aerospace

MDPI

Article

Dynamic Load Balancing and Communication Optimization for Data-Parallel Training on Heterogeneous Clusters

Ruiming Wen [1], Wenjing Hou [2,*], Hongchen Wen [2] and Wenhong Tian [1]

[1] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
[2] School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
* Correspondence: hwj@uestc.edu.cn

Abstract

The growing adoption of artificial intelligence in space communications has imposed increasingly demanding efficiency requirements on deep learning training for tasks such as intelligent signal processing and resource allocation. However, performance disparities and workload fluctuations across heterogeneous nodes cause synchronization delays, while gradient aggregation and redundant parameter updates further increase training overhead. This paper proposes a data-parallel training method that combines dynamic load balancing with communication optimization. The Iteration-Level Dynamic Batch Adjustment algorithm (IDAT) estimates node processing rates online and dynamically assigns local batch sizes while maintaining a fixed global batch size, with a time-variation threshold, integer adjustment, and GPU memory constraints used to control overhead. The communication optimization algorithm (MRA) integrates Reduce-Scatter, sharded parameter updates, and All-Gather into a unified workflow to reduce communication startup and redundant update costs. Experiments using ResNet-50, AlexNet, and VGG-16 on a cluster of three servers and five NVIDIA V100-series GPUs show that IDAT achieves a 1.03×–1.04× speedup over the baseline, while the combined IDAT–MRA scheme achieves a 1.41×–1.82× speedup over the native PyTorch DDP 1.12.1 and Horovod 0.28.1 configurations, with an absolute accuracy difference of no more than 0.02 percentage points. These results demonstrate the effectiveness of the proposed framework in improving synchronous data-parallel training efficiency on the evaluated heterogeneous cluster and indicate its potential applicability to the training of intelligent models for space communication systems.

Keywords: distributed deep learning; data parallelism; heterogeneous computing; dynamic load balancing; communication optimization

Academic Editor: Lamei Zhang

Received: 23 September 2026
Revised: 21 October 2026
Accepted: 23 October 2026
Published: 3 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

In recent years, with the rapid development of low-Earth-orbit satellite constellations, 5G non-terrestrial networks (NTNs), and integrated space–air–ground networks, space communication systems have evolved toward large-scale networking, dynamic collaboration, and intelligent operation. Deep learning has been increasingly applied to channel estimation, signal recognition, interference detection, beam management, resource allocation, and satellite-state prediction, providing new tools for intelligent perception, analysis,

and decision-making in space communication systems [1–4]. However, the continuous growth of model sizes and training datasets makes it difficult for a single computing device to train deep models efficiently. Distributed training improves training efficiency by parallelizing data processing, gradient computation, and parameter updates across multiple computing nodes [5–7]. Among existing approaches, data parallelism maintains a complete model replica on each node, processes different subsets of the training data, and updates model parameters through gradient synchronization. Owing to its simple implementation, scalability, and compatibility with existing model architectures, data parallelism has been widely adopted in frameworks such as PyTorch DistributedDataParallel (DDP) and Horovod [8–10].

Deep learning workloads in aerospace systems exhibit diverse computational and communication characteristics. For example, satellite communication signal classification requires models to extract discriminative representations from complex-valued in-phase and quadrature (I/Q) samples, whereas synthetic aperture radar (SAR) image interpretation relies on convolutional networks for target detection and recognition. Training these models on heterogeneous computing clusters can introduce substantial synchronization delays because individual workers differ in processing capability, available memory, and runtime workload. Moreover, gradient synchronization and parameter updates may become significant performance bottlenecks as the model size and the number of participating workers increase. These characteristics motivate the investigation of joint computation-load balancing and communication optimization for distributed training in aerospace applications.

More specifically, the efficiency of synchronous data-parallel training is affected by heterogeneous GPU performance, CPU processing capability, memory capacity, data-loading efficiency, network bandwidth, and shared-resource contention. These differences can result in inconsistent processing rates across computing nodes [11–14]. Since each iteration must wait for all nodes to complete local computation before gradient aggregation, slow nodes prolong the overall training time and leave high-performance nodes idle. Resource-adaptive scheduling and workload allocation can alleviate this problem by assigning different workloads to nodes with different capabilities [15–17]. However, coarse-grained adjustment may respond slowly to short-term performance variations, whereas overly frequent adjustment introduces additional monitoring and scheduling overhead. Meanwhile, gradient aggregation and parameter updates remain on the critical path of each iteration. Although Ring-Allreduce can perform decentralized distributed reduction, a large number of small tensors may incur considerable communication startup overhead. In addition, when every node repeatedly performs a complete parameter update after full gradient aggregation, redundant computation and synchronization costs are introduced [18–20]. Therefore, jointly optimizing computational load allocation and the communication-update workflow is essential for improving synchronous data-parallel training in heterogeneous environments.

To address these issues, this paper investigates the joint optimization of computational load allocation and the communication-update workflow for synchronous data-parallel training in heterogeneous environments, with the goal of supporting efficient model training for AI-enabled space communications. First, we propose an Iteration-Level Dynamic Batch Adjustment algorithm (IDAT), which monitors the iteration time of each node online, estimates its processing rate, and dynamically assigns local batch sizes under a fixed global batch size so that the local computation times of different nodes remain closely aligned. To limit adjustment overhead, IDAT incorporates a time-variation threshold, integer batch-size allocation, and GPU-memory constraints, and redistributes samples only when changes in node performance satisfy the triggering condition. Second, we propose an MRA-based communication-update optimization algorithm. MRA fuses

https://doi.org/10.3390/aerospace13111021

gradients into a contiguous tensor and reorganizes Reduce-Scatter, sharded parameter updates, and All-Gather into a unified workflow. Once a node obtains its reduced gradient shard, it immediately updates the corresponding parameter shard before synchronizing the updated model parameters. This design reduces communication startup overhead for small tensors and avoids redundant full-model updates [21–24].

IDAT and MRA optimize synchronous data-parallel training from the computation and communication perspectives, respectively. IDAT reduces synchronization delays by adapting sample allocation across heterogeneous nodes, whereas MRA shortens the critical path of each iteration by reorganizing gradient aggregation and parameter updates. Under the requirements of a fixed global batch size, correct sample-weighted aggregation, and consistent optimizer states, neither method changes the model architecture or objective function. They can therefore be jointly applied to model training for a variety of intelligent space communication tasks. The main contributions of this paper are summarized as follows:

(1) We develop a node-level iteration-time model subject to global batch-size and GPU memory constraints and propose the IDAT algorithm. IDAT dynamically adjusts local batch sizes according to the processing rates observed during training and uses a time-variation threshold to control the adjustment frequency, thereby reducing inter-node computation-time disparities and synchronization overhead.

(2) We propose the MRA communication-update workflow, which integrates gradient fusion, Reduce-Scatter, sharded parameter updates, and All-Gather into a unified process. By allowing each node to update only a subset of the model parameters, MRA reduces both small-message communication startup overhead and redundant parameter-update costs.

(3) We conduct extensive experiments on heterogeneous GPU clusters using different model architectures and datasets to evaluate the effectiveness of IDAT, MRA, and their joint optimization. The evaluation considers training time, speedup, synchronization overhead, convergence accuracy, and time-to-accuracy. We further investigate the impact of computational heterogeneity and validate the applicability of the proposed framework to aerospace-related workloads, including automatic modulation recognition and SAR ship detection.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the system model and optimization objectives. Section 4 introduces IDAT, MRA, and their collaborative training workflow. Section 5 presents the experimental results, and Section 6 concludes the paper.

2. Related Work

2.1. Parallel Training and Communication Frameworks

In this section, we review distributed training frameworks from two perspectives: task partitioning and communication organization. As shown in Figure 1, data parallelism keeps a complete model replica on each node and splits the global batch across nodes; model parallelism partitions the model and deploys different parts on multiple devices; hybrid parallelism combines data, pipeline, and tensor parallelism to support the training of extremely large models [25].

