AMDP: ASYNCHRONOUS MULTI-DIRECTIONAL PIPELINE PARALLELISM FOR LARGE-SCALE MODELS TRAINING

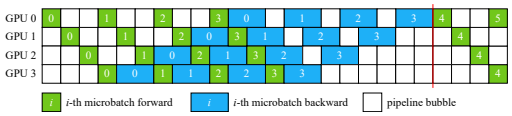
Anonymous authors
Paper under double-blind review

ABSTRACT

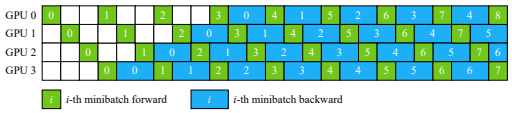
Pipeline parallelism has become a critical technique for scaling up the training of large models. However, existing asynchronous pipeline approaches often suffer from degraded convergence due to parameter mismatch between forward and backward passes. To address this, we propose **A**synchronous **M**ulti-**D**irectional **P**ipeline parallelism (**AMDP**). AMDP limits stage 0 of each pipeline to read only two minibatches before initiating the first backward pass, thereby reducing the number of parameter updates that occur between the forward and backward passes of each minibatch. To mitigate the pipeline bubbles introduced by this restriction, AMDP instantiates multiple concurrent pipelines and adapts their number according to pipeline depth. Furthermore, AMDP accumulates gradients across minibatches and applies them in a single parameter update, ensuring that only a small number of minibatches (bounded by the pipeline depth) encounter parameter mismatch, which is constrained to within one step. Experiments on GPT- and BERT-style models demonstrate that AMDP significantly accelerates the training of large-scale models while preserving convergence. The source code based on Megatron-LM is avaiable at https://anonymous.4open.science/r/Megatron-AMDP-BB23.

1 INTRODUCTION

Since the introduction of Transformers (Vaswani et al., 2017), scaling model size has become a primary driver of progress in deep learning, enabling remarkable advances across diverse applications. Larger models consistently deliver stronger performance, but their training poses formidable challenges. For example, GPT-3 (Brown et al., 2020) contains 175B parameters, requiring about 350 GB of memory in 16-bit precision. Even disregarding memory constraints, training GPT-3 on a single NVIDIA V100 GPU would take an estimated 288 years (Narayanan et al., 2021b). These prohibitive costs necessitate distributed parallel training across multi-GPU clusters.



(a) Synchronous pipeline parallelism



(b) Asynchronous pipeline parallelism

Figure 1: Comparison of synchronous and asynchronous pipeline parallelism, with four model stages assigned to four devices. The backward pass duration is assumed to be twice that of the forward pass.

Pipeline parallelism (Guan et al., 2024) has emerged as a key strategy for distributed parallel training. By partitioning a model into sequential stages, each executed on a different GPU, pipeline parallelism reduces per-device memory requirements and increases throughput. Depending on how parameters are updated, pipeline parallelism can be classified into synchronous and asynchronous schemes. In synchronous pipeline parallelism (Huang et al., 2019), a minibatch is split into microbatches. Gradients from all microbatches are accumulated and applied only after all backward passes complete, and the next iteration begins once all parameters are updated. As shown in Figure 1(a), this scheme is straightforward but leads

to pipeline bubbles—idle periods caused by inter-stage dependencies. In contrast, asynchronous pipeline parallelism (Narayanan et al., 2019) continuously feeds minibatches into the pipeline, updating parameters immediately after each backward pass. As shown in Figure 1(b), this design eliminates bubbles once steady state is reached, but sacrifices convergence because parameter updates may occur between the forward and backward passes of a minibatch, resulting in parameter mismatch. To mitigate this mismatch, two techniques have been introduced: parameter stash and parameter prediction. Parameter stash (Narayanan et al., 2019) preserves the parameters used in the forward pass and reuses them in the corresponding backward pass to enforce consistency. Parameter prediction (Chen et al., 2018; Ajanthan et al., 2025) instead forecasts future parameter values using momentum-based optimizers (Kingma, 2014), thereby reducing mismatch by aligning computations with predicted parameters.

Existing asynchronous approaches predominantly adopt a 1F1B schedule, where each device alternates between one forward and one backward passes. To fully eliminate pipeline bubbles, these approaches continuously read minibatches into the pipeline. However, this design causes the severity of parameter mismatch to increase with pipeline depth, ultimately degrading convergence. Parameter stash mitigates mismatch but introduces delayed gradients, while parameter prediction avoids delay but relies on simplified forecasting that often deviates from true parameter values.

To address the aforementioned issues, we propose **A**synchronous **M**ulti-**D**irectional **P**ipeline parallelism (**AMDP**). AMDP builds upon the bidirectional scheduling concept introduced in Chimera (Li & Hoefler, 2021) for synchronous training, but extends it to a fully asynchronous and multi-directional setting with three components absent in Chimera: 1) a structural mismatch-control mechanism that bounds forward–backward inconsistency to one step; 2) a gradient-accumulation update scheme that confines mismatch effects to minibatches within a window sized by the pipeline depth; and 3) a zero-redundancy optimizer (ZeRO) that efficiently manages the memory overhead of concurrent pipelines. The contributions of AMDP are summarized as follows:

- We propose an asynchronous multi-directional pipeline schedule with a one-step mismatch bound, which limits stage 0 of each pipeline to read only two minibatches before initiating the first backward pass and instantiates multiple concurrent 1F1B pipelines with the optimal number determined analytically based on the pipeline depth.
- We introduce a gradient accumulation update strategy that aggregates gradients across multiple minibatches and applies them in a single update. This reduces communication frequency and ensures that parameter mismatch is bounded to at most one step, affecting only a number of minibatches equal to the pipeline depth.
- We incorporate ZeRO into the multi-directional schedule, ensuring that only one replica of each stage maintains the full optimizer state. This eliminates the redundant memory cost of naive pipeline replication while preserving efficiency.
- Extensive experiments demonstrate that AMDP achieves up to 17% higher throughput compared to state-of-the-art baselines, with marginal additional memory overhead, significantly accelerating large-scale language model training.

2 Related Work

Synchronous Pipeline Parallelism. Synchronous pipeline methods launch a new minibatch only after completing the previous update. GPipe (Huang et al., 2019) introduced microbatch-based pipeline training with an "all-forward-then-all-backward" schedule, achieving stable convergence but incurring high activation memory. DAPPLE (Fan et al., 2021) reduced memory cost by adopting a 1F1B schedule that releases activations earlier, and Interleaved 1F1B (Narayanan et al., 2021b) further decreased bubble size via finer stage partitioning. Chimera improved utilization by running two counter-directed pipelines whose idle periods overlap, and zero-bubble pipeline parallelism (ZB-V) (Qi et al., 2024) theoretically eliminates bubbles by splitting backward computation. Despite these advances, synchronous schemes remain fundamentally constrained by sequential dependencies, leading to persistent bubbles and limited throughput.

Asynchronous Pipeline Parallelism. Asynchronous methods remove bubbles by updating parameters immediately after each backward pass, at the cost of parameter mismatch. PipeDream (Narayanan et al., 2019) addresses this via parameter stashing, caching forward-pass weights for

Table 1: Comparison of representative pipeline parallelism approaches. d denotes the pipeline depth, n denotes the number of microbatches per minibatch, and M_θ/M_a indicate the weight/activation memory footprint of a single stage.

