illustrated in Figure 5. Similar to existing schedules [22, 23, 30], our proposed method comprises three main phases: the warm-up phase, the steady phase, and the cool-down phase. First, the warm-up phase begins with the maximum feasible number of in-flight microbatches before the commencement of the first backward pass. This helps to maintain a low PP rate and facilitates the construction of the braided blocks, as the backward passes of earlier microbatches form the blocks in conjunction with the forward passes of subsequent microbatches. The number of microbatches involved in the warm-up phase is determined by the number of stages. During this phase, the overlapped forward and backward (F&B) start from the braided computation of the first and second microbatches, and weight gradient separation is activated to quickly propagate gradients to the next stage, except for the last stage.

After several iterations of overlapped forward and weight (F&W) and F&B, the schedule transitions into the steady phase. In this phase, weight separation is deactivated as long as new microbatches continue to arrive. Under these conditions, the system performs one F&B for model chunk 1, followed by one F&B for chunk 0. When the supply of microbatches is exhausted, maintaining full F&B becomes infeasible, and the process degrades into a full backward pass followed by a separated F&B. During this degraded phase, weight separation is reactivated to align the time of F&B with that of the full backward pass. During the cool-down phase, the backward passes for all remaining in-flight microbatches are completed. The pipeline bubbles are filled with stored weight gradient computations from earlier steps. Moreover, activating weight separation during the steady phase leads to memory accumulation, as the corresponding weight computations remain pending until the final steps. Therefore, the saved activations required for the weight gradients are offloaded to the CPU in parallel with the computation streams and reloaded when necessary.

4.3 Theoretical Analysis

We define the number of PP stages and microbatches as p and m respectively, where $p \ll m$. For simplicity, multimodal models are not taken into account in this context. For LLMs, we assume the model is evenly divided into chunks along with the layer dimension, and each microbatch requires the same activation memory M_a for each model chunk across all devices. The computation time for the forward pass of one chunk is denoted as T_F, while the durations of activation and weight gradient computation are T_B and T_W, respectively. The TP communication time of one model chunk is represented as T_{AR}, which remains consistent for both forward and backward passes.

Table 1: Comparison between 1F1B-I, Zero Bubble V, and our proposed schedule on theoretical PP bubble size, non-overlapped TP communication bubble, and peak activation memory.

Schedule	PP Bubble	TP Bubble	Peak Act. Memory
1F1B-I	$(p-1)*(T_F+T_{AR}+T_B+T_W)$	$2*m*T_{AR}$	$(3*p-2)*M_a$
ZB-V	$(p-1)*(T_F+2T_{AR}+T_B-2T_W)$	$4*m*T_{AR}$	$2*p*M_a$
Ours	$(p-1)*(T_F+T_{AR}+T_B-T_W)$	$(2*p+1)*T_{AR}$	$3*p*M_a$

As shown in Table 1, 1F1B-I [23] with 2 virtual stages requires $(3*p-2)*M_a$ peak activation memory for the first device, which progressively decreases as the stage index increases. In the backward pass, the TP communications are overlapped naturally, but the TP communications in the forward pass are non-overlapped, resulting in a total TP communication time is $2*m*T_{AR}$. In contrast, Zero Bubble V (ZB-V) [29] achieves a smaller theoretical bubble size of $(p-1)*(T_F+2T_{AR}+T_B-2T_W)$ and maintains the lower peak activation memory than that of 1F1B-I. However, due to the decoupling of the backward into activation and weight gradient computations, the TP communication in both forward and activation backward pass becomes non-overlapped, leading to a total TP communication time of $4*m*T_{AR}$ and a larger practical bubble size. Our schedule exhibits the approximate bubble size of $(p-1)*(T_F+T_{AR}+T_B-T_W)$ compared to the ZB-V schedule, which is much smaller than that of 1F1B-I. And it achieves substantially reduced TP overheads, benefiting from the overlapped computation and communication, albeit with a trade-off in the peak activation memory.

4.4 Enhanced Pipeline Schedule

Considering the practical demands of limited CUDA memory, we attempt to reduce the peak memory footprint of our scheduling strategy. The elevated peak memory usage primarily stems from two phases: the warm-up phase, where the number of in-flight microbatches is maximized to maintain a low bubble rate, and the steady phase, where activations from model chunk 0 persist for an extended period before being released, denoted as lifespan. Existing methods [40, 34] such as activation checkpoint and offloading have been used to reduce memory. However, checkpointing introduces an extra forward pass during backward, increasing execution time. Thus, activation offloading offers a better trade-off between memory reduction and e2e performance, when high-speed bandwidth is available between the host and the device.

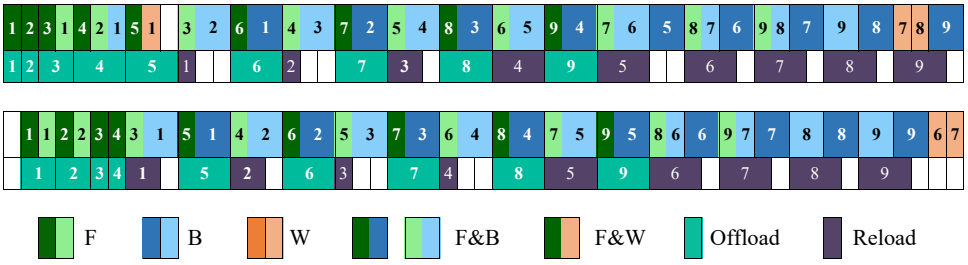


Figure 6: Illustration of our proposed schedule with offloading and reloading for one microbatch in warm-up and steady phase with the setting of PP size 2.