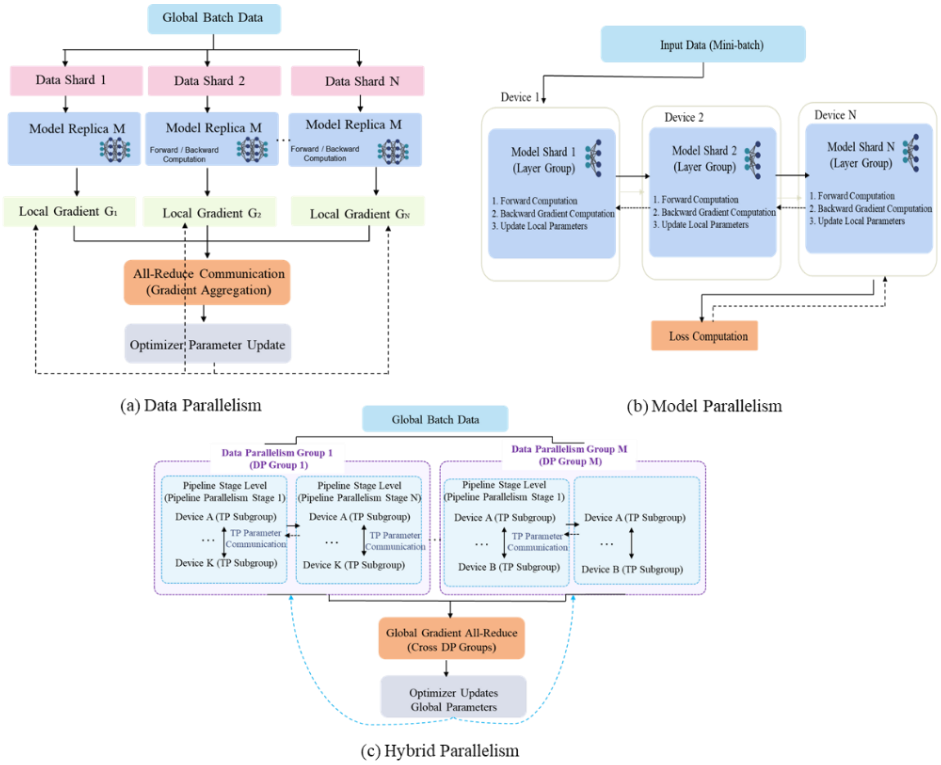


Figure 1. Parallel strategies for distributed training: (**a**) data parallelism, (**b**) model parallelism, and (**c**) hybrid parallelism.

In terms of communication organization, parameter-server architectures, as illustrated in Figure 2, perform gradient aggregation and parameter distribution through centralized servers, which may become communication bottlenecks as the number of workers increases. In contrast, Ring-Allreduce uses decentralized peer-to-peer communication for gradient synchronization [26]. As shown in Figure 3, Ring-Allreduce consists of two phases: Reduce-Scatter and All-Gather. Figure 3a illustrates gradient-shard transmission and accumulation, Figure 3b shows the propagation of reduced results around the ring, and Figure 3c summarizes the complete relationship between the two phases.

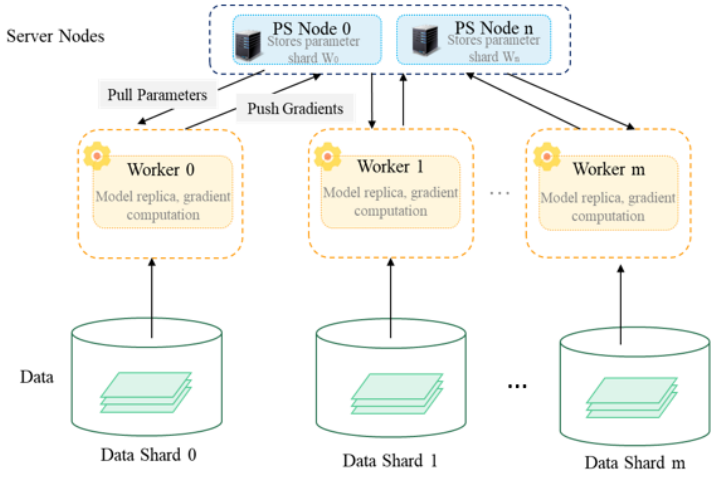


Figure 2. Parameter Server Architecture.

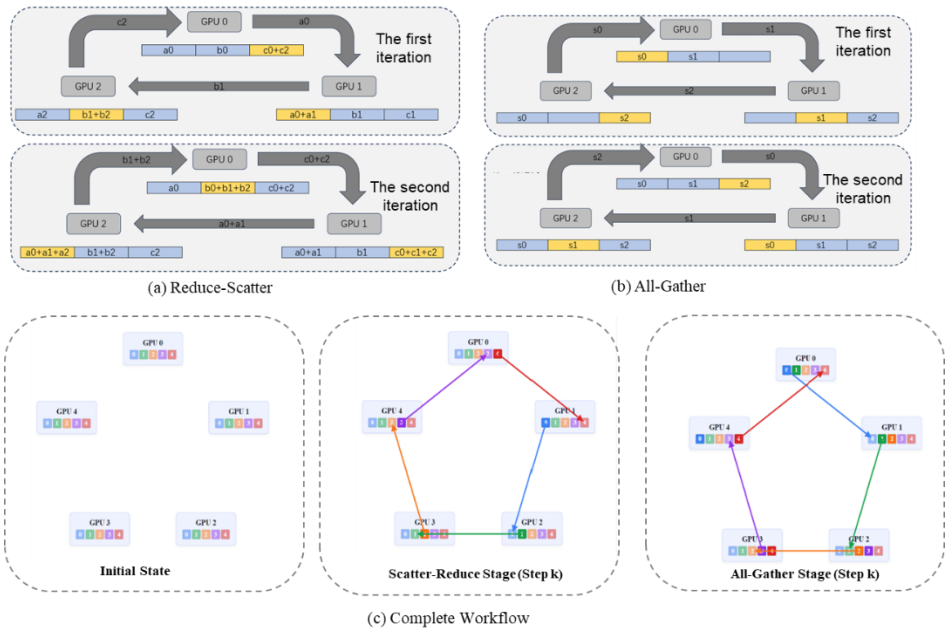


Figure 3. Ring-Allreduce Communication Workflow.

2.2. Synchronous Training and Load Balancing in Heterogeneous Environments

According to the parameter-update strategy, distributed training can be divided into synchronous and asynchronous training. Synchronous training requires all workers to update parameters based on the same model version, which generally provides stable convergence but makes each iteration limited by the slowest worker. Asynchronous training reduces synchronization waiting by allowing workers to update parameters independently, but it may introduce stale gradients [27,28]. Delay-compensation methods have been proposed to mitigate the impact of gradient staleness on convergence; however, they do not directly address load imbalance in synchronous training.

Existing studies on synchronous-training load imbalance mainly focus on resource scheduling, data partitioning, and local workload adjustment. Conventional data-parallel training assigns the same number of samples to each node. However, heterogeneous platforms often differ in computing capacity, GPU memory, data-loading speed, and runtime workload, leading to increased synchronization delays. Resource-adaptive frameworks such as ScaleFL and TapFinger adjust model workloads, task placement, or resource allocation according to the capabilities of heterogeneous clients and computing nodes [15,16]. TimelyFL and AntDT further consider time-varying resource availability and straggler behavior during distributed training [13,17]. LBB proposes load-balanced batching for heterogeneous GPU clusters but operates at epoch-level granularity and lacks explicit GPU memory constraints [11]. Pollux and Sia focus on cluster-level job scheduling and goodput optimization rather than fine-grained intra-iteration batch adaptation [10,13]. Nevertheless, these methods mainly focus on client participation, task scheduling, or coarse-grained workload adaptation, and may not respond promptly to short-term performance variations within an epoch.

To address this issue, this paper proposes the iteration-level dynamic batch adjustment algorithm IDAT. Different from existing epoch-level or scheduling-based methods, IDAT estimates node processing rates within each monitoring interval and dynamically assigns local batch sizes under global batch-size and GPU memory constraints. It uses a time-variation threshold to suppress unnecessary adjustments and weights local gradients according to the number of processed samples, thereby preserving the contribution of each sample to the global gradient and guaranteeing consistent optimization objectives.

2.3. Gradient Communication and Parameter Update Optimization

Gradient communication is a critical component of synchronous data-parallel training. Existing systems mainly reduce communication overhead through tensor fusion, communication bucketing, gradient compression, and computation–communication overlap. Horovod employs Tensor Fusion to merge small gradient tensors and reduce communication startup overhead [6]. PyTorch DDP groups gradients into communication buckets and overlaps Allreduce with backpropagation [8]. Recent methods, including Cupcake, SLAMB, SparDL, and L-GreCo, further optimize communication through compression scheduling, sparse communication, and adaptive gradient compression [20,22–24].

Other approaches improve memory efficiency through parameter, gradient, and optimizer-state sharding. ZeRO-1 partitions optimizer states, ZeRO-2 additionally partitions gradients, and ZeRO-3 further partitions model parameters. ZeRO++ reduces communication overhead through quantization and data remapping [21,29,30]. PyTorch FSDP similarly employs parameter and optimizer-state sharding to improve memory scalability, while Lancet coordinates computation and communication for Mixture-of-Experts training [30].