Approach	Bubble Ratio	Weight Memory	Peak Activation Memory	Extra Mem., Comp., and Comm.	Convergence
DAPPLE	$(d-1)/(n+d-1)$	M_θ	$n \cdot M_a$	$[\times, \times, \times]$	Stable
Inter-1F1B	$(d-1)/(2n+d-1)$	M_θ	$d \cdot M_a$	$[\checkmark, \times, \checkmark]$	Stable
Chimera	$(d-2)/(2n+d-2)$	$2M_\theta$	$d \cdot M_a$	$[\checkmark, \times, \checkmark]$	Stable
ZB-V	$\approx 0\%$	M_θ	$(2d-1) \cdot M_a$	$[\times, \checkmark, \times]$	Stable
PipeDream	$\approx 0\%$	$[M_\theta, d \cdot M_\theta]$	$d \cdot M_a$	$[\checkmark, \times, \times]$	Unstable
PipeDream-2BW	$\approx 0\%$	$2M_\theta$	$d \cdot M_a$	$[\checkmark, \times, \times]$	Moderate
XPipe	$\approx 0\%$	M_θ	$d \cdot M_a$	$[\checkmark, \checkmark, \times]$	Moderate
vNAG	$\approx 0\%$	$[M_\theta, d \cdot M_\theta]$	$d \cdot M_a$	$[\checkmark, \times, \times]$	Moderate
AMDP (ours)	$\approx 0\%$	M_θ	$d \cdot M_a$	$[\checkmark, \times, \checkmark]$	Near-stable

reuse during backpropagation; PipeDream-2BW (Narayanan et al., 2021a) reduces stash memory by maintaining two parameter versions per device. These approaches avoid mismatch but introduce delayed gradients. Parameter prediction offers an alternative: SpecTrain (Chen et al., 2018) predicts future parameters via momentum states, and XPipe (Guan et al., 2019) improves accuracy using Adam-based prediction. Recent work vNAG (Ajanthan et al., 2025) applies momentum-based extrapolation to alleviate staleness in PipeDream-like pipelines. Prediction-based methods mitigate delay but remain limited by the simplicity of their forecasting rules, which can produce large deviations between predicted and actual weights.

Table 1 summarizes the trade-offs of representative pipeline approaches in terms of bubble ratio, memory overhead, convergence stability, and additional costs in memory, computation, and communication. Synchronous approaches (e.g., DAPPLE, Inter-1F1B, Chimera, and ZB-V) provide stable convergence but incur either high bubble ratios or substantial memory usage. Asynchronous approaches (e.g., PipeDream, PipeDream-2BW, and XPipe) eliminate bubbles but suffer from elevated memory requirements and unstable convergence due to parameter mismatch.

Optimization and Memory Techniques. Gradient accumulation (Hermans et al., 2017; Li et al., 2020) is widely used in large-scale model training, particularly under pipeline parallelism, where gradients from multiple microbatches are aggregated into a single update. This enables large-batch training with manageable per-device memory, though under synchronous schedules it can exacerbate pipeline bubbles and reduce efficiency. While asynchronous PipeDream-style methods may increase the update interval to slow down parameter changes, this strategy does not impose any structural bound on forward–backward mismatch, which typically still grows with pipeline depth. Memory-reduction methods such as ZeRO (Rajbhandari et al., 2020; Zhao et al., 2023) partition optimizer states, gradients, and parameters across devices, substantially lowering memory overhead while preserving correctness. AMDP leverages these techniques in an asynchronous, multi-directional setting: gradient accumulation suppresses mismatch within each depth-sized window, and ZeRO eliminates redundant optimizer-state replication across pipelines. Together, these components allow AMDP to achieve both high throughput and near-stable convergence.

3 METHODOLOGY

As highlighted in Table 1, the limitations of existing pipeline parallelism approaches motivate the design of AMDP, an asynchronous multi-directional pipeline parallelism framework that addresses efficiency, memory usage, and convergence stability. It comprises four interrelated components: *parameter mismatch control* that structurally limits inconsistencies between forward and backward passes, a *multi-directional scheduling scheme* that maximizes hardware utilization, a *gradient accumulation update strategy* that reduces communication overhead and bounds parameter mismatch, and a *zero-redundancy optimizer* that minimizes redundant memory usage across pipelines.

3.1 Parameter Mismatch Control

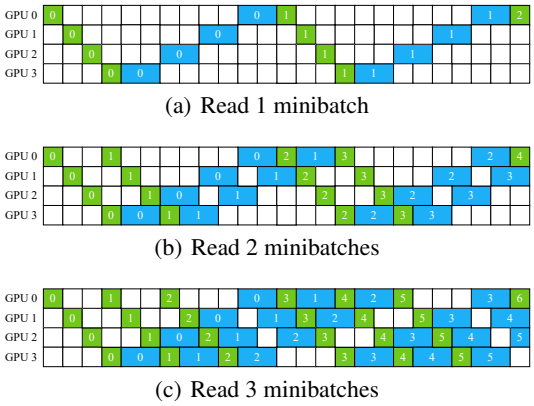
In asynchronous pipeline parallelism, a minibatch always performs its forward pass at a stage before receiving the corresponding backward pass. As a result, the *parameter mismatch* at a given stage (i.e., the number of parameter updates occurring between a minibatch's forward and backward passes) equals the number of minibatches that have completed forward passes but have not yet started backward passes when the current minibatch enters. In a steady 1F1B schedule, each stage issues a new forward only after completing the previous backward, so the mismatch is exactly "the number of minibatches read before the first backward, minus one."

This quantity is determined by two structural limits. First, to maintain bubble-free execution during backward propagation, each stage must execute one additional forward relative to its successor before observing its first backward. Hence stage i can read at most $d - i$ minibatches in this warm-up phase, where d is the pipeline depth. Second, the number of minibatches read at any stage cannot exceed the number read by stage 0, denoted by n. Combining both constraints yields the following structural bound:

Lemma 1 (Mismatch Bound) *For any stage $i \in \{0, \ldots, d-1\}$, the parameter mismatch satisfies:*

$$\text{mismatch}(i) = \min(n,\, d - i) - 1. \tag{1}$$

This simple expression explains the instability of existing asynchronous pipelines. When prior work sets $n = d$ to eliminate bubbles, the mismatch becomes $\text{mismatch}(i) = d-i-1$, which grows linearly with depth and results in increasingly stale gradients.



(a) Read 1 minibatch

(b) Read 2 minibatches

(c) Read 3 minibatches

Figure 2: Parameter mismatch at stage 0 increases linearly with the number of minibatches read before the first backward pass. The case with the number set to 4 is shown in Figure 1(b).

Figures 2 and 1(b) illustrate this effect. With $n = 1$, mismatch is zero; with $n = 2$, mismatch becomes one step (e.g., a single update from minibatch 2 occurs between the forward and backward of minibatch 3 at stage 0). As n increases, mismatch grows proportionally.