During the warm-up phase in Figure 6, where memory usage accumulates rapidly, the activations of chunk 1 have a short lifespan and are thus deprioritized for offload. The offload time T_o, determined

by the offloading ratio α, is restricted to be less than the forward time T_F to avoid interference with subsequent operations. In the steady phase, chunk 0 activations persist much longer than those in chunk 1, making them the primary target for offloading. The offload ratio α in this phase can be set higher than in the warm-up phase due to the braided execution blocks. To minimize PCIe bandwidth contention caused by frequent dual CPU-GPU data transfers, we avoid offloading chunk 1 activations, ensuring they can be reloaded in time without disrupting computation. The ratio α is adjusted based on hardware characteristics such as PCIe bandwidth and FLOPs.

5 Experiments

5.1 Setup

We evaluated our proposed schedule on the series of Qwen2 (LLM) [39] and Qwen2-VL (MLLM) [36] models, as detailed in Table 2. Meanwhile, Flash Attention 2 [3] is leveraged in all models for efficiency. Our implementation is built upon the open-source Megatron-Core project [33] and tested on up to 32 NVIDIA A800 SXM4 80G GPUs distributed across 4 nodes. Each experimental result is recorded after several warm-up iterations to ensure stability.

In LLM scenarios, we adopt the approach described in [30], uniformly splitting the model while ensuring that the last stage contains two fewer layers than the other stages, considering the large vocabulary size of 152,064 in Qwen. In MLLM scenarios, the ViT encoder is assigned to the first virtual stage on device 0, and the LM model is uniformly distributed across the remaining virtual stages, and the last virtual stage also contains two fewer layers compared to the other stages.

To be specific, our experiments primarily concentrate on the following pipeline parallel schedules: a) interleaved 1F1B (1F1B-I), as implemented in Megatron-LM [23], which reduces the bubble ratio by interleaving stages among workers but slightly increases memory footprint; b) ZB-V, introduced in [29], which achieves a balanced peak memory distribution through a "V"-shape dataflow; and c) our proposed schedule. For consistency, all schedules are configured with two virtual stages per device, and the implementations are based on open-source codebases.

Table 2: Model configurations evaluated on the experiments.

Model	Scale	ViT Layers	ViT Heads	ViT Dim.	LM Layers	LM Q Heads	LM KV Heads	LM Dim.
LLM	12.1B	-	-	-	30	40	8	5120
LLM	26.3B	-	-	-	46	56	8	7168
MLLM	14.9B	32	16	2048	33	40	8	5120
MLLM	28.8B+	26	16	4096	40/43	56	8	7168

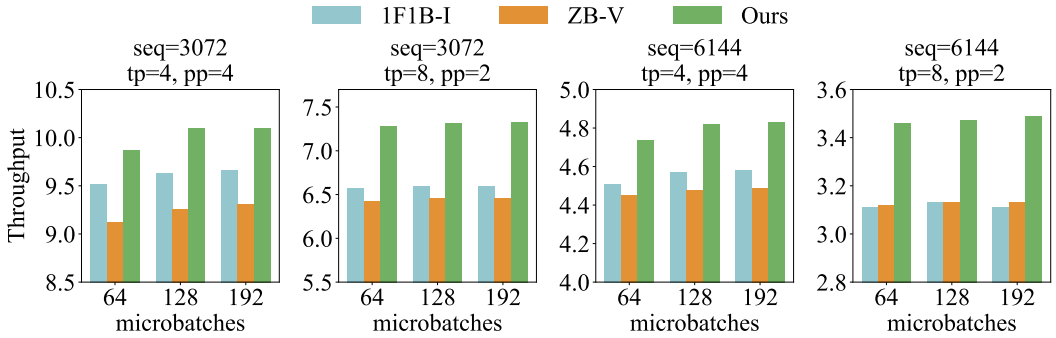


Figure 7: Experiment results on 12.1B LLM across 16 GPUs in terms of throughput (samples per second).

5.2 Comparison with Existing Schedules on LLM

In Figures 7 and 8, we report experimental results by evaluating Qwen2 on 2-node and 4-node setups, respectively, showing the achieved throughput under various pipeline strategies, sequence lengths,

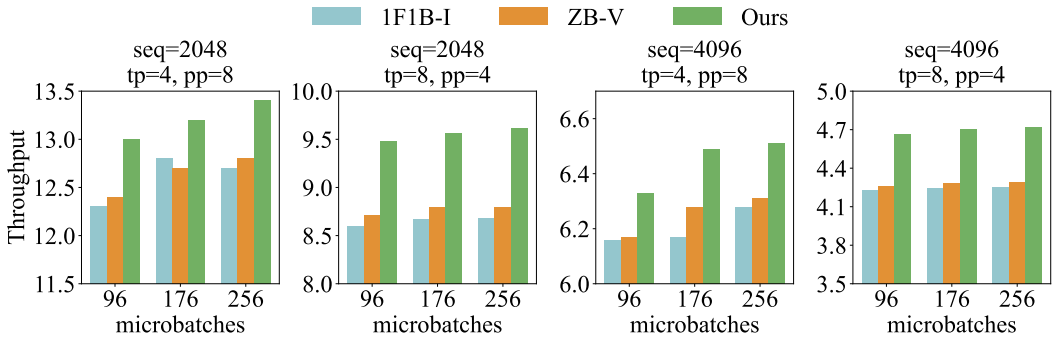


Figure 8: Experiment results on 26.3B LLM across 32 GPUs in terms of throughput.

and model scale configurations. Our proposed schedule outperforms all other strategies across all settings in terms of throughput. As shown in the right part of Figure 7, our method achieves a maximum improvement of 12.2% over 1F1B-I. This performance gain is attributed to the large TP size of 8, the long sequence length of 6,144, and the low PP size of 2 used in the experiments. As illustrated in Figure 1, TP-related bubbles occupy a significant proportion under TP size 8, especially for attention units, which are effectively mitigated by our scheduling approach.