Compared with these methods, MRA focuses on communication–computation workflow optimization in heterogeneous data-parallel training. Unlike Horovod Tensor Fusion and DDP bucketing, which primarily optimize gradient communication, MRA integrates gradient fusion, Reduce-Scatter, sharded parameter updates, and All-Gather into a unified workflow. Each worker updates only its assigned parameter shard before synchronizing the updated parameters, avoiding redundant full-model updates. Although this mechanism shares similarities with ZeRO-2, MRA emphasizes workflow integration rather than introducing a new sharding principle. Unlike ZeRO-3 and fully sharded FSDP, MRA maintains complete model replicas during forward and backward propagation instead of targeting persistent parameter-memory reduction. Furthermore, MRA complements IDAT by reducing communication and update overhead, while IDAT mitigates computational imbalance, enabling joint optimization of heterogeneous distributed training.

3. System Model

3.1. Heterogeneous Cluster Training Scenario

We adopt DDP-style synchronous data-parallel training as the system model. Each process maintains a complete model replica, obtains a local batch through distributed data loading, and invokes the communication backend during backpropagation to synchronize gradients. The corresponding software and communication hierarchy is illustrated in Figure 4. IDAT adjusts data partitioning and local batch-size allocation at the user level, whereas MRA reorganizes gradient reduction and parameter updates within the DDP communication layer.

Consider a distributed training cluster consisting of N computing nodes (e.g., GPUs), with a user-defined global batch size B. In each iteration of data-parallel training, the global batch B is partitioned into N local batches. Let b_j denote the local batch size assigned to node j, subject to the following constraint:

$$\sum_{j=1}^{N} b_j = B \tag{1}$$

Each computing node j has an inherent, potentially time-varying computing capacity or processing rate, denoted by v_j, measured in samples per second. Accordingly, the time required for node j to complete its assigned computation, primarily including the forward and backward passes, is expressed as

$$T_{comp,j} = \frac{b_j}{v_j} \tag{2}$$

Under synchronous data-parallel training, the duration of a complete iteration T_{iter}, is determined by the computation time of the slowest node plus the subsequent gradient synchronization time T_{comm}. Neglecting the relatively small parameter-update overhead, T_{iter} can be formulated as

$$T_{iter} = \max_{j \in 1,..,N} \left(T_{comp,j}\right) + T_{comm} = \max_{j \in 1,..,N} \left(\frac{b_j}{v_j}\right) + T_{comm} \tag{3}$$

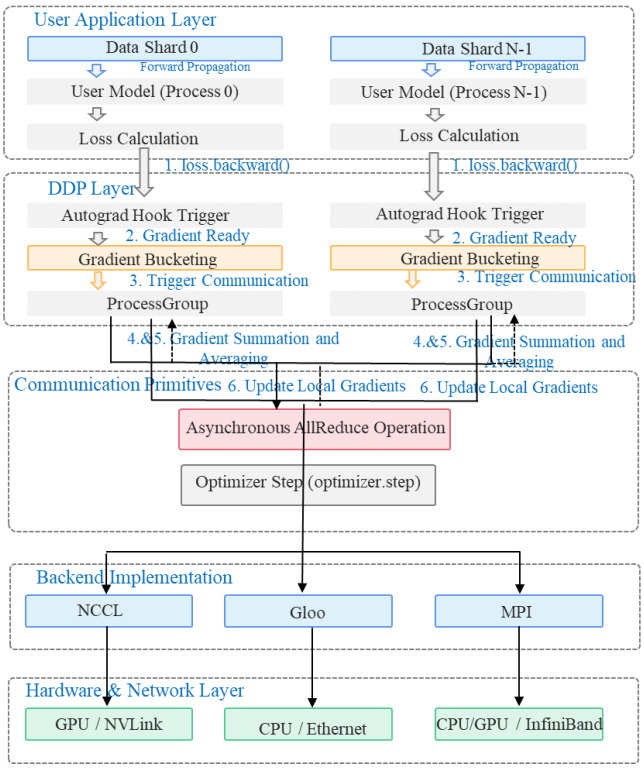


Figure 4. DDP Training and Communication Framework.

3.2. Problem Formulation and Optimization Objective

The primary optimization objective of this study is to minimize the duration of each training iteration T_{iter}, thereby reducing the overall model training time. Under a given hardware and network configuration, the communication time T_{comm} of an All-Reduce operation is relatively stable. Therefore, the optimization problem can be reduced to minimizing the maximum computation time among all nodes:

$$\begin{gathered} min\left(\max_{j \in 1,..,N} \left(\frac{b_j}{v_j}\right)\right) \\ \text{Conj. } \sum_{j=1}^{N} b_j = B, b_j \text{ is a positive integer.} \end{gathered} \tag{4}$$

It follows from this objective that $max(T_{comp,j})$ is minimized when the computation times of all nodes are equal, i.e.,

$$T_{comp,1} = T_{comp,2} = \cdots = T_{comp,N} \tag{5}$$

$$\frac{b_1}{v_1} = \frac{b_2}{v_2} = \cdots = \frac{b_N}{v_N} \tag{6}$$

Therefore, under ideal conditions, the batch size b_j assigned to each node should be proportional to its processing rate v_j. Specifically, the optimal batch allocation follows.

$$b_j^* = B \times \frac{v_j}{\sum_{k=1}^{N} v_k} \tag{7}$$

In practice, however, the actual processing rate v_j of node j is unknown and may vary dynamically because of external workloads, such as contention for shared resources. Consequently, the above optimization problem becomes an online estimation and dynamic decision-making problem: the processing rate v_j of each node must be estimated accurately in real time during training, and the local batch allocation b_j must then be adjusted dynamically so that it approaches the theoretical optimum b_j^*.

When local batch sizes vary across nodes, conventional unweighted gradient averaging would deviate from the standard global-batch gradient descent objective. To preserve consistency, IDAT adopts sample-weighted gradient aggregation.

Let $\nabla L_j(b_j)$ denote the average gradient over local batch b_j on node j. The globally aggregated gradient is computed as:

$$\nabla L_{global} = \frac{1}{B} \sum_{j=1}^{N} b_j \cdot \nabla L_j(b_j) \tag{8}$$

This formulation is mathematically equivalent to the standard gradient computed over the full global batch B, since each sample contributes equally to the global gradient regardless of which node it is assigned to. This weighting mechanism ensures that dynamic batch reallocation does not alter the optimization objective or convergence behavior of the model.

4. Joint Optimization Algorithm Design

4.1. IDAT Algorithm

Based on the preceding problem formulation, we develop an Iteration-Level Dynamic Batch Adjustment algorithm (IDAT). The central idea of IDAT is to refine the load-balancing granularity from the conventional epoch level to the iteration level and dynamically approach the optimal load allocation through online performance profiling. This design enables the algorithm to respond promptly to runtime performance fluctuations in the computing environment. The complete IDAT workflow comprises three stages: real-time performance monitoring, dynamic adjustment triggering, and adaptive batch reallocation. The detailed procedure is presented in Algorithm 1, which dynamically adjusts local batch sizes based on node processing capabilities while satisfying the global batch size and memory constraints.

4.1.1. Real-Time Performance Monitoring Mechanism

To estimate the unknown processing rate v_j, IDAT introduces a lightweight online profiler that continuously monitors node performance during training. The algorithm first logically divides each training epoch into M iteration intervals, referred to as monitoring intervals. Each interval contains several training iterations, and the complete dataset is correspondingly divided into M partitions.

During each monitoring interval, the system accurately records the time required by each node to complete its assigned computation, including data loading and the forward and backward passes. The empirical performance metric P_j^i is then used as a real-time estimate of the theoretical processing rate v_j:

$$P_j^i \approx v_j \quad where\, P_j^i = \frac{b_j^i}{T_{comp,j}^i} \tag{9}$$

where b_j^i denotes the batch size assigned to node j during the i monitoring interval, and $T_{comp,j}^i$ denotes the corresponding computation time.

4.1.2. Dynamic Adjustment Triggering Strategy

To avoid unnecessary and potentially costly adjustments caused by minor, transient performance fluctuations, IDAT employs a relative time-variation threshold as the trigger for load reallocation. Batch-size and dataset reallocation are initiated for the next monitoring interval only when the relative difference in total iteration time between two consecutive intervals exceeds the predefined threshold. The relative time-variation rate is calculated as:

$$\Delta_{rel} = \left| \frac{T_i - T_{i-1}}{T_{i-1}} \right| \tag{10}$$

Here, T_i denotes the total training time measured during the current monitoring interval, and T_{i-1} denotes that measured during the preceding interval. An adjustment is triggered if $\Delta_{rel} > \delta$, where δ is the user-specified time-variation threshold. This mechanism improves algorithm stability by triggering an adjustment only when a significant change in performance is detected.