To mitigate mismatch, one would ideally minimize n. However, using $n = 1$ requires launching many parallel pipelines to avoid large bubbles, substantially increasing memory usage. Instead, AMDP enforces $n = 2$ by design. From Lemma 1, this guarantees $\text{mismatch}(i) \leq 1$ for all stages i, regardless of pipeline depth, multi-node deployment, or multi-directional stage placement (details provided in the Appendix B). A fixed one-step mismatch has important convergence implications:

Theorem 1 (Near-Synchronous Convergence of AMDP). Let F be an L-smooth objective with stochastic gradients of variance at most σ^2. Suppose AMDP enforces a strict one-step mismatch bound $\tau(t) \leq 1$ for all t, and choose a stepsize satisfying $\eta L \leq 1$. Then the AMDP iterates satisfy:

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla F(\theta_t)\|^2\right] \leq \frac{2(F(\theta_0) - F^\star)}{\eta T} + \eta L \sigma^2 + O(\eta^2), \tag{2}$$

matching the convergence rate of synchronous SGD up to an $O(\eta^2)$ perturbation.

Proof. We follow the standard smooth nonconvex SGD analysis (Tsitsiklis, 1994; Bertsekas & Tsitsiklis, 2015; Lian et al., 2015; Zheng et al., 2017) while adopting the delay-one gradient. Since AMDP enforces $\tau(t) \leq 1$, the gradient evaluated at iteration t is $g(\theta_{t-1}) = \nabla F(\theta_t) + O(\eta)$.This deviation induces a parameter discrepancy $\Delta_t = \theta_t - \theta_t^{\text{sync}}$ that evolves as $\Delta_{t+1} = \Delta_t - \eta\left(g(\theta_{t-1}) - \nabla F(\theta_t^{\text{sync}})\right)$. By substituting the gradient error and applying L-smoothness, one derives a recursion showing $\mathbb{E}[\|\Delta_t\|^2] = O(\eta^2)$. The Lipschitz continuity of ∇F then bounds the gradient deviation by $O(\eta^2)$, and combining with the standard synchronous SGD bound via the inequality $\|a\|^2 \leq 2\|b\|^2 + 2\|a-b\|^2$ yields the stated result. A complete derivation is provided in Appendix B.3. □

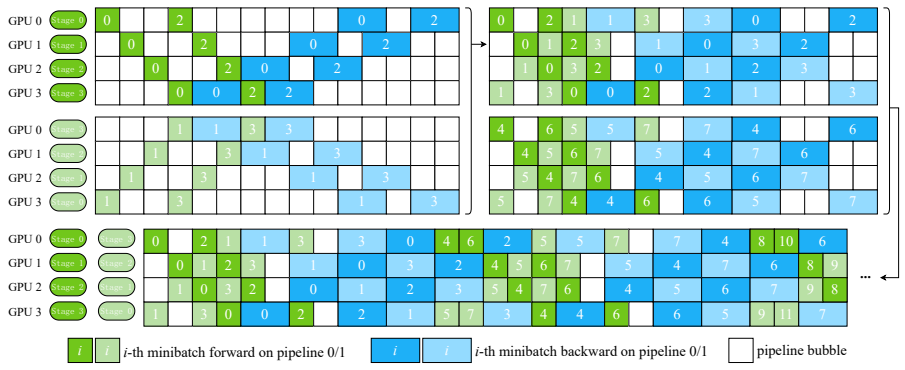


Figure 3: Illustration of AMDP bidirectional schedule, with 4 model stages and 4 pipeline devices deployed. Both trailing bubbles from minibatches 0–3 and leading bubbles for minibatches 4–6 are eliminated.

3.2 Multi-Directional Scheduling

Reducing the number of minibatches read by stage 0 improves convergence but induces significant bubbles, as shown in Figure 2. AMDP avoids this overhead by employing *multi-directional scheduling*: multiple pipelines with complementary directions execute in parallel and fill one another's idle periods. Figure 3 shows a 4-GPU example with two counter-directed pipelines, each GPU hosting two stages. Pipelines process distinct minibatches independently, and GPUs holding the same stage synchronize gradients via all-reduce before applying updates.

AMDP constructs multi-directional schedules in three steps: (i) determining the number of pipelines according to pipeline depth, (ii) assigning their directions while avoiding device conflicts, and (iii) filling residual bubbles. A single bidirectional schedule is insufficient for large d, so the number of pipelines must scale inversely with the active ratio r—the fraction of time a single pipeline keeps a GPU busy. If stage 0 reads only one minibatch, the pipeline is nearly serialized ($r = 1/d$). With two minibatches, the number of concurrently active GPUs doubles while per-GPU work remains unchanged, giving $r = 2/d$. Since AMDP fixes stage 0 to read two minibatches, setting the number of pipelines to $d/2$ fully eliminates bubbles. Figure 4 shows this scheme for $d = 8$.

Pipeline directions follow the Chimera-style mapping (Li & Hoefler, 2021): for pipeline j, if j is even, stage i maps to GPU $(2j + i) \mod d$; if j is odd, to $(2j - i + d + 1) \mod d$. For example, in Figure 4, pipeline 0 traverses GPUs $[0, 1, 2, \ldots, 7]$ while pipeline 1 traverses them in reverse order as $[3, 2, 1, 0, 7, 6, 5, 4]$. However, unlike Chimera, AMDP must handle the asymmetry between forward and backward costs, which can create conflicts when multiple pipelines submit work to the same GPU. AMDP resolves these via a simple FIFO rule: the earliest operation proceeds and the later one is deferred (e.g., the conflict between the two backward passes on GPU 2 in Figure 3).

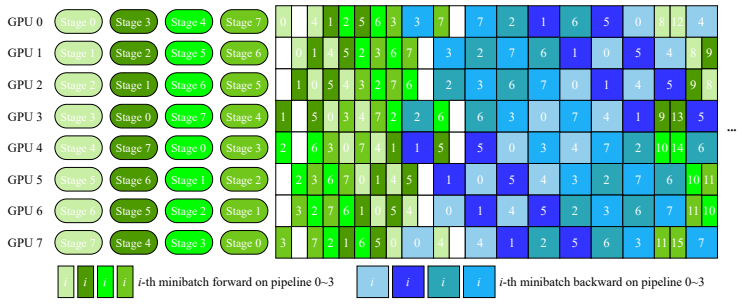
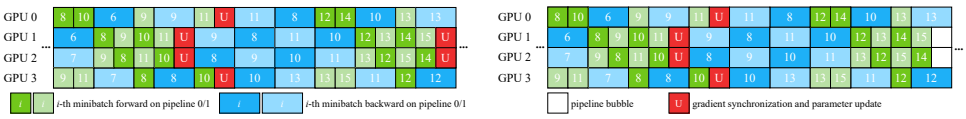


Figure 4: Illustration of AMDP multi-directional scheduling for d=8. Four (i.e., $d/2$) pipelines with distinct directions is sufficient to filling bubbles.



(a) d=4, threshold=4 (b) d=4, threshold=8

Figure 5: Gradient accumulation update strategy in AMDP. (a) Each minibatch experiences a one-step parameter mismatch. (b) Only the first d minibatches in the window incur mismatch.

After direction assignment, each block of d minibatches contains three types of bubbles. The middle bubble arises from inherent forward/backward imbalance and cannot be removed. It becomes dominant only when the imbalance is extreme or at very large d. The leading and trailing bubbles are scheduling gaps and are eliminated via controlled minibatch preloading. At the boundary of each d-minibatch segment, AMDP injects additional forward passes to fill idle slots. The number of minibatches inserted equals the floor of the backward-to-forward cost ratio. For instance, in Figure 3 (top-right), preloading minibatches 4 and 6 removes both the trailing bubble of minibatches 0–3 and the leading bubble of minibatches 4–6, yielding balanced and efficient pipeline execution.