However, the performance improvements under PP=8 in Figure 8 are less pronounced. This is due to the explicit pipeline communication in our schedule, which is executed immediately after computation and cannot be overlapped with computation, thereby negatively impacting performance. Nevertheless, our scheduling strategy still outperforms the other two methods. The highest throughput improvements for both 12B and 26B LLMs are achieved at TP=8, demonstrating that our method offers substantial potential when scaling to larger models with TP=8, even TP=16. Notably, ZB-V demonstrates comparable or even worse performance than 1F1B-I. Since ZB-V is based on the decoupling of the full backward pass, which exposes previously overlapped TP bubbles during the full backward in 1F1B-I and introduces additional bubble overheads, thereby eliminating the potential benefit of reduced PP bubbles from decoupling.

Furthermore, we show the peak activation memory footprints with 4 and 2 PP stages in Figure 9. While our schedule exhibits a slightly higher peak memory footprint, it can achieve optimal throughput, especially under memory-unconstrained scenarios. For memory-limited scenarios, we introduce an enhanced variant that significantly reduces memory footprints while maintaining comparable throughput, detailed in Section 5.4.

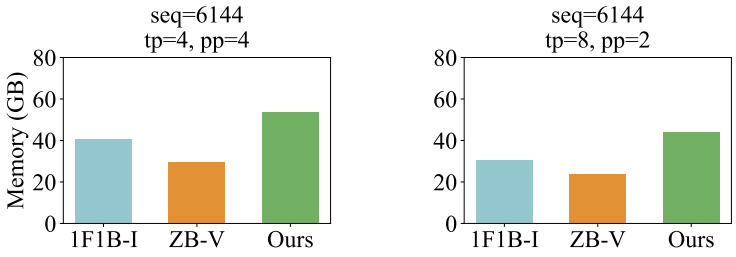


Figure 9: Peak activation memory footprint tested on 12.1B LLM across 2 nodes.

5.3 Comparison with Existing Schedules on MLLM

As shown in Tables 3, we present experimental evaluations of Qwen2-VL on 2-node and 4-node setups, respectively, which report throughput and peak activation memory under various configurations, including ViT lengths, LM lengths, and model scale parameters. For the PP=4, we carefully adjust the ViT and LM sequence lengths to balance computational workload across virtual stages by ensuring the FLOPs of the ViT approximately match those of individual virtual stages. In contrast, for PP=2 and PP=8, we intentionally maintain disparities in ViT/LM FLOPs to conduct diverse workload scenarios. Our scheduling approach demonstrates superior performance compared to baseline methods in all balanced computational scenarios. Notably, we observe a maximum throughput improvement

Table 3: Experimental results for the 14.9B, 28.8B, and 30.3B MLLMs, showing throughput (samples per second) and peak activation memory footprint (GB). mbs denotes the number of microbatches.

Model	ViT Length	LM Length	TP	PP	Schedule	mbs 64	mbs 128	mbs 192	Memory
1.7B ViT	3136	5120	4	4	1F1B-I	4.36	4.44	4.46	40
					ZB-V	4.25	4.30	4.31	**30**
13.2B LM					Ours	**4.53**	**4.63**	**4.65**	56
			8	2	1F1B-I	2.42	2.44	2.46	**26**
16 GPUs					ZB-V	2.48	2.49	2.49	30
					Ours	**2.86**	**2.86**	**2.87**	51

Model	ViT Length	LM Length	TP	PP	Schedule	mbs 96	mbs 176	mbs 256	Memory
5.6B ViT	1600	3072	4	8	1F1B-I	5.79	5.89	5.85	60
					ZB-V	5.90	5.97	6.01	**48**
23.2B / 24.7B LM					Ours	**5.97**	**6.19**	**6.19**	69
	2304	5120	8	4	1F1B-I	3.37	3.40	3.40	43
32 GPUs					ZB-V	3.51	3.53	3.53	**38**
					Ours	**3.72**	**3.80**	**3.80**	47

of 11.7% in the configuration with TP=8 and PP=4, which is significantly higher than the 2% improvement in TP=4 and PP=4, compared to 1F1B-I. This trend aligns with observations from LM experiments, indicating that larger TP sizes generally yield greater performance gains.

A particularly notable result is the 16.7% performance improvement achieved by our schedule in the PP=2 setting, where the ViT component has relatively lower computational intensity compared to other model chunks. In this case, the lowest peak memory footprint is achieved by 1F1B-I in stage 1, where only a few microbatches are in-flight and the memory in ViT is much smaller compared to that in LM. In the PP=8 scenario, where the ViT's FLOPs exceed those of the LM chunks, our schedule and ZB-V achieve comparable throughputs at $mbs = 96$. However, as the number of microbatches increases, our approach demonstrates superior scalability and robustness, gaining more improvements compared to ZB-V and 1F1B-I.

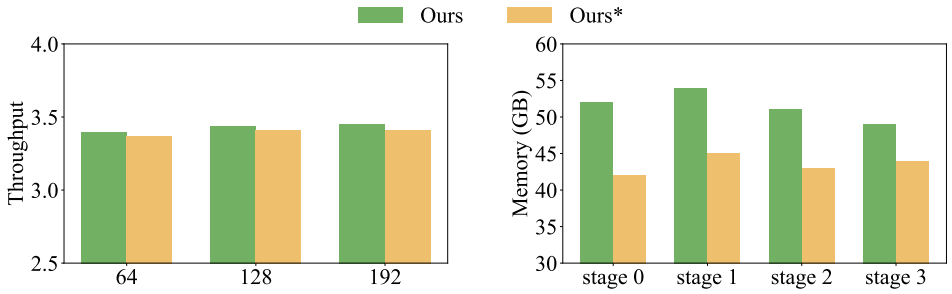


Figure 10: Experimental results in terms of throughput and peak activation memory footprints over 4 PP stages on the 12.1B LLM. $*$ indicates the enhanced variant with offloading.