4.1.3. Batch Reallocation

After dynamic adjustment is triggered, IDAT reallocates the local batch size for the next period according to the latest performance estimates of all nodes. Let B denote the global batch size and P_j^i denote the performance estimate of node j in period i. The theoretical batch size assigned to node j in the next period is calculated as

$$b_j^{(i+1)*} = B \times \frac{P_j^i}{\sum_{k=1}^{N} P_k^i} \tag{11}$$

This allocation assigns more samples to nodes with higher processing capacity and fewer samples to slower nodes, thereby reducing the difference in local computation time among nodes.

Since $b_j^{(i+1)*}$ may be non-integer, IDAT adopts an upward rounding and difference correction strategy:

Round each theoretical batch size upward to the nearest integer: $\hat{b}_j^{(i+1)} = \lceil b_j^{(i+1)*} \rceil$.

Compute the total number of excess samples: $d = \sum_{j=1}^{N} \hat{b}_j^{(i+1)} - B$.

Sort all nodes in ascending order of performance P_j^i, and remove one sample from each of the d slowest nodes.

The final local batch sizes satisfy:

$$\sum_{j=1}^{N} b_j^{(i+1)} = B, b_j^{(i+1)} \in \mathbb{Z}^+ \tag{12}$$

In addition, to avoid out-of-memory errors after batch reallocation, IDAT introduces a memory estimation and validation mechanism. This mechanism estimates the memory usage of each node according to the model size and the newly assigned batch size. If the estimated memory usage of a node exceeds its memory limit, its batch size is capped at the maximum feasible value, and the excess samples are reassigned to other nodes with available memory. In this way, IDAT preserves the global batch size while considering

both node processing capacity and memory constraints, thereby reducing the risk of OOM errors caused by dynamic batch adjustment.

Algorithm 1: Iteration-Level Dynamic Batch Adjustment Algorithm

Input: the deep learning model to be trained $Model$; the complete training dataset D; the total number of worker nodes N in the distributed cluster; the user-defined global batch size B; the total number of training epochs E; the monitoring and execution period A; and the maximum safe batch-size set of each node $b_{max,j}$.

Output: The batch size b_j^{i+1} assigned to each node in monitoring period $i+1$, and the optimized model $Model_{trained}$

Initialization phase;
Execute in parallel on each node j:
    Initialize the local model replica $Model_j \leftarrow Model$
    Initialize the local batch size $b_j \leftarrow \text{floor}(B/N)$
    Allocate the remaining samples to the first $B(\text{mod}N)$ nodes
    if $j \leq (B(\text{mod}N))$ **then** $b_j \leftarrow b_j + 1$
    Initialize the computation-time recorder $T_{comp,j} \leftarrow 0$
End parallel execution.
Main training loop
 for $e = 1$ **to** E **do**
     for $t = 1$ **to** ceil$(|D|/B)$ **do**
         Step 1: Parallel computation
         Execute in parallel on each node j:
             $start_time \leftarrow CurrentTime()$
             Load a mini-batch of size b_j from dataset D.
             Perform forward and backward propagation on the local data and obtain the local gradient$\nabla Model_j$
             $end_time \leftarrow CurrentTime()$
             $T_(comp, j) \leftarrow end_time - start_time$
          End parallel execution
          Step 2: Global synchronization and model update
              Perform global gradient synchronization so that all nodes obtain the global gradient $\nabla Model$
          Each node updates its local model replica $Model_j$ using the global gradient $\nabla Model$
         Step 3: Periodic dynamic adjustment
             if $t(\text{mod}A) == 0$ **then**
                Trigger-condition evaluation
                 Compute the performance indicator on each node in parallel $P_j \leftarrow b_j/T_{comp,j}$ and average iteration time of current interval
                $$T_i = \max_{j \in 1,..,N} (T_{comp,j}) + T_{comm}$$
                if $|(T_i - T_{i-1})/T_{i-1}| > \delta$ **then**
                    Batch reallocation
                    Call Algorithm ReallocateBatchSize:
                    $b_j \leftarrow \text{ReallocateBatchSize}(b_j, Tcomp, j, B, N, b_{max,j})$
                    Broadcast the new batch-allocation scheme b_j to all nodes
                end if
            end if
     end for
 end for
 return the model replica $Model_j$ from any node

4.2. MRA Algorithm

The IDAT algorithm addresses the synchronization waiting time T_{wait} caused by imbalanced computational loads across nodes, thereby improving cluster resource utilization in dynamic heterogeneous environments. However, the gradient communication time T_{comm} and parameter update time still account for a non-negligible portion of the total iteration time T_{iter}. This issue becomes more pronounced in modern deep learning models with increasingly large parameter scales. Therefore, we propose Merge-Ring-Allreduce (MRA), a training algorithm that deeply integrates gradient synchronization communication with parameter update computation, to further shorten the training iteration cycle.

4.2.1. Gradient Fusion Mechanism

Considering the gradient synchronization process in data-parallel training, Figure 5 illustrates the Ring-Allreduce communication workflow on three GPUs. Each GPU initially holds a different tensor and divides it into three data blocks. The Scatter-Reduce phase is then performed, allowing each GPU to obtain one reduced data block. After that, the All-Gather phase propagates the reduced results among all GPUs, so that every GPU finally obtains the fully synchronized tensor.

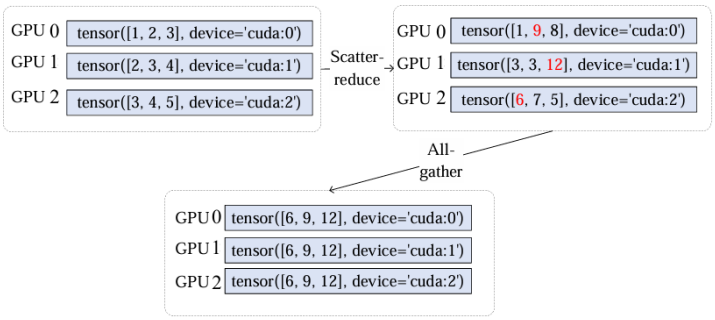


Figure 5. Example of Ring-Allreduce Gradient Synchronization Across Three GPUs.

Assume that the model contains L parameter layers that need to be synchronized and the cluster consists of N computing nodes. In the standard synchronization process, after backpropagation is completed, the total time for gradient synchronization and parameter update in one iteration is mainly composed of two parts: the communication time of L independent Ring-Allreduce operations and the time required for each node to update the full model parameters:

$$T_{standard} = T_{comm_{grad}} + T_{update_{var}} \tag{13}$$

The gradient communication time $T_{comm_{grad}}$ can be further expressed as the sum of the startup overhead and data transfer time of each communication operation:

$$T_{comm_{grad}} = \sum_{i=1}^{L} \left(t_{startup} + t_{transfer}(s_i) \right) \tag{14}$$

Here, the accumulated startup overhead $\sum_{i=1}^{L} t_{startup} = L \times t_{startup}$ increases linearly with the number of layers, resulting in considerable fixed communication overhead. Moreover, $T_{update_{var}}$ denotes the time for each node to independently update all model parameters, without exploiting the structure of gradient synchronization and parameter updates to reduce redundant computation.

Therefore, the goal of MRA is to design a reorganized training workflow whose total time T_{MRA} is significantly lower than $T_{standard}$. This is achieved through gradient fusion and workflow reorganization.

The MRA algorithm first introduces a gradient fusion mechanism. After backpropagation is completed, it does not initiate communication for the gradients of each layer separately. Instead, it flattens and concatenates the gradients of all trainable parameters into a single contiguous tensor, and then performs one Ring-Allreduce operation on the fused tensor. This design merges numerous small communication operations into one large-scale communication operation, reducing startup overhead and improving network bandwidth utilization.

4.2.2. Communication–Computation Fusion Workflow

MRA integrates gradient fusion, Reduce-Scatter, sharded parameter updates, and All-Gather into a unified communication–computation workflow while maintaining full-model replicas for synchronous data-parallel training. Although these operations are closely related to ZeRO-based optimization, MRA focuses on reducing communication overhead and redundant parameter updates.