3.3 Gradient Accumulation Updates

Naively applying the asynchronous multi-directional schedule introduces two drawbacks: (i) an all-reduce after every backward pass, incurring heavy communication overhead, and (ii) disruption of the 1F1B schedule by bubble-filling, which can lead to multi-step mismatches. For example, as shown in Figure 3, two parameter updates (from minibatch 2 and minibatch 4) occur between the forward and backward passes of minibatch 6 on GPU 0.

To address these issues, AMDP introduces a gradient accumulation update strategy. Instead of updating after every backward pass, gradients are accumulated across multiple minibatches until a predefined threshold is reached. At that point, the accumulated gradients are reduced across devices and applied during the next bubble. This design lowers communication frequency, thereby mitigating communication overhead. Furthermore, it ensures that only the first d minibatches experience a bounded one-step mismatch per update, which enhances training convergence. In practice, the threshold is typically set much larger than d, rendering the effect of mismatch negligible. This makes AMDP fundamentally different from PipeDream-style methods, where mismatch typically grows with the number of pipeline stages.

Figure 5 illustrates the gradient accumulation update strategy in AMDP. For clarity, non-essential computation steps are omitted to highlight the effect of accumulation threshold on parameter updates. When the threshold equals the pipeline depth (Figure 5(a)), each minibatch is subject to a one-step parameter mismatch. With a larger threshold (Figure 5(b)), only the first d minibatches in each accumulation window are exposed to mismatch, while subsequent minibatches use consistent parameters. For example, when accumulating gradients from minibatches 8 to 15, parameter updates occur between the forward and backward passes only for minibatches 8–11.

3.4 Zero Redundancy Optimizer

Deploying multiple pipelines increases memory usage, as each device must store parameters, gradients, and optimizer states (e.g., momentum and variance in Adam (Kingma, 2014; Loshchilov, 2017)) for multiple stages. To reduce optimizer-state memory usage without incurring the latency and contention of CPU offloading, AMDP incorporates ZeRO (Rajbhandari et al., 2020). In our design, GPU i is solely responsible for updating parameters of stage i, and the optimizer for stage i resides exclusively on GPU i. During updates, GPUs holding replicas of stage i send their gradients to GPU i for reduction, replacing the all-reduce used in naive designs; once the update is applied, the new parameters are broadcast back to all replicas. This scheme reduces optimizer state memory on each GPU to $2/d$ of the naive requirement, scaling inversely with the number of pipelines. Furthermore, the communication pattern becomes a reduce followed by a broadcast, which has the same total cost as all-reduce and therefore introduces no additional communication overhead. Synchronization occurs only once per update and does not grow with the number of pipelines.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Hardware and Software. All experiments are conducted on a Linux server running Ubuntu 20.04 (kernel 5.15) equipped with 8 NVIDIA A800 80GB GPUs interconnected via NVLink 3.0 (400 GB/s), dual-socket 128-core CPUs, and 1 TB DDR4 memory. All methods use the NVIDIA PyTorch container `nvcr.io/nvidia/pytorch:24.03-py3`, ensuring a consistent software environment across baselines. A detailed description of the server is provided in Appendix A.

Baselines and Implementation. We evaluate AMDP against both synchronous and asynchronous baselines: synchronous approaches include DAPPLE (Fan et al., 2021), interleaved 1F1B (Inter-1F1B) (Narayanan et al., 2021b), Chimera (Li & Hoefler, 2021), and zero bubble pipeline parallelism (ZB-V) (Qi et al., 2024); asynchronous approaches include PipeDream (Narayanan et al., 2019), PipeDream-2BW (Narayanan et al., 2021a), XPipe (Guan et al., 2019), and vNAG (Ajanthan et al., 2025). For DAPPLE and Inter-1F1B, we adopt the open source implementation from Megatron-LM (NVIDIA, 2024); for ZB-V and vNAG, we use the codes released by the authors. AMDP, Chimera, and other asynchronous approaches are implemented by us on top of Megatron-LM.

Models and Datasets. We test two representative architectures: a GPT-style autoregressive language model and a BERT-style bidirectional encoder, with detailed configurations in Table 2. The GPT-style model is trained on OpenWebText (Peterson et al., 2019), while the BERT-style model is trained on Wikipedia (Devlin et al., 2019).

Table 2: Configurations of benchmark models.

Configuration	GPT-style model	BERT-style model
# Layers	48	32
# Attention Heads	25	32
Hidden Size	1600	1600
Sequence Length	1024	1024
# Parameters	1557686400	1036179458

Metrics. We evaluate the performance of each approach using three key metrics: throughput, memory footprint, and training convergence. Throughput, defined as the number of tokens processed per second, reflects the computational efficiency and hardware utilization of each approach. Memory footprint is assessed in terms of both its distribution and peak consumption across devices. Training convergence is evaluated along two dimensions: loss versus iteration and loss versus wall-clock time. The former highlights the intrinsic effect of each approach on convergence behavior, while the latter illustrates its practical efficiency and applicability in real training scenarios.

Training Settings. Unless otherwise stated, we set the microbatch size to 4 for all approaches. All experiments employ AdamW (Loshchilov, 2017) with mixed-precision training (NVIDIA, 2019). Learning rates and other hyperparameters follow standard configurations commonly used in in GPT/BERT training to ensure fair comparability. Because the author-released implementation of vNAG does not run under our model settings, we adapt AMDP to match their configuration and report the comparison results in Appendix C.

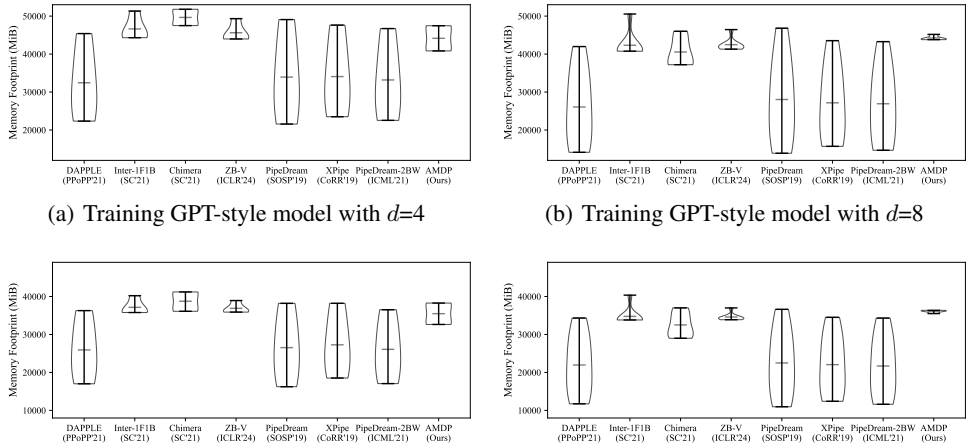
4.2 MAIN RESULTS

Throughput. Table 3 summarizes throughput results under varying pipeline depths d and update batch sizes b. Key findings are: (1) AMDP consistently achieves the highest throughput across both models and all configurations, outperforming PipeDream-2BW by up to 17%. On GPT-style models, AMDP yields relative gains of 1.01–1.17×, while on BERT-style models, gains range from 1.02–1.13×. These improvements arise because language models exhibit uneven stage partitioning (e.g., large embeddings in first/last stages), creating imbalance-induced bubbles that AMDP mitigates via multi-directional scheduling, reducing bubble time to $2/d$. (2) The relative advantage of AMDP grows with pipeline depth and update batch size. Larger d exacerbates stage imbalance, while larger b reduces update frequency, both amplifying AMDP's efficiency. (3) Among asynchronous baselines, PipeDream-2BW delivers the strongest throughput, consistently outperforming XPipe and PipeDream. This aligns with its design of double-buffered updates, which eliminate bubbles without additional computation.