5.4 Efficiency of Our Enhanced Schedule

Due to the limited PCIe bandwidth of the A800, where the effectiveness of the offloading variant cannot be fully demonstrated, we perform experiments on 16 NVIDIA H20 GPUs, which are equipped with PCIe Gen 5 interconnection. As shown in Figure 10, the offloading variant incurs a negligible throughput degradation due to additional CPU overheads associated with data transmission. However, as demonstrated in the right part of Figure 10, the memory footprints across all stages are approximately equal, indicating our schedule effectively utilizes the memory resources to improve

the throughput. And the offloading strategy achieves a peak memory reduction ranging from 10% to 19.2%, with the resulting peak memory footprint being close to that of 1F1B-I (40G).

5.5 Results under Maximized Resource Utilization

We conduct experiments on 16 NVIDIA H20 96GB GPUs, adjusting the microbatch size across various parallel configurations to maximize memory utilization and achieve optimal throughput and Model FLOPs Utilization (MFU). As shown in Table 4, our proposed method delivers superior performance under maximized memory utilization. Specifically, under the TP=2, PP=8 configuration, our enhanced schedule achieves slightly higher throughput (2.74 samples/s) and MFU (92.86%) compared to 1F1B-I, despite the lower proportion of TP bubbles on H20 GPUs (see the Appendix D for details), while maintaining a lower and more balanced memory footprint of 68 GB.

Table 4: Experiment results on 12.1B LLM under maximized memory utilization, which are evaluated on 16 H20 GPUs. $*$ denotes the enhanced variant of our schedule with activation offloading.

Configurations	Parallelism	mbs	Schedules	Throughput	MFU	Memory
mbs=192 seq_len=8192	TP=2 pp=8	1	1F1B-I	2.72	92.09	76
			ZB-V	2.61	88.36	54
			Ours	OOM		
			Ours*	**2.74**	**92.86**	68
	TP=4 pp=4	1	1F1B-I	2.47	83.62	53
			ZB-V	2.41	81.59	38
			Ours	2.52	85.32	71
		2	1F1B-I	OOM		
			ZB-V	2.47	83.62	75
			Ours*	OOM		
	TP=8 pp=2	1	1F1B-I	2.06	69.74	40
			ZB-V	2.07	70.08	31
			Ours	2.12	71.78	57
		2	1F1B-I	2.14	72.45	76
			ZB-V	2.18	73.81	61
			Ours*	OOM		
		3	1F1B-I	OOM		
			ZB-V	1.74	58.91	90
			Ours*	OOM		

6 Conclusion

In this paper, we presented a novel hybrid parallel strategy that synergistically integrates tensor and pipeline parallelism for distributed training of LLMs and MLLMs. By decomposing Transformer layers into fine-grained computation units and interleaving forward and backward computations within each PP stage, we constructed execution blocks that effectively overlap TP communication bubbles. Based on these blocks, we designed a "V"-shape PP schedule, which achieves balanced memory across stages and significantly reduces both TP and PP bubbles, albeit at the cost of increased peak memory. Experimental results demonstrated that our approach consistently outperforms existing scheduling methods on both LLMs and MLLMs. We also presented an enhanced variant incorporating offloading to better accommodate memory-constrained scenarios. In future work, we plan to explore more memory- and throughput-efficient hybrid schedules in the large-scale machine learning system.

Acknowledgement

This work was partly supported by the Funds for the National Natural Science Foundation of China under Grant 62202063, 62572072, U24B20176, and Beijing Natural Science Foundation (L243027).

References

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

[2] Li-Wen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Chengji Yao, Ziheng Jiang, et al. Flux: fast software-based communication overlap on gpus through kernel fusion. *arXiv preprint arXiv:2406.06858*, 2024.

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

[4] Wei Deng, Mengshi Qi, and Huadong Ma. Global-local tree search in vlms for 3d indoor scene generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 8975–8984, 2025.

[5] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[6] Jun Huang, Zhen Zhang, Shuai Zheng, Feng Qin, and Yida Wang. {DISTMM}: Accelerating distributed multimodal model training. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1157–1171, 2024.

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

[8] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 402–416, 2022.

[9] Byungsoo Jeon, Mengdi Wu, Shiyi Cao, Sunghyun Kim, Sunghyun Park, Neeraj Aggarwal, Colin Unger, Daiyaan Arfeen, Peiyuan Liao, Xupeng Miao, et al. Graphpipe: Improving performance and scalability of dnn training with graph pipeline parallelism. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 557–571, 2025.

[10] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

[11] Chenyu Jiang, Zhen Jia, Shuai Zheng, Yida Wang, and Chuan Wu. Dynapipe: Optimizing multi-task training through dynamic pipelines. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 542–559, 2024.

[12] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed {DNN} training in heterogeneous {GPU/CPU} clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479, 2020.

[13] Taebum Kim, Hyoungjoo Kim, Gyeong-In Yu, and Byung-Gon Chun. Bpipe: Memory-balanced pipeline parallelism for training large language models. In *International Conference on Machine Learning*, pages 16639–16653. PMLR, 2023.

[14] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353, 2023.

[15] Mu Li, David G Andersen, Alexander Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. *Advances in neural information processing systems*, 27, 2014.

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

[17] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[18] Ziming Liu, Shenggan Cheng, Haotian Zhou, and Yang You. Hanayo: Harnessing wave-like pipeline parallelism for enhanced large model training efficiency. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2023.