Specifically, after backpropagation, local gradients are flattened and fused into a contiguous tensor. Reduce-Scatter aggregates and partitions the gradients across workers. Each worker then updates only its assigned parameter shard using the corresponding gradient shard and locally maintained optimizer states. For SGD with momentum, each shard owner maintains its corresponding momentum buffers, avoiding redundant optimizer-state updates and replication. Finally, All-Gather synchronizes the updated parameter shards to reconstruct identical full-model replicas before the next forward pass.

As illustrated in Figure 6, unlike conventional data-parallel training, where each worker independently updates the complete model after gradient synchronization, MRA performs partitioned updates between Reduce-Scatter and All-Gather. This workflow reduces redundant update computation while ensuring parameter consistency across workers.

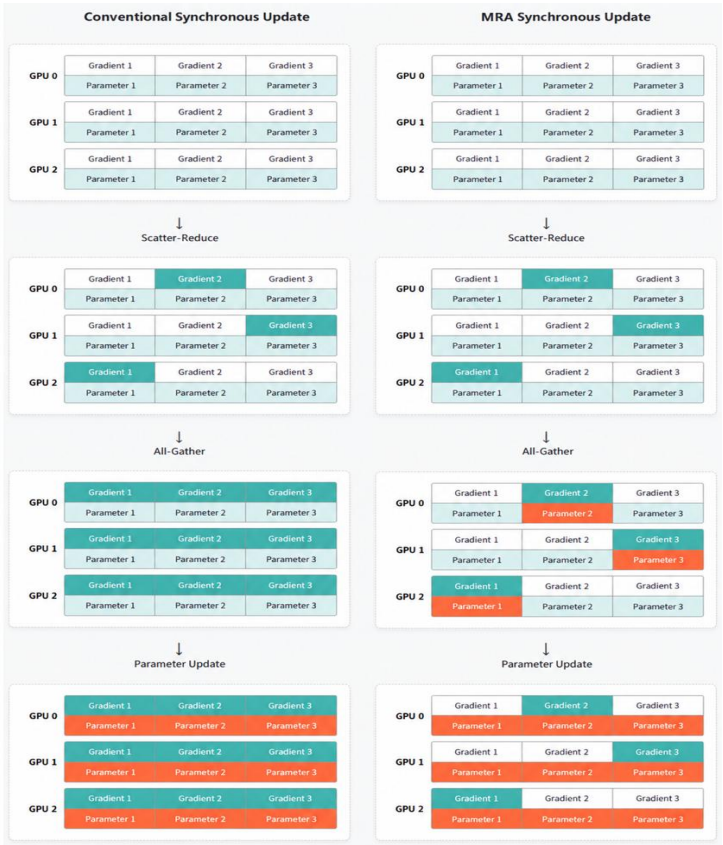


Figure 6. Comparison of Conventional and MRA Synchronous Update Workflows.

4.2.3. Algorithm Execution Workflow

The complete execution workflow of MRA is illustrated in Algorithm 2, which integrates gradient aggregation, sharded parameter updates, and parameter synchronization to reduce communication overhead and redundant computation. In each training iteration, each worker obtains its globally aggregated gradient shard and updates the corresponding parameter shard using its locally maintained optimizer states. The updated parameter shards are then synchronized through All-Gather to reconstruct identical full-model replicas across all workers. Since each shard has a unique update owner and the next forward pass begins only after synchronization, parameter consistency is guaranteed. Combined with IDAT's sample-weighted gradient aggregation, this mechanism preserves the global-batch optimization objective despite unequal local batch sizes.

Algorithm 2: Merge-Ring-Allreduce Algorithm

Initialization: Number of training epochs E, number of iterations per epoch T, and the deep learning model to be trained M.

for $e = 1$ to E do
  for $i = 1$ to T do
  Stage 1: Local computation
  $batch \leftarrow DataLoader.next()$
  $loss \leftarrow ForwardPropagation(M, batch)$
  $BackwardPropagation(loss)$
  Stage 2: Communication-computation fusion
  $G \leftarrow FuseGradients(M.gradients)$
  $G_aggregated \leftarrow ScatteReduce(G)$
  $UpdateParametersLocally(M.params, G_aggregated)$
  $M.params \leftarrow All - Gather(M.params)$
  end for
end for

By integrating gradient fusion and sharded parameter updates, MRA reduces communication startup overhead and redundant full-model update computation. Unlike conventional replicated updates, each worker updates only its assigned parameter partition, reducing its update workload while maintaining synchronous training semantics. However, MRA retains complete model replicas during forward and backward propagation and does not inherently reduce parameter-memory requirements. Its primary benefit therefore lies in communication and update efficiency rather than memory scalability, and the overall speedup remains dependent on computation, communication, and synchronization overhead.

5. Experimental Results and Discussion

5.1. Experimental Setup

5.1.1. Software and Hardware Environment

To sufficiently simulate the heterogeneity of computing resources commonly found in real-world environments, which is one of the core challenges addressed by the proposed IDAT algorithm, this experiment constructs a distributed GPU cluster with mixed hardware configurations. The cluster consists of three physical servers and contains a total of five NVIDIA Tesla V100 GPUs (Santa Clara, CA, USA).

To ensure the reproducibility of the experimental results and the fairness of comparison, all servers are deployed with a unified software stack. The detailed hardware and software configurations of the experimental platform are shown in Table 1.

Table 1. Hardware and software configurations of the experimental platform.

Configuration	Device 1	Device 2	Device 3
CPU model	Intel(R) Core(TM) i7-9800 @3.80 GHz	Intel(R) Core(TM) i9-10850K@ 3.60 GHz	Intel(R) Core(TM) i9-10850K@ 3.60 GHz
Memory capacity	94 GB	62 GB	62 GB
GPU model	Tesla V100-PCIE 32G, Tesla V100-SXM2, 16G	Tesla V100-DGXS 32G, Tesla V100-DGXS 32G	Tesla V100-SXM3 32G
CUDA version	11.2	11.2	11.2
cuDNN version	8.6	8.6	8.6
Operating system	Ubuntu 18.04	Ubuntu 18.04	Ubuntu 18.04
Python version	3.8	3.8	3.8
PyTorch version	1.12.1	1.12.1	1.12.1

5.1.2. Evaluation Models and Datasets

To comprehensively evaluate the performance of the proposed algorithm under different computation and communication workloads, this study selects three representative convolutional neural network models in the field of computer vision: ResNet-50, AlexNet, and VGG-16. These models differ significantly in terms of parameter size and floating-point operations, making them suitable for evaluating the adaptability of the proposed optimization algorithm to compute-intensive, communication-intensive, and mixed workloads.

The key characteristics of the evaluated models are summarized in Table 2. AlexNet and VGG-16 contain a large number of parameters and therefore impose higher bandwidth demands during training, making them representative communication-intensive models. ResNet-50 has a moderate parameter size, but its computational cost is significantly higher than that of AlexNet, making it more compute-intensive. VGG-16 has both high parameter size and high computational cost, placing greater demands on the distributed training system. Evaluating these models with different characteristics enables a more comprehensive assessment of the optimization performance and generalization ability of the proposed algorithm.

Table 2. Basic Information of the Experimental Models.

Network Model	Trainable Parameters	Computational MACs
ResNet-50	25.6 M	4.1 G
AlexNet	61.1 M	0.77 G
Vgg-16	138.4 M	15.6 G

To verify the stability and effectiveness of the algorithm under different data scales, two widely used image classification datasets, CIFAR-10 and Mini-ImageNet, are used in the experiments. CIFAR-10 contains 60,000 color images of size 32 × 32 from 10 classes, with a total size of approximately 177 MB. It is suitable for algorithm prototype validation and ablation studies. Mini-ImageNet is a subset of ImageNet and contains 60,000 images from 100 classes, with a data size of approximately 3 GB. It places higher demands on computing resources, I/O performance, and distributed communication efficiency, and is therefore used to evaluate the practical effectiveness of the proposed method on larger-scale tasks.

5.1.3. Baselines and Evaluation Metrics

To evaluate the effectiveness of the proposed framework, we select three representative methods as experimental baselines: native PyTorch DistributedDataParallel (DDP),

Horovod, and Dynamic Load Balancing (DLB). PyTorch DDP and Horovod represent conventional synchronous data-parallel training frameworks employing gradient bucketing and tensor fusion, respectively. DLB serves as a direct load-balancing baseline for evaluating the effectiveness of IDAT in mitigating computational imbalance across heterogeneous workers. To distinguish the individual and combined contributions of IDAT and MRA, four configurations are evaluated within the PyTorch DDP and Horovod frameworks: Native, IDAT-only, MRA-only, and IDAT+MRA. All configurations use consistent hardware settings, global batch sizes, optimizer settings, and training hyperparameters.