Memory Footprint. Figure 6 illustrates the memory footprint distribution for both models under two pipeline depths. We summarize the following observations: (1) AMDP achieves more balanced per-GPU memory utilization, which is a direct consequence of its multi-directional scheduling that

Table 3: Throughput comparison (in ktokens/s) . The best and second-best results are **bolded** and underlined. d denotes the pipeline depth and b indicates the number of samples processed per update.

Model	d	b	DAPPLE	Inter-1F1B	Chimera	ZB-V	PipeDream	XPipe	PipeDream-2BW	AMDP
GPT -style	4	16	25.2	35.4	25.3	26.3	34.5	38.5	38.6	**39.1**
	4	64	36.3	39.8	30.1	30.9	34.5	40.7	41.0	**42.1**
	8	32	44.0	57.0	46.3	45.2	61.0	66.0	70.3	**75.5**
	8	128	61.9	67.5	56.0	54.4	61.0	69.7	71.6	**83.7**
BERT -style	4	16	26.0	34.7	29.3	28.9	37.2	37.8	41.0	**41.6**
	4	64	37.1	41.0	34.5	33.8	37.2	41.7	42.9	**43.6**
	8	32	45.2	37.5	52.8	40.4	65.0	73.6	74.3	**78.5**
	8	128	64.8	58.8	63.8	54.1	65.0	75.6	75.8	**86.1**



(a) Training GPT-style model with d=4

(b) Training GPT-style model with d=8

(c) Training BERT-style model with d=4

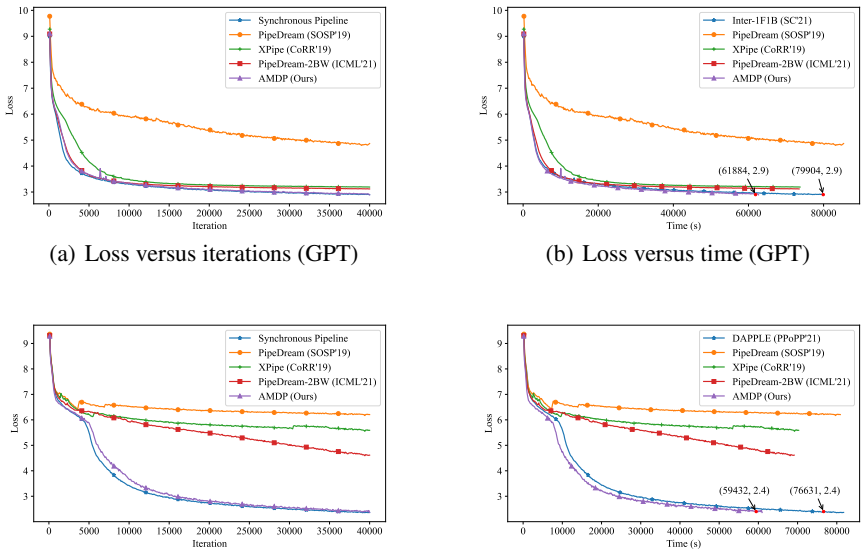
(d) Training BERT-style model with d=8

Figure 6: Memory footprint comparison.

spreads the workload more evenly across devices. (2) Inter-1F1B exhibits the largest peak memory footprint, as the first GPU must retain additional activations, amounting to 1.5 microbatches for $d = 4$ and 3.5 microbatches for $d = 8$ compared with other approaches. While asynchronous methods generally maintain multiple parameter versions, prior work (Lin et al., 2025) shows that activation storage dominates the memory footprint in modern Transformers. This explains why Inter-1F1B suffers from significantly higher peak memory. (3) AMDP incurs slightly a higher peak memory footprint than PipeDream-2BW and XPipe due to the need to temporarily retain additional parameters and gradients. However, this overhead is minor relative to the improvements in convergence stability and does not hinder scalability to larger clusters.

Training Convergence. Figure 7 presents the convergence results under $d = 8$, $b = 128$, and a learning rate of 1.5×10^{-4} over 40k iterations. Synchronous approaches (e.g., DAPPLE, Inter-1F1B, Chimera, and ZB-V) exhibit nearly identical behavior and are collectively reported as "Synchronous Pipeline". For the loss–time comparisons, we highlight the best synchronous baseline: Inter-1F1B for GPT-style model and DAPPLE for BERT-style model.

Key observations are: (1) AMDP closely follows the convergence trajectory of synchronous baselines. The small early gap is expected, as the first few iterations accumulate gradients under mild mismatch; once gradients stabilize, AMDP's higher utilization enables it to catch up rapidly. After 40k iterations, AMDP reaches a training loss of 2.90 on the GPT-style model (vs. 2.88 for Inter-1F1B) and 2.36 on the BERT-style model (matching DAPPLE), confirming the effectiveness of the one-step mismatch bound. (2) AMDP significantly shortens the time to reach target losses: 23% faster than Inter-1F1B on GPT-style model at loss 2.9, and 22% faster than DAPPLE on BERT-style model at loss 2.4. These gains stem from AMDP's higher throughput while maintaining convergence, highlighting its practical value for efficient large-scale models training.



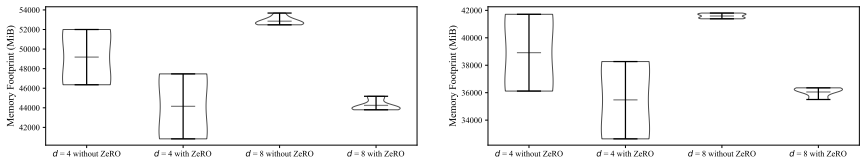
(a) Loss versus iterations (GPT)

(b) Loss versus time (GPT)

(c) Loss versus iterations (BERT)

(d) Loss versus time (BERT)

Figure 7: Training convergence comparison.



(a) Training GPT-style model

(b) Training BERT-style model

Figure 8: Memory footprint results of ablation study.

4.3 Ablation Study

We assess the effect of ZeRO by disabling it under the same setup as the convergence study. Figure 8 shows memory results: without ZeRO, average GPU memory usage rises by 11.4% and 19.4% for GPT-style model (at $d = 4$ and $d = 8$), and by 9.7% and 15.4% for BERT-style model, respectively. Table 4 further indicates that enabling ZeRO improves throughput by approximately 4%, as optimizer state sharding reduces redundant computations. These findings confirm that ZeRO not only alleviates memory pressure but also delivers modest performance gains, making it a crucial component of AMDP's scalability.

Table 4: Throughput results of ablation study.