[19] Changsheng Lv, Shuai Zhang, Yapeng Tian, Mengshi Qi, and Huadong Ma. Disentangled counterfactual learning for physical audiovisual commonsense reasoning. *Advances in Neural Information Processing Systems*, 36:12476–12488, 2023.

[20] Changsheng Lv, Mengshi Qi, Xia Li, Zhengyuan Yang, and Huadong Ma. Sgformer: Semantic graph transformer for point cloud-based 3d scene graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 4035–4043, 2024.

[21] Xupeng Miao, Yining Shi, Zhi Yang, Bin Cui, and Zhihao Jia. Sdpipe: A semi-decentralized framework for heterogeneity-aware pipeline-parallel training. *Proceedings of the VLDB Endowment*, 16(9):2354–2363, 2023.

[22] 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 *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

[23] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–15, 2021.

[24] NVIDIA Corporation. NVLink & NVSwitch: Fastest HPC Data Center Platform. `https://www.nvidia.com/en-us/data-center/nvlink/`.

[25] Suchita Pati, Shaizeen Aga, Mahzabeen Islam, Nuwan Jayasena, and Matthew D Sinclair. T3: Transparent tracking & triggering for fine-grained overlap of compute & collectives. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 1146–1164, 2024.

[26] Mengshi Qi, Weijian Li, Zhengyuan Yang, Yunhong Wang, and Jiebo Luo. Attentive relational networks for mapping images to scene graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3957–3966, 2019.

[27] Mengshi Qi, Jie Qin, Yi Yang, Yunhong Wang, and Jiebo Luo. Semantics-aware spatial-temporal binaries for cross-modal video retrieval. *IEEE Transactions on Image Processing*, 30:2989–3004, 2021.

[28] Mengshi Qi, Hao Ye, Jiaxuan Peng, and Huadong Ma. Action quality assessment via hierarchical pose-guided multi-stage contrastive regression. *arXiv preprint arXiv:2501.03674*, 2025.

[29] Penghui Qi, Xinyi Wan, Nyamdavaa Amar, and Min Lin. Pipeline parallelism with controllable memory. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[30] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble (almost) pipeline parallelism. In *The Twelfth International Conference on Learning Representations*, 2024.

[31] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *ICML 2022*, January 2022.

[32] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018.

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

[34] Zhenbo Sun, Huanqi Cao, Yuanwei Wang, Guanyu Feng, Shengqi Chen, Haojie Wang, and Wenguang Chen. Adapipe: Optimizing pipeline parallelism with adaptive recomputation and partitioning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 86–100, 2024.

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

[36] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

[37] Shibo Wang, Jinliang Wei, Amit Sabne, Andy Davis, Berkin Ilbeyi, Blake Hechtman, Dehao Chen, Karthik Srinivasa Murthy, Marcello Maggioni, Qiao Zhang, et al. Overlap communication with dependent computation via decomposition in large deep learning models. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 93–106, 2022.

[38] Weihu Wang, Yaqi Xia, Donglin Yang, Xiaobo Zhou, and Dazhao Cheng. Accelerating distributed dlrm training with optimized tt decomposition and micro-batching. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2024.

[39] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[40] Tailing Yuan, Yuliang Liu, Xucheng Ye, Shenglong Zhang, Jianchao Tan, Bin Chen, Chengru Song, and Di Zhang. Accelerating the training of large language models using efficient activation rematerialization and optimal hybrid parallelism. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 545–561, 2024.

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: We claim the contribution of our work clearly in the abstract and introduction.

   Guidelines:

   - The answer NA 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 NA 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: The main limitation of our work is discussed in the last section.

   Guidelines:

   - The answer NA 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: [NA]

Justification: There is no theoretical result in our work.

Guidelines:

- The answer NA 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: Our implementation is based on open sourced M-Core, which is accessible from everyone and we describe the setting of experiments clearly.

   Guidelines:

   - The answer NA 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: : We would open the source code after the paper is accepted.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.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://nips.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, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Detailed in Section Experiments.

   Guidelines:

   - The answer NA 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: The results are stable in our clusters.

   Guidelines:

   - The answer NA 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 list all resources information in Section Experiments.

   Guidelines:

   - The answer NA 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: The research is conducted with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA 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: [NA]

    Justification: Our research focus on the training of LLMs and has no impact on society.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA 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: Our work is developed based on Megatron-Core with a license that everyone can work on it.

    Guidelines:

    - The answer NA 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: We will open source the related documents and codes on the camera ready phase.

Guidelines:

- The answer NA 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: [NA]

Justification: Not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA 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: [NA]

Justification: Not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA 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 rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The method in this research does not involve LLMs as any important, original, or non-standard components, just uses it for training tests.

Guidelines:

- The answer NA 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 (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Appendix