To assess statistical reliability, each configuration included in the repeated-run evaluation is independently executed five times under consistent experimental conditions. Training time and convergence accuracy are reported as mean ± standard deviation, and 95% confidence intervals are calculated for the mean training time. For the IDAT comparisons, statistical significance is assessed using a two-sided paired *t*-test against the corresponding native PyTorch DDP or Horovod baseline, with $p < 0.05$ indicating statistical significance. Paired runs use identical random seeds and controlled disturbance settings to reduce experimental variability. These evaluations provide a consistent basis for assessing the reproducibility and performance improvements of the proposed methods.

This study adopts metrics from three perspectives: overall efficiency, fine-grained runtime behavior, and final training effectiveness. First, system throughput measures the number of training samples processed per unit time and is defined as

$$Throught = \frac{batch_size}{time} \tag{15}$$

where $batch_size$ denotes the training batch size, and $time$ denotes the corresponding elapsed time. This metric is mainly used during parameter tuning to determine the global batch size that can fully utilize the cluster hardware resources.

Second, speedup measures the efficiency improvement of the optimized method over the baseline method and is defined as

$$S = \frac{T_{baseline}}{T_{optimized}} \tag{16}$$

where $T_{baseline}$ denotes the training time of the baseline method, and $T_{optimized}$denotes the training time after applying the proposed optimization algorithm. When $S > 1$, the optimized method improves training efficiency.

In addition, single-iteration time profiling is used to analyze the time distribution across different stages of the algorithm, including computation time, communication time, and synchronization waiting time on each GPU node. This metric reveals how IDAT reduces synchronization waiting and how MRA lowers communication and parameter update overhead.

Finally, time-to-accuracy (TTA) and model convergence accuracy are used to evaluate the practical training effectiveness of the proposed method. TTA is measured based on the actual cumulative wall-clock training time, rather than the number of training iterations. Specifically, for a predefined target accuracy A_{target}, TTA is defined as

$$TTA(A_{\text{target}}) = min\{t \mid A_{\text{val}}(t) \geq A_{\text{target}}\}, \tag{17}$$

where $A_{\text{val}}(t)$ denotes the validation accuracy measured at cumulative wall-clock time t. For the CIFAR-10 experiments, the target accuracies are consistently set to 85% for ResNet-50, 82% for AlexNet, and 85% for VGG-16. The same target accuracy is applied to all configurations of a given model to ensure a fair comparison.

5.2. Hyperparameter Sensitivity Analysis

This section examines the effects of the global batch size and the time-variation threshold on training performance. First, using ResNet-50 and CIFAR-10 as examples, we evaluate the system throughput under different global batch sizes. As shown in Figure 7, the throughput increases with the batch size and reaches its peak at a global batch size of 512.

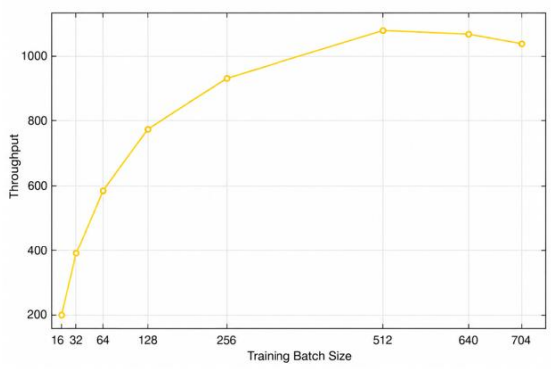


Figure 7. Cluster throughput under different training batch sizes.

We then investigate the effect of the time-variation threshold on the performance of IDAT. Under a fixed experimental setup, the threshold is varied in increments of 1 percentage point, and the corresponding training speedup is recorded. As shown in Figure 8, the speedup steadily increases as the threshold rises from 0% to 10%, indicating that a properly selected threshold can reduce frequent batch adjustments caused by transient performance fluctuations. When the threshold exceeds 10%, the speedup gradually declines, and no further improvement is observed beyond 22%. These results show that a threshold of 10% provides a good balance between responsiveness to changes in node performance and the overhead of batch adjustment.

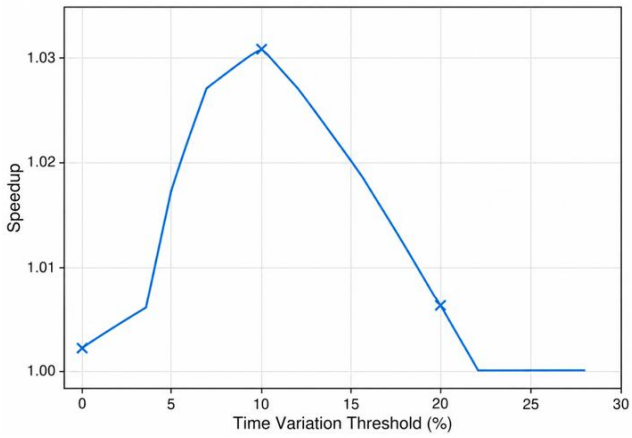


Figure 8. Relationship between Time Volatility Threshold and Speedup Ratio of IDAT Algorithm.

5.3. IDAT Performance Evaluation

After completing the experimental setup and hyperparameter tuning, this section evaluates the performance of the proposed intra-iteration dynamic batch adjustment (IDAT) algorithm under heterogeneous and dynamically varying workloads. To emulate resource contention in shared computing environments, small random sleep delays are introduced at different nodes during training iterations. The performance of IDAT is compared with native PyTorch DDP, Horovod, and the existing DLB method.

5.3.1. Performance Under Different Models and Frameworks

Experiments are conducted on CIFAR-10 using ResNet-50, AlexNet, and VGG-16, which exhibit different computational and communication characteristics. Figures 9–14 present the experimental results for these three models. Specifically, Figures 9, 11 and 13 present the average iteration time in the 50th epoch, where the bars denote the time consumed by each training stage and labels ending in "IDAT" indicate the corresponding optimized framework.

As shown in these figures, native synchronous training suffers from considerable waiting time before gradient synchronization because of differences in node processing speeds. After IDAT is applied, the local batch sizes are adjusted according to the observed processing rates, allowing faster nodes to undertake more samples. Consequently, the computation times across nodes become more balanced, and the synchronization waiting time is substantially reduced.

Figures 10, 12 and 14 further compare the average iteration time during training. Across all three models, PyTorch-IDAT and Horovod-IDAT achieve lower iteration times than the corresponding native frameworks and the DLB-based methods. Moreover, the IDAT curves fluctuate less than the DLB curves, indicating that intra-iteration adjustment responds more promptly to short-term workload changes than epoch-level adjustment and provides more stable performance gains.

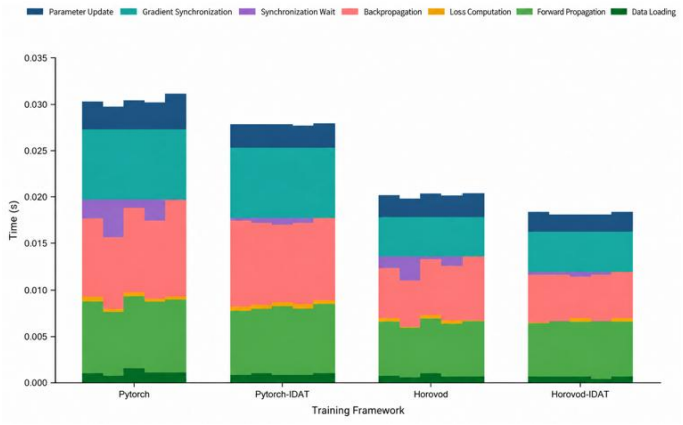


Figure 9. Analysis Diagram of Per-Iteration Time Consumption for ResNet-50 (CIFAR-10 Dataset).

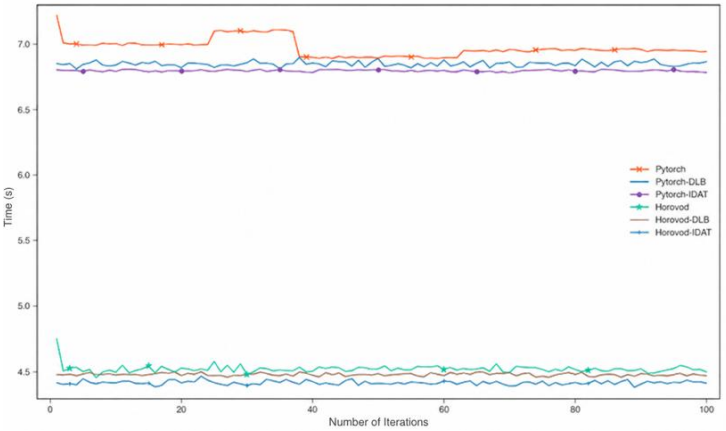


Figure 10. Comparison of Average Time per Iteration for ResNet-50 (CIFAR-10).