Configuration	GPT-style model	BERT-style model
without ZeRO	80.3 (-4.1%)	82.7 (-3.9%)
with ZeRO	**83.7**	**86.1**

5 Conclusions

In this paper, we propose **AMDP**, an **a**synchronous **m**ulti-**d**irectional **p**ipeline-parallel training scheme that achieves up to 17% higher throughput compared to state-of-the-art approaches, with only marginal additional memory overhead. By structurally bounding forward–backward mismatch and combining multi-directional scheduling with ZeRO-based state partitioning, AMDP delivers near-synchronous convergence while substantially improving utilization. In the future, we plan to investigate mismatch-aware learning-rate adaptation technique, which may further enhance optimization stability under mild asynchrony.

6 Ethics Statement

Our work focuses solely on scientific problems and does not involve human subjects, animals, or environmentally sensitive materials. We foresee no ethical risks or conflicts of interest.

7 Reproducibility Statement

We have rigorously formalized the proposed pipeline scheduling strategy, model architecture, and training objectives in the main text using tables, figures, and detailed descriptions. All experiments are conducted on publicly available datasets (e.g., Wikipedia, OpenWebText) for pretraining. We provide the reproducibility details in the Appendix A, including full hardware/software description, code implementation, and experiment workflow. We provide our source code in an anonymous link: https://anonymous.4open.science/r/Megatron-AMDP-BB23, which will be publicly available upon acceptance.

References

Thalaiyasingam Ajanthan, Sameera Ramasinghe, Yan Zuo, Gil Avraham, and Alexander Long. Nesterov method for asynchronous pipeline parallel optimization. In *Proceedings of the International Conference on Machine Learning*, 2025.

Dimitri Bertsekas and John Tsitsiklis. *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Chi-Chung Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. Efficient and robust parallel DNN training through model parallelism on multi-GPU platform. *arXiv preprint arXiv:1809.02839*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186, 2019.

Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, et al. DAPPLE: A pipelined data parallel approach for training large models. In *Proceedings of the ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pp. 431–445, 2021.

Lei Guan, Wotao Yin, Dongsheng Li, and Xicheng Lu. XPipe: Efficient pipeline model parallelism for multi-GPU DNN training. *arXiv preprint arXiv:1911.04610*, 2019.

Lei Guan, Dong-Sheng Li, Jiye Liang, Wen-Jian Wang, Ke-shi Ge, et al. Advances of pipeline model parallelism for deep learning training: An overview. *Journal of Computer Science and Technology*, 39(3):567–584, 2024.

Joeri R Hermans, Gerasimos Spanakis, and Rico Möckel. Accumulated gradient normalization. In *Proceedings of the Asian Conference on Machine Learning*, pp. 439–454, 2017.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, et al. GPipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in Neural Information Processing Systems*, 32, 2019.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, et al. PyTorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

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*, pp. 1–14, 2021.

Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. *Advances in Neural Information Processing Systems*, 28, 2015.

Junfeng Lin, Ziming Liu, Yang You, Jun Wang, Weihao Zhang, et al. WeiPipe: Weight pipeline parallelism for communication-effective long-context large model training. In *Proceedings of the ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pp. 225–238, 2025.

I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, et al. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pp. 1–15, 2019.

Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel DNN training. In *Proceedings of the International Conference on Machine Learning*, pp. 7937–7947, 2021a.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, 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*, pp. 1–15, 2021b.

NVIDIA. Automatic mixed precision for deep learning. https://developer.nvidia.com/automatic-mixed-precision, 2019.

NVIDIA. Nvidia collective communications library. https://developer.nvidia.com/nccl, 2020.

NVIDIA. Megatron-LM & Megatron-Core. https://github.com/NVIDIA/Megatron-LM, 2024.

Joshua Peterson, Stephan Meylan, and David Bourgin. OpenWebText dataset. https://github.com/jcpeterson/openwebtext, 2019.

Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble (almost) pipeline parallelism. In *Proceedings of the The International Conference on Learning Representations*, 2024.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16, 2020.

John N Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine learning*, 16 (3):185–202, 1994.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *Proceedings of the International Conference on Machine Learning*, pp. 4120–4129, 2017.

A Implementation Details

A.1 Server and System Configuration

All experiments are conducted on a dedicated Linux server running Ubuntu 20.04 with kernel 5.15. The compute node contains:

- **CPUs:** Dual-socket AMD EPYC 7763 (128 physical cores in two socket).
- **Memory:** 1 TB DDR4-3200, NUMA-balanced across both sockets.
- **GPUs:** 8 NVIDIA A800 80GB SXM5 GPUs, connected via NVLink 3.0 in a hybrid cube-mesh topology, providing 400 GB/s bidirectional bandwidth.
- **Networking:** Local experiments use NCCL's NVLink transport; multi-node configuration (not used in main experiments) supports dual 200 Gbps HDR InfiniBand.
- **Storage:** Parallel NVMe scratch with 7.4 GB/s sequential read bandwidth for fast dataset preprocessing.

The software stack is based on the NVIDIA PyTorch container with tag `24.03-py3`[1], consisting of: PyTorch 2.3.0, CUDA 12.4, cuDNN 9.0.0, NCCL 2.20, Apex, TransformerEngine 1.4.0, and Python 3.10.

A.2 Code Modifications

We summarize the key extensions made to Megatron-LM to support asynchronous multi-directional execution:

- **Pipeline Engine Extensions.** We extend the Megatron pipeline engine to support asynchronous execution by enabling `async_op` for gradient reductions and parameter updates and introducing an `async_schedules` module. This module implements the scheduling logic for PipeDream, PipeDream-2BW, and AMDP, including forward/backward task queues, causal dependency enforcement, and multi-directional stage assignment.
- **Gradient-Accumulation Mechanism.** AMDP adopts a window-based gradient accumulation mechanism that aligns with ZeRO partitioning. Gradients are accumulated only on the parameter-owner rank, while non-owner ranks store only the transient gradient tensors needed for the reduce operation. This avoids redundant memory usage, guarantees correct accumulation over the update window, and ensures that communication occurs exactly at the intended synchronization boundaries.
- **ZeRO Integration Logic.** We modify the initialization and optimizer-step routines to support multi-directional pipelines under ZeRO. Optimizer states are materialized exclusively on the owner rank of each parameter shard. Each all-reduce is replaced by a *reduce* followed by a *broadcast* of identical communication complexity, triggered once per update window and aligned with AMDP's scheduling.

A.3 Experiment Workflow

For reproducibility, we outline the common workflow used for all baselines and AMDP:

- **Data Preprocessing.** Training data are first stored as loose JSON (one text record per line) and then converted into Megatron-LM's binary format following the official preprocessing pipeline. All baselines use identical datasets.
- **Pretraining.** Pretraining follows the standard Megatron-LM workflow with additional arguments (e.g., `--enable-fourdirectional-pipeline`). We provide `pretrain_gpt_distributed_async.sh` for AMDP and analogous scripts for baseline methods in the *examples* directory.

[1] nvcr.io/nvidia/pytorch:24.03-py3

- **Logging and Profiling.** Throughput, per-GPU memory, and iteration-time breakdowns are collected using Megatron-LM's profiler with auxiliary custom hooks. Experiments across different depths (4–8) and model families (GPT, BERT) share the same workflow to ensure consistency.