A The Warm-Up Phase of the Pipeline Schedule

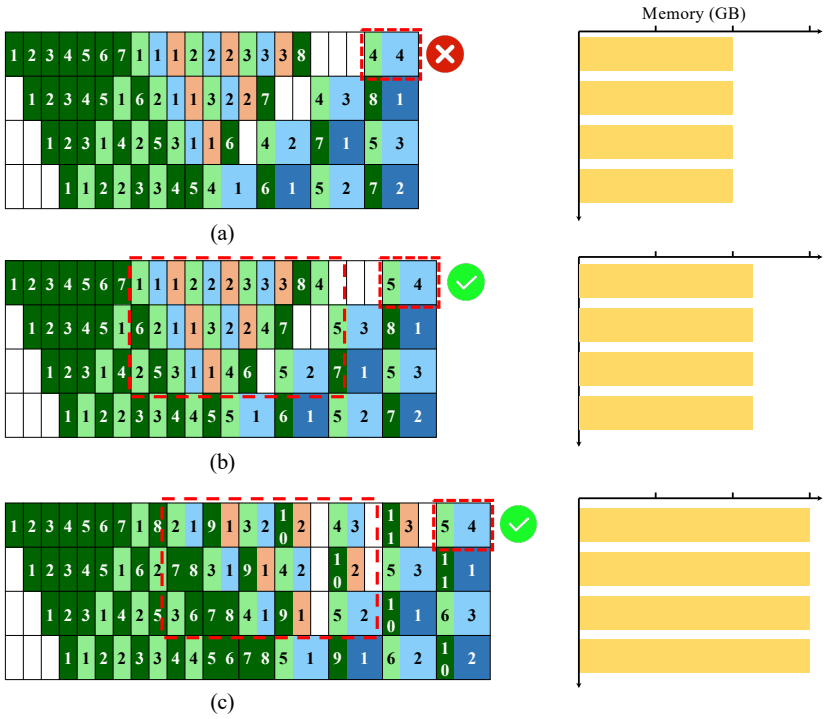


Figure 11: The construction of the warm-up phase: (a) wrong warm-up phase; (b) memory-efficient warm-up phase; (c) throughput-efficient warm-up phase.

The guiding principle for constructing the warm-up phase is to ensure the correct execution of overlapped forward and backward passes. As illustrated at the top of Figure 11, a key requirement is that overlapping between forward and backward passes must occur across different microbatches. Specifically, the microbatch index in the forward pass should be greater than that in the backward pass. To satisfy this condition, an additional forward pass is required before the overlapped F&B execution begins, as shown in Figure 11(b). However, this approach necessitates decoupling the backward pass into activation and weight gradient computations, which exposes TP communication and introduces additional PP communication overheads. Please refer to Section B for more details about the memory-efficient warm-up phase.

B Comparison with Other Schedules

We compare 1F1B-I [23], ZB-V [29], and our proposed schedule under identical configurations: 4 pipeline stages, 2 virtual stages per pipeline stage, and 12 microbatches, as illustrated in Figure 12. Intuitively, both ZB-V and our schedule have a similar number of PP bubbles, which are significantly fewer than those in 1F1B-I. The PP bubbles in 1F1B-I primarily result from the warm-up and cool-down phases. In the warm-up phase, the parallel dataflow across virtual stages results in a large number of bubbles. For instance, the long lifespan of virtual stage 2 for microbatch 1 cannot be efficiently filled without introducing additional forward passes from more microbatches, which would increases the peak memory of the stage 1. Consequently, 1F1B-I incurs greater PP bubble overheads during the warm-up phase. Meanwhile, the bubbles in the cool-down phase are inevitable due to inherent data dependencies.

Compared to ZB-V, our schedule results in a small number of PP bubbles at the end of the warm-up phase, which are attributed to the relatively long execution time and specific PP dataflow pattern

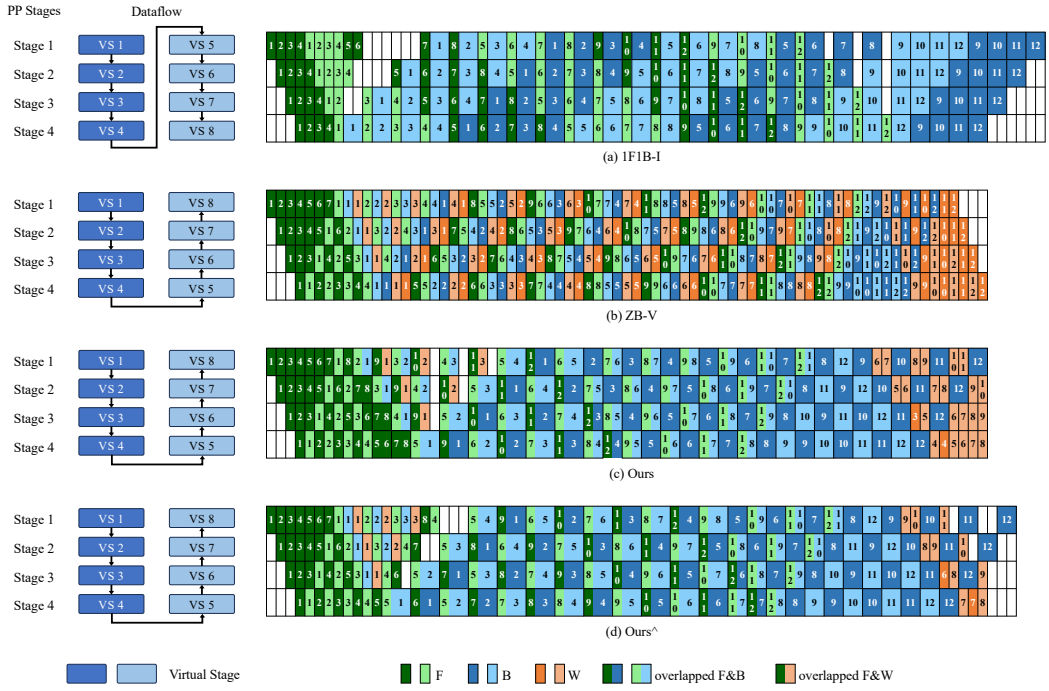


Figure 12: Synchronous pipeline parallelism schedules compared in our work, with 4 pipeline stages and 12 microbatches within a training iteration. Dark colors denote the first virtual stage and light colors are the second virtual stage. ˆdenotes the tensor and pipeline schedule beginning with the memory-efficient warm-up phase.

of the overlapped F&B blocks. In general, the activation gradient computation time T_B satisfies $T_B > T_W$, where T_W is the weight gradient computation time. As a result, small bubbles also appear at the end of the cool-down phase, and their magnitudes are approximately equal for both ZB-V and our proposed schedule.