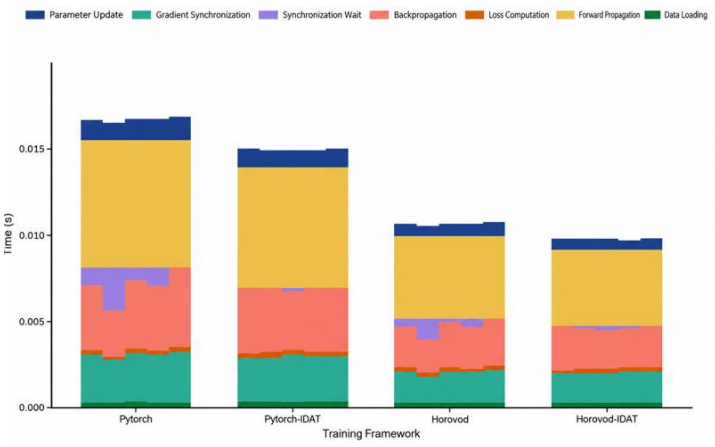


Figure 11. Analysis Diagram of Per-Iteration Time Consumption for AlexNet (CIFAR-10 Dataset).

A comparison across models shows that AlexNet and VGG-16 have a higher proportion of gradient synchronization time than ResNet-50. This is mainly because their larger numbers of trainable parameters produce larger gradients and increase the communication volume. These results confirm that IDAT effectively reduces synchronization overhead across different model architectures and training frameworks.

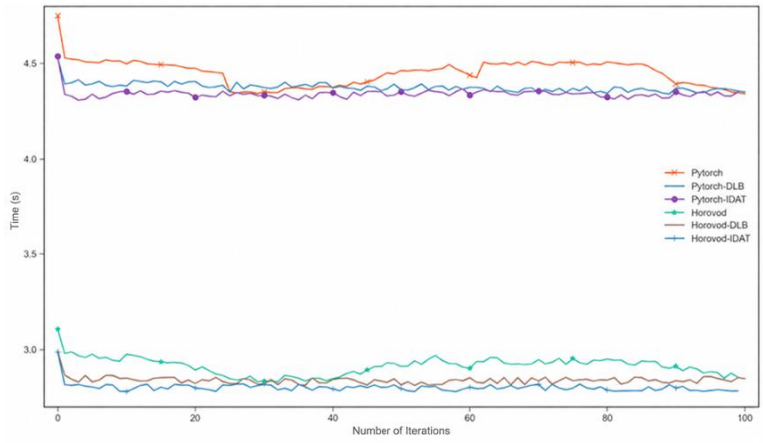


Figure 12. Comparison of Average Time per Iteration for AlexNet (CIFAR-10).

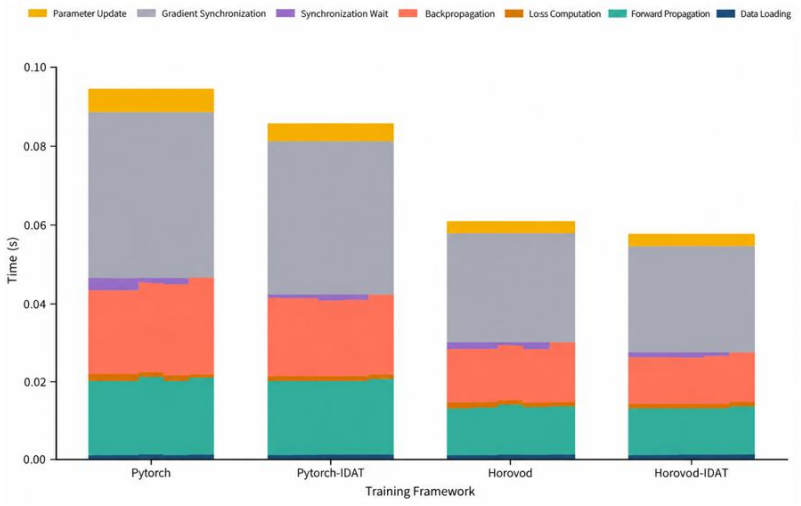


Figure 13. Analysis Diagram of Per-Iteration Time Consumption for VGG-16 (CIFAR-10 Dataset).

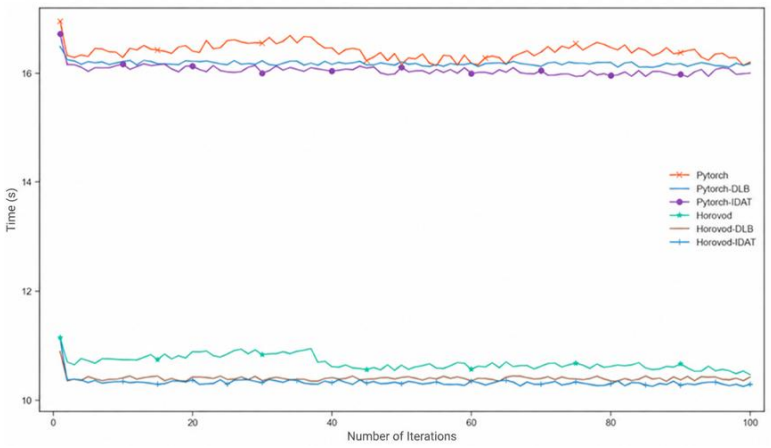


Figure 14. Comparison of Average Time per Iteration for VGG-16 (CIFAR-10).

5.3.2. Performance Quantification and Generalization Ability on Datasets of Different Scales

Tables 3 and 4 quantitatively evaluate the performance of IDAT on CIFAR-10 and Mini-ImageNet, respectively. To assess whether the observed performance improvements are robust to runtime variation, each experimental configuration was independently repeated five times. Training time and convergence accuracy are reported as the mean ± standard deviation, and 95% confidence intervals are additionally calculated for the mean training time. An asterisk (*) denotes a statistically significant reduction in training time relative to the corresponding native framework according to a two-sided paired *t*-test ($p < 0.05$).

As shown in Table 3, IDAT consistently reduces the training time of ResNet-50, AlexNet, and VGG-16 under both PyTorch DDP and Horovod. Compared with the corresponding native frameworks, IDAT achieves mean speedups of approximately 1.03×–1.04× and generally outperforms DLB. Although the absolute improvement is relatively modest, the reductions in training time are consistent across the five independent runs and statistically significant under the evaluated configurations, indicating that the observed gains are not dominated by normal run-to-run variation. Meanwhile, the convergence accuracy remains nearly unchanged after IDAT is applied, demonstrating that dynamic workload adjustment improves training efficiency without materially affecting model convergence.

Table 3. Training Time (s) and Accuracy (%) on CIFAR-10 over Five Independent Runs (Mean ± Standard Deviation).

Method	ResNet-50 Time (s)	Accuracy (%)	AlexNet Time (s)	Accuracy (%)	VGG-16 Time (s)	Accuracy (%)
PyTorch	699.1 ± 4.2	85.36 ± 0.04	445.7 ± 3.1	82.67 ± 0.05	1654.1 ± 12.3	86.03 ± 0.03
PyTorch-DLB	686.6 ± 5.7	85.35 ± 0.05	439.8 ± 3.8	82.66 ± 0.04	1627.7 ± 15.1	86.02 ± 0.04
PyTorch-IDAT	**673.9 ± 3.5 ***	85.35 ± 0.04	**433.2 ± 2.9 ***	82.67 ± 0.03	**1596.2 ± 10.8 ***	86.03 ± 0.03
Horovod	454.4 ± 3.3	85.36 ± 0.03	289.7 ± 2.4	82.66 ± 0.04	1075.2 ± 9.6	86.03 ± 0.02
Horovod-DLB	447.2 ± 4.1	85.33 ± 0.04	286.3 ± 2.8	82.63 ± 0.05	1059.2 ± 11.2	86.03 ± 0.03
Horovod-IDAT	**440.3 ± 2.8 ***	85.36 ± 0.03	**282.2 ± 2.1 ***	82.65 ± 0.04	**1041.9 ± 8.7 ***	86.02 ± 0.03

Note: Bold values indicate the shortest training time for each model within the corresponding framework. * indicates a statistically significant difference compared with the corresponding native baseline (paired *t*-test, $p < 0.05$).