B Convergence Analysis of AMDP

This part presents a formal convergence analysis of AMDP, demonstrating that it achieves *near-synchronous convergence* under bounded parameter mismatch. Drawing on classical asynchronous SGD theory (Tsitsiklis, 1994; Bertsekas & Tsitsiklis, 2015; Lian et al., 2015; Zheng et al., 2017), we show that AMDP's parameter trajectory deviates from synchronous SGD only by a second-order perturbation of order $O(\eta^2)$. This results from AMDP's structural enforcement of a maximum delay of one update step ($\tau_{\max} = 1$) between the forward and backward passes of any minibatch. We further prove that this delay bound remains invariant under variations in pipeline depth, multi-node deployment, and multi-directional stage permutations.

B.1 Preliminaries

Let $F(\theta) = \mathbb{E}_\xi[f(\theta;\xi)]$ denote the expected loss function, where $\theta \in \mathbb{R}^p$ represents the model parameters. We adopt the following standard assumptions in stochastic optimization:

A1. (L**-smoothness**) The loss function satisfies $\|\nabla F(x) - \nabla F(y)\| \le L\|x - y\|$ for all x and y.

A2. (**Unbiased stochastic gradients with bounded variance**) The stochastic gradient estimator $g(\theta;\xi)$ satisfies $\mathbb{E}_\xi[g(\theta;\xi)] = \nabla F(\theta)$ and $\mathbb{E}_\xi\|g(\theta;\xi) - \nabla F(\theta)\|^2 \le \sigma^2$.

A3. (**Bounded second moment**) The second moment of the stochastic gradient is bounded by $\mathbb{E}\|g(\theta;\xi)\|^2 \le G^2$ for all iterates.

In AMDP, parameter updates follow the rule:

$$\theta_{t+1} = \theta_t - \eta\, g(\theta_{t-\tau(t)};\xi_t), \tag{3}$$

where $\tau(t) \in \{0, 1\}$ denotes the *logical delay*, which is the number of parameter updates between the forward and backward passes of the same minibatch. This delay is controlled by AMDP's scheduling semantics. For comparison, the synchronous SGD iterating with the same samples is:

$$\theta_{t+1}^{\text{sync}} = \theta_t^{\text{sync}} - \eta\, g(\theta_t^{\text{sync}};\xi_t). \tag{4}$$

We define the deviation between the two sequences as $\Delta_t := \theta_t - \theta_t^{\text{sync}}$.

B.2 Bounded Parameter Mismatch

Lemma 2 (Bounded Parameter Mismatch in AMDP) *Let d be the pipeline depth, and let n denote the number of minibatches that stage 0 reads before receiving its first backward message. For any stage $i \in \{0, \dots, d-1\}$, the number of parameter updates between the forward and backward passes of a minibatch at stage i is:*

$$\text{mismatch}(i) = \min(n, d - i) - 1. \tag{5}$$

In AMDP, we set $n = 2$, which implies $\text{mismatch}(i) \le 1$ for all stages i.

Proof 1 *A stage cannot process more than n minibatches before receiving the first backward message from stage 0. Similarly, it cannot process more than $d - i$ minibatches before receiving the first backward from its downstream neighbor, as each downstream stage contributes at most one additional forward pass during pipeline filling. Thus, stage i processes at most $\min(n, d - i)$ forward passes before any backward pass arrives. The mismatch count is this number minus one (excluding the forward pass of the current minibatch), yielding the stated bound.*

B.3 Near-Synchronous Convergence Analysis

We now establish that AMDP achieves a convergence rate comparable to synchronous SGD, with a second-order perturbation, under a bounded delay of $\tau_{\max} = 1$.

Theorem 1 (Near-Synchronous Convergence of AMDP) *Under Assumptions A1–A3 and with AMDP enforcing a maximum delay $\tau_{\max} = 1$ (via Lemma 2), if the learning rate satisfies $\eta L \leq 1$, then the AMDP updates satisfy:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla F(\theta_t)\|^2\right] \leq \frac{2\left(F(\theta_0) - F^\star\right)}{\eta T} + \eta L\sigma^2 + O(\eta^2), \tag{6}$$

where the $O(\eta^2)$ term captures the second-order perturbation induced by AMDP's delay-one dynamics.

Proof 2 *Define the deviation $\Delta_t = \theta_t - \theta_t^{\text{sync}}$. Subtracting the synchronous update from AMDP's update gives:*

$$\Delta_{t+1} = \Delta_t - \eta\left(g(\theta_{t-\tau};\xi_t) - g(\theta_t^{\text{sync}};\xi_t)\right). \tag{7}$$

Taking conditional expectations and expanding yields:

$$\begin{aligned}\mathbb{E}\|\Delta_{t+1}\|^2 = \mathbb{E}\|\Delta_t\|^2 - 2\eta\,\mathbb{E}\left\langle\Delta_t, \nabla F(\theta_{t-\tau}) - \nabla F(\theta_t^{\text{sync}})\right\rangle \\ + \eta^2\,\mathbb{E}\|\nabla F(\theta_{t-\tau}) - \nabla F(\theta_t^{\text{sync}})\|^2 + \eta^2\,\mathbb{E}\|\xi_t'\|^2,\end{aligned} \tag{8}$$

where ξ_t' is zero-mean noise with $\mathbb{E}\|\xi_t'\|^2 \leq 4\sigma^2$.

By Young's inequality ($\alpha = 1$) and Lipschitz continuity (A1):

$$\|\nabla F(\theta_{t-\tau}) - \nabla F(\theta_t^{\text{sync}})\| \leq L\|\theta_{t-\tau} - \theta_t^{\text{sync}}\|. \tag{9}$$

Substituting into the expectation leads to the recursion:

$$\mathbb{E}\|\Delta_{t+1}\|^2 \leq (1+\eta)\,\mathbb{E}\|\Delta_t\|^2 + \eta(1+\eta)L^2\,\mathbb{E}\|\theta_{t-\tau} - \theta_t^{\text{sync}}\|^2 + 4\eta^2\sigma^2. \tag{10}$$

We bound the term $\mathbb{E}\|\theta_{t-\tau} - \theta_t^{\text{sync}}\|^2$ by decomposing:

$$\theta_{t-\tau} - \theta_t^{\text{sync}} = (\theta_{t-\tau} - \theta_{t-\tau}^{\text{sync}}) + \sum_{k=t-\tau}^{t-1}(\theta_k^{\text{sync}} - \theta_{k+1}^{\text{sync}}). \tag{11}$$

Applying the triangle inequality and $\tau \leq 1$:

$$\mathbb{E}\|\theta_{t-\tau} - \theta_t^{\text{sync}}\|^2 \leq 2\,\mathbb{E}\|\Delta_{t-\tau}\|^2 + 2\sum_{k=t-\tau}^{t-1}\mathbb{E}\|\theta_k^{\text{sync}} - \theta_{k+1}^{\text{sync}}\|^2. \tag{12}$$

From the synchronous update, $\theta_{k+1}^{\text{sync}} - \theta_k^{\text{sync}} = -\eta g(\theta_k^{\text{sync}};\xi_k)$, and by A3, $\mathbb{E}\|g(\theta_k^{\text{sync}};\xi_k)\|^2 \leq G^2$. Thus:

$$\mathbb{E}\|\theta_k^{\text{sync}} - \theta_{k+1}^{\text{sync}}\|^2 \leq \eta^2 G^2. \tag{13}$$