Moreover, we also present schedule Oursˆ (d), which begins with a memory-efficient warm-up phase, as illustrated in Figure 12. Although schedule (d) has a lower peak memory footprint compared to our standard schedule (c), it introduces additional PP bubbles towards the end, negatively impacting throughput performance. Therefore, for scenarios involving large-scale model training with large TP sizes and ample memory availability, we prefer schedule (c) for its higher performance. Our enhanced schedule variant, which includes activation offloading, is preferable for reducing peak memory usage, assuming sufficient bandwidth. Schedule (d) is suitable only for scenarios with extremely limited memory and large TP overheads.

C Detailed Experimental Results on LLMs

We present detailed experimental results based on the 12.1B and 26.3B Qwen2 models, as summarized in Tables 5, 6, and 7. Notably, the GPU memory footprint reported in Table 5 is slightly higher than the theoretical estimate. We attribute this discrepancy primarily to practical aspects of our engineering implementation, such as the implementation of the fine-grained computation units and the decoupling of weight gradient computation. For example, under TP=8, profiling the activation memory of one microbatch per virtual stage shows that ZB-V consumes 3.6 GB, while our approach uses 4.3 GB, an increase of nearly 20%. Although this issue has no impact on throughput, we plan to further refine the engineering implementation to reduce this overhead in future work.

Table 5: Experimental results on 12.1B and 26.3B LLMs in terms of peak memory (GB), where the microbatch sizes of samples with sequence length 3072 and 2048 are set to 2.

Model	Sequence Length	TP	PP	1F1B-I	ZB-V	Ours
12.1B LLM 16 GPUs	3072	4	4	41	30	53
		8	2	31	24	43
	6144	4	4	41	30	54
		8	2	31	24	43
26.3B LLM 32 GPUs	2048	4	8	55	38	72
		8	4	43	32	59
	4096	4	8	55	38	72
		8	4	43	32	59

Table 6: Experimental results on 12.1B and 26.3B LLMs in terms of throughput (samples per second). mbs represents the number of microbatches.

Model	Sequence Length	TP	PP	mbs	1F1B-I	ZB-V	Ours
12.1B LLM 16 GPUs	3072	4	4	64	9.52	9.12	9.87
				128	9.63	9.26	**10.1**
				192	9.66	9.31	**10.1**
		8	2	64	6.57	6.42	7.28
				128	6.60	6.46	7.32
				192	6.60	6.46	**7.33**
	6144	4	4	64	4.51	4.45	4.74
				128	4.57	4.48	4.82
				192	4.58	4.49	**4.83**
		8	2	64	3.11	3.13	3.46
				128	3.13	3.13	3.47
				192	3.11	3.13	**3.49**
26.3B LLM 32 GPUs	2048	4	8	96	12.3	12.4	13.0
				176	12.8	12.7	13.2
				256	12.7	12.8	**13.4**
		8	4	96	8.60	8.71	9.48
				176	8.67	8.79	9.56
				256	8.68	8.79	**9.61**
	4096	4	8	96	6.16	6.17	6.33
				176	6.17	6.28	6.49
				256	6.28	6.31	**6.51**
		8	4	96	4.23	4.26	4.66
				176	4.24	4.28	4.70
				256	4.25	4.29	**4.72**

D Comparison with Other Schedule Methods on H20 GPUs

We also conduct experiments on 16 NVIDIA H20 96G GPUs to evaluate the performance of our proposed schedule alongside other baselines, as summarized in Table 8. Our proposed schedule outperforms other approaches by a margin, but the performance improvement is far less significant compared to that achieved on A800 GPUs. This discrepancy can be primarily attributed to the characteristics of the H20 GPUs, which have lower BF16 FLOPs but higher communication

Table 7: Experimental results on 12.1B and 26.3B LLMs in terms of Model FLOPs Utilization (MFU, %).

Model	Sequence Length	TP	PP	mbs	1F1B-I	ZB-V	Ours
12.1B LLM 16 GPUs	3072	4	4	64	46.31	44.36	48.01
				128	46.84	45.04	**49.13**
				192	46.99	45.28	**49.13**
		8	2	64	31.96	31.23	35.41
				128	32.10	31.42	35.61
				192	32.10	31.42	**35.65**
	6144	4	4	64	50.16	49.50	52.72
				128	50.83	49.83	53.61
				192	50.94	49.94	**53.72**
		8	2	64	34.59	34.81	38.49
				128	34.81	34.81	38.60
				192	34.59	34.81	**38.82**
26.3B LLM 32 GPUs	2048	4	8	96	42.16	42.50	44.56
				176	43.87	43.53	45.24
				256	43.53	43.87	**45.93**
		8	4	96	29.48	29.85	32.49
				176	29.72	30.13	32.77
				256	29.75	30.13	**32.94**
	4096	4	8	96	46.32	46.40	47.60
				176	46.40	47.23	48.80
				256	47.23	47.45	**48.95**
		8	4	96	31.81	32.04	35.04
				176	31.88	32.19	35.34
				256	31.96	32.26	**35.49**

bandwidth. Furthermore, we profile the computation and communication times for the Attention and MLP modules of a single Transformer layer in the 12.1B Qwen2 model, as shown in Figure 13. It is evident that the proportion of TP communication is significantly lower than that measured on A800 GPUs. As a result, the benefits derived from optimizing TP bubbles have slightly diminished.

Table 8: Experimental results evaluated on 12.1B LLM on 16 H20 GPUs. mbs and seq_len denote the number of microbatches and the sequence length, respectively.