As $\tau \leq 1$, the sum in equation 12 has at most one term, so:

$$\mathbb{E}\|\theta_{t-\tau} - \theta_t^{\text{sync}}\|^2 \leq 2\,\mathbb{E}\|\Delta_{t-\tau}\|^2 + 2\eta^2 G^2. \tag{14}$$

Substituting into equation 10 gives:

$$\mathbb{E}\|\Delta_{t+1}\|^2 \leq A\,M_t + B\,\eta^2, \tag{15}$$

where $M_t := \max\left\{\mathbb{E}\|\Delta_t\|^2, \mathbb{E}\|\Delta_{t-1}\|^2\right\}$, and:

$$A := 1 + \eta + 2\eta(1+\eta)L^2, \quad B := 2(1+\eta)L^2G^2 + 4\sigma^2. \tag{16}$$

Solving the recursion with $M_0 = 0$ yields the steady-state bound:

$$M_t \leq \frac{B}{\alpha}\eta, \quad \textit{where } A = 1 + \alpha\eta \textit{ and } \alpha \leq 4L^2 + 1. \tag{17}$$

A refined analysis shows $\mathbb{E}\|\Delta_t\|^2 \leq C_\Delta \eta^2$*, with:*

$$C_\Delta = \frac{8L^2G^2 + 16\sigma^2}{4L^2 + 2}. \tag{18}$$

By Lipschitz continuity (A1):

$$\mathbb{E}\|\nabla F(\theta_t) - \nabla F(\theta_t^{\text{sync}})\|^2 \leq L^2\, \mathbb{E}\|\Delta_t\|^2 \leq L^2 C_\Delta \eta^2. \tag{19}$$

The synchronous SGD bound is:

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\|\nabla F(\theta_t^{\text{sync}})\|^2 \leq \frac{2(F(\theta_0) - F^\star)}{\eta T} + \eta L \sigma^2. \tag{20}$$

Using $\|a\|^2 \leq 2\|b\|^2 + 2\|a-b\|^2$*, we derive the AMDP bound:*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\|\nabla F(\theta_t)\|^2 \leq \frac{2(F(\theta_0) - F^\star)}{\eta T} + \eta L \sigma^2 + 2L^2 C_\Delta \eta^2. \tag{21}$$

The $O(\eta^2)$ *term is explicitly* $2L^2 C_\Delta \eta^2$*, completing the proof.*

Implications Choosing $\eta = \Theta(T^{-1/2})$ yields the standard $O(1/\sqrt{T})$ nonconvex stochastic optimization rate. The $O(\eta^2)$ deviation from synchronous SGD explains AMDP's empirically observed near-stable convergence.

B.4 Invariance of the Mismatch Bound

Proposition 1 (Invariance of the Mismatch Bound) *Assume AMDP preserves the causal ordering of update, reduce, and broadcast operations (e.g., through FIFO task queues and no logical reordering of updates). Then, the mismatch bound in Lemma 2 depends only on* n *and the logical stage index* i*. Consequently, for* $n = 2$*, the bound* $\text{mismatch}(i) \leq 1$ *remains invariant under:*

- *Any pipeline depth* d*,*
- *Any mapping of logical stages to physical GPUs,*
- *Any multi-node deployment with bounded communication latency,*
- *Any multi-directional scheduling permutation.*

Proof 3 *The mismatch count, defined as the number of forward passes a stage executes before receiving the corresponding backward pass of a given minibatch, is fundamentally a property of the logical scheduling sequence. This integer value is determined exclusively by the logical topology of the pipeline (which defines the dataflow dependencies) and the fixed preload parameter* n *at the initial stage (Stage 0), which controls the initial pipeline fill. The factors listed in the proposition (pipeline depth, physical GPU mapping, multi-node deployment, and multi-directional permutations) are attributes of the physical execution environment. While these factors can undoubtedly affect the wall-clock latency of individual operations, they do not alter the logical ordering of the update events themselves under the stipulated FIFO and causal semantics. The causal ordering guarantee ensures that the sequence in which minibatches are processed forwards and backwards remains consistent with the logical program order. Therefore, since the logical schedule that dictates the number of forward passes before a backward pass is received remains unchanged, the bound on the mismatch count is invariant under these physical transformations.*

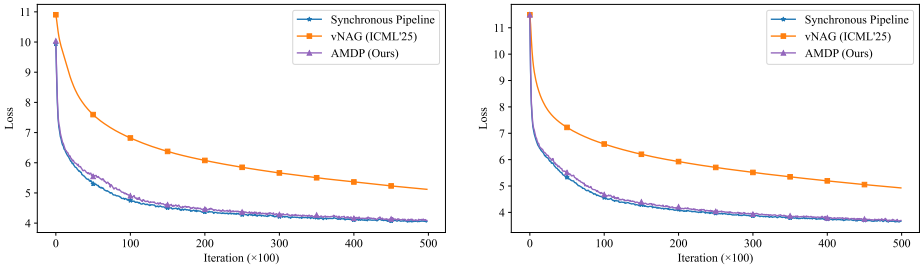
C Experiments

C.1 Comparison with recent work

Because the author-released implementation of vNAG (Ajanthan et al., 2025) does not run under our model configurations, we adapt both AMDP and synchronous pipeline approach (i.e., 1F1B implementation in Megatron-LM) to match the experimental setup used in vNAG. The configurations are detailed below:

Table 5: Comparison with vNAG. The best and second-best results are **bolded** and underlined. d denotes the pipeline depth and b indicates the number of samples processed per update.

Model	d	b	Throughput (ktokens/s) Sync. PP	vNAG	AMDP	Loss (over 50k iterations) Sync. PP	vNAG	AMDP
Model-1	8	8	31.0	47.1	**58.5**	**4.05**	5.12	4.09
Model-2	8	8	8.9	11.1	**18.2**	**3.67**	4.93	3.69



(a) Loss versus iterations (Model-1)

(b) Loss versus iterations (Model-2)

Figure 9: Training loss.

- Model-1. A GPT-style architecture with a context length of 512, an embedding dimension of 768, 12 attention heads, and 8 transformer layers (approximately 134M parameters), where each layer is treated as one pipeline stage. The minibatch size is set to 8, with a learning rate of 3e-4 and a weight decay of 0.01.
- Model-2. A larger GPT variant that preserves the total number of stages (8) but increases the context length to 1024 and the embedding dimension to 2688, with 24 attention heads (approximately 1B parameters). A learning rate of 1e-4 is applied to all methods for this model.

We train both models from scratch for 50k iterations on the OpenWebText dataset. All experiments are performed on the sever equipped with 8 A800 GPUs. Table 5 reports throughput as well as the final training loss. Figure 9 presents loss trajectories for the experiments. These results highlight two consistent trends: (1) AMDP delivers the highest throughput across both model scales, surpassing vNAG by 24.2% on Model-1 and by 64.0% on Model-2. (2) AMDP tracks the convergence behavior of synchronous baseline much more closely than vNAG, which shows a pronounced deviation. These findings reinforce the central advantage of AMDP, namely its structural one-step mismatch bound, which ensures both efficient utilization and stable optimization dynamics.

D USE OF LLMS

The authors use LLM (e.g., ChatGPT) solely as a general-purpose assistive tool for grammar and format refinement. LLM **does not** contribute to research ideation or experimental design. The authors take full responsibility for the content of this paper.