Configurations	TP	PP	Schedules	Throughput	MFU	Memory
mbs=192 seq_len=6144	2	8	1F1B-I	3.77	88.34	58
			ZB-V	3.63	85.06	**42**
			Ours	**3.79**	**88.81**	73
	4	4	1F1B-I	3.39	77.43	41
			ZB-V	3.30	77.32	**30**
			Ours	**3.44**	**80.61**	54
	8	2	1F1B-I	2.80	65.61	31
			ZB-V	2.80	65.61	**24**
			Ours	**2.89**	**67.72**	43

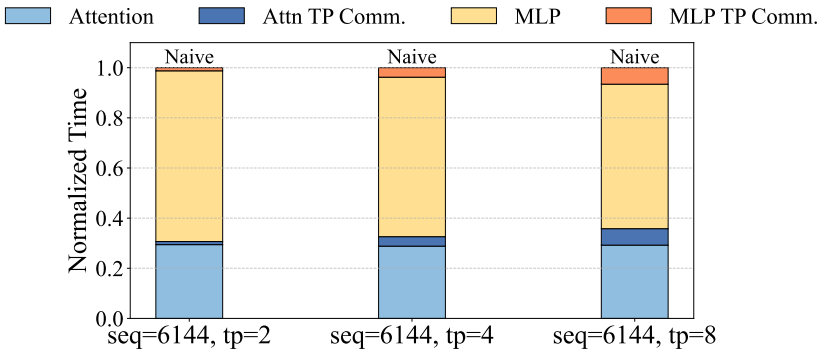


Figure 13: The computation and communication time proportion of Attention and MLP modules profiled on H20 GPUs.

E Evaluations with Existing Training Strategies

E.1 Activation Checkpointing

Our proposed schedule is compatible with activation checkpointing strategies. We conducted the experiment on Qwen2-12.1B using activation checkpointing (AC) with a batch size of 128 and a sequence length of 6k, and report the results in the following table:

Table 9: Performance comparison with different activation checkpointing configurations.

Config	Throughput (samples/s)	Peak Memory (GB)
AC disabled	4.79	56.0
AC enabled in Ours w/ MLP	4.19	44.7
AC enabled in Ours w/ Attn+MLP	3.94	41.5
AC enabled in Ours w/ Attn+MLP+Norm	3.75	36.3

As shown in Table 9, disabling AC yields the highest throughput of 4.79 samples per second but requires a peak memory footprint of 56.0 GB. Enabling AC selectively on the MLP modules reduces peak memory by 20.2% to 44.7 GB, with a corresponding throughput decrease of 12.5% to 4.19 samples/s. Extending AC to both Attention and MLP modules further reduces peak memory to 41.5 GB (a 25.9% reduction), while throughput declines to 3.94 samples/s. The most aggressive configuration, applying AC to all modules, achieves the greatest memory savings, reducing peak memory by 35.2% to 36.3 GB, at the cost of a 21.7% reduction in throughput, resulting in 3.75 samples/s. These results confirm that our scheduling framework is fully compatible with activation checkpointing strategies.

E.2 Data Parallelism and Context Parallelism

To demonstrate the compatibility of our schedule with both data parallelism (DP) and context parallelism (CP), we conduct experiments on a 12.1B Qwen2 model. As shown in Table 10, under identical tensor parallelism (TP=2) and pipeline parallelism (PP=4) configurations, our method achieves the highest throughput in both settings: 2.71 samples/s with CP (CP=2, $mbs = 128$, sequence length 12k) and 9.40 samples/s with DP (DP=2, $mbs = 256$, sequence length 4k). Our approach consistently outperforms baseline schedules (1F1B-I and ZB-V) across both parallelism paradigms, confirming its robustness and adaptability to different distributed training strategies.

F Impact of Communication Overlap on GEMM Execution Efficiency

In distributed training of large language models, overlapping communication (e.g., AllReduce) with computation (e.g., GEMM) is widely used to improve hardware utilization and reduce training time.

Table 10: Experimental results on 12.1B LLM compatible with data parallelism and context parallelism.

TP	PP	CP	mbs	Seq	Schedules	Throughput
2	4	2	128	12k	1F1B-I	2.64
					ZB-V	2.61
					Ours	**2.71**
TP	PP	DP	mbs	Seq	Schedules	Throughput
2	4	2	256	4k	1F1B-I	9.22
					ZB-V	9.16
					Ours	**9.40**

However, this overlap may cause resource contention, particularly for streaming multiprocessors (SMs) which potentially degrade GEMM performance.

To evaluate this effect, we conduct microbenchmarks under two scenarios: (1) GEMM dominates AllReduce, enabling full overlap; and (2) GEMM finishes early, leaving part of the communication exposed. In both cases, we compare overlapped execution against the sequential baseline (GEMM followed by AllReduce) and the standalone GEMM time.

Table 11: Execution times (in milliseconds) for GEMM, AllReduce, and their combinations under sequential and overlapped execution.

Operation	Experiment 1	Experiment 2
GEMM	8.605	0.334
AllReduce	3.364	1.643
GEMM + AllReduce (sequential)	11.969	1.977
GEMM with overlapped AllReduce	9.251	1.685

The results, summarized in Table 11, show that in the first scenario, where GEMM takes 8.605 ms and AllReduce 3.364 ms, the overlapped execution completes in 9.251 ms, only 7.5% slower than the GEMM alone and 22.6% faster than the sequential execution (11.969 ms). This indicates that the communication is effectively hidden with minimal interference.

In the second scenario, where GEMM is very short (0.334 ms) and AllReduce dominates (1.643 ms), the overlapped execution finishes in 1.685 ms, which is just 2.6% overhead over the AllReduce time and still 14.8% faster than the sequential baseline (1.977 ms). Notably, the GEMM itself is not measurably slowed down; rather, the slight increase in total time stems from the unavoidable tail of communication that cannot be hidden.