Table 4 presents the training performance of different methods on the Mini-ImageNet dataset. Compared with the corresponding native PyTorch DDP and Horovod baselines, IDAT consistently reduces training time across ResNet-50, AlexNet, and VGG-16, achieving speedups of approximately 1.03×–1.04×. The results indicate that iteration-level dynamic batch adjustment can effectively mitigate synchronization delays caused by heterogeneous worker performance. Moreover, the relatively small variations in convergence accuracy suggest that IDAT maintains comparable model performance while improving training efficiency. Together with the CIFAR-10 results in Table 3, these findings demonstrate consistent training-time improvements across different model architectures and dataset scales. Although the gains under the evaluated cluster are relatively modest, they are consistent with the moderate degree of computational heterogeneity, which limits the potential reduction in synchronization waiting time.

Table 4. Training Time (min) and Accuracy (%) on Mini-ImageNet over Five Independent Runs (Mean ± Standard Deviation).

Method	ResNet-50 Time (min)	Accuracy (%)	AlexNet Time (min)	Accuracy (%)	VGG-16 Time (min)	Accuracy (%)
PyTorch	238.6 ± 2.1	76.32 ± 0.06	153.6 ± 1.4	63.36 ± 0.05	429.6 ± 4.8	83.56 ± 0.04
PyTorch-DLB	233.6 ± 2.5	76.30 ± 0.05	151.3 ± 1.7	63.36 ± 0.04	421.5 ± 5.3	83.53 ± 0.05
PyTorch-IDAT	**229.3 ± 1.9** *	76.31 ± 0.04	**148.9 ± 1.3** *	63.36 ± 0.04	**413.7 ± 4.2** *	83.55 ± 0.04
Horovod	149.2 ± 1.6	76.30 ± 0.05	91.6 ± 1.1	63.35 ± 0.04	256.2 ± 3.7	83.53 ± 0.04
Horovod-DLB	146.5 ± 1.8	76.31 ± 0.04	90.4 ± 1.2	63.33 ± 0.05	251.8 ± 3.9	83.53 ± 0.04
Horovod-IDAT	**144.3 ± 1.5** *	76.32 ± 0.03	**88.9 ± 1.0** *	63.36 ± 0.03	**248.0 ± 3.3** *	83.53 ± 0.03

Note: Bold values indicate the shortest training time for each model within the corresponding framework. * indicates a statistically significant difference compared with the corresponding native baseline (paired t-test, $p < 0.05$).

The results in Tables 3 and 4 show that IDAT consistently achieves approximately 1.03×–1.04× speedup over the native baselines. Although the improvements are relatively modest, the five-run statistical analysis provides evidence of their reproducibility under the evaluated cluster configuration. The limited gains primarily reflect the moderate effective heterogeneity of the physical cluster, where synchronization imbalance is relatively small. As further investigated in Section 5.3.3, the benefit of IDAT increases with greater inter-worker performance disparities.

5.3.3. Performance Under Increasing Degrees of Heterogeneity

To investigate the sensitivity of IDAT to worker-performance disparities, controlled delays are introduced into the second GPU of Node 1, with the injected sleep time increasing in 2 ms increments. This experiment simulates straggler-induced synchronization delays while keeping the remaining training configurations unchanged.

As shown in Figure 15, IDAT achieves greater speedups as the injected delay increases. Under moderate heterogeneity, the synchronization imbalance is limited, resulting in approximately 3–4% improvement. With increasing delays, IDAT reallocates samples from slower to faster workers, reducing synchronization waiting. These results support the effectiveness of dynamic batch adjustment under controlled computational imbalance. Nevertheless, delay injection does not capture all characteristics of real hardware heterogeneity, such as memory bandwidth differences, GPU architectural variations, and network contention. Further validation on larger heterogeneous clusters remains necessary.

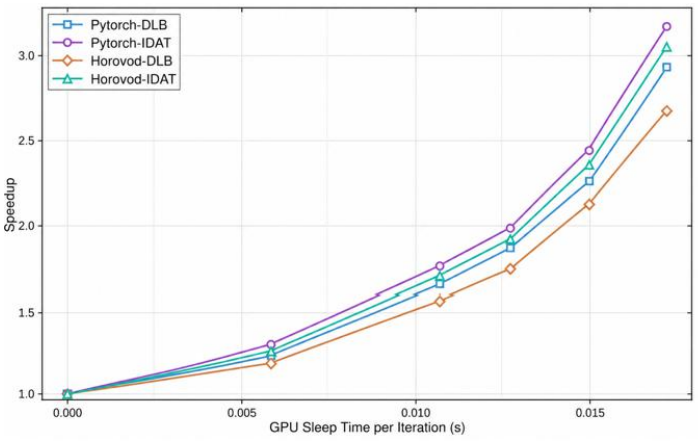


Figure 15. Variation in Speedup Ratio for IDAT Algorithm under Extreme Heterogeneous Environments.

5.4. MRA Communication Optimization and Joint Performance Evaluation

5.4.1. Acceleration Performance Across Different Models and Frameworks

After evaluating the load-balancing effectiveness of IDAT in Section 5.3, this section focuses on the communication optimization provided by MRA and its complementary effect when combined with IDAT. To avoid ambiguity between the contributions of the two components, the experiments are organized into two comparison levels. First, an MRA-only comparison is performed between the native PyTorch DDP/Horovod configurations and their corresponding MRA-enabled configurations, thereby evaluating the performance gain attributable specifically to communication and parameter-update optimization. Second, a joint incremental comparison is performed between the IDAT-only configurations and the corresponding IDAT+MRA configurations, which quantifies the additional benefit provided by MRA after computational load imbalance has been mitigated by IDAT.

Accordingly, the experimental configurations are explicitly denoted as Native, IDAT, MRA, and IDAT+MRA for each training framework. The Native configuration uses the original framework without either proposed optimization; IDAT enables only dynamic batch adjustment; MRA enables only the communication-update optimization; and IDAT+MRA enables both components. This configuration separation ensures that the individual and combined contributions of IDAT and MRA can be evaluated independently and avoids ambiguity in interpreting the reported training times. The corresponding experimental results are presented in Figures 16–21.

As shown in Figures 16, 18 and 20, the comparison between the Native and MRA configurations demonstrates the direct effect of MRA on the communication-update workflow. MRA substantially reduces gradient synchronization and parameter-update overhead. This improvement is mainly attributed to two mechanisms. First, gradient fusion combines multiple small communication operations into a larger collective operation, thereby reducing repeated communication startup overhead. Second, the sharded update mechanism distributes parameter-update computation across nodes and eliminates redundant full-model updates on each worker.

In addition, the comparison between IDAT and IDAT+MRA shows that MRA continues to provide performance gains after IDAT has reduced inter-node computational imbalance. This observation confirms that IDAT and MRA address different components of the iteration critical path: IDAT primarily reduces synchronization waiting caused by heterogeneous computation, whereas MRA reduces communication and parameter-update overhead.

To directly evaluate the practical end-to-end training efficiency of MRA, Figures 17, 19 and 21 plot validation accuracy against the actual cumulative wall-clock training time, rather than the number of training iterations. The target accuracies are consistently defined as 85% for ResNet-50, 82% for AlexNet, and 85% for VGG-16, and the same threshold is applied to all configurations of each model. TTA is measured as the first wall-clock time at which the validation accuracy reaches or exceeds the corresponding target threshold.

The quantitative results of MRA and its joint optimization with IDAT are summarized in Tables 5–7, covering training efficiency, convergence accuracy, and time-to-accuracy performance. The quantitative TTA results are summarized in Table 7. For ResNet-50, MRA reduces TTA from 582 s to 415 s under PyTorch DDP and from 381 s to 279 s under Horovod, corresponding to reductions of 28.7% and 26.8%, respectively. For AlexNet, TTA decreases from 376 s to 237 s under PyTorch DDP and from 247 s to 161 s under Horovod, yielding reductions of 37.0% and 34.8%, respectively. For VGG-16, MRA reduces TTA from 1392 s to 901 s under PyTorch DDP and from 912 s to 621 s under Horovod, corresponding to reductions of 35.3% and 31.9%, respectively. Overall, MRA reduces time-to-accuracy by 26.8–37.0% across the evaluated configurations. Meanwhile, the final convergence accuracy remains essentially unchanged, as shown in Table 5. These results indicate that MRA does not obtain training acceleration by relaxing the convergence target. Instead, by reducing communication startup overhead and redundant parameter-update computation, MRA enables the same optimization progress to be achieved in less wall-clock time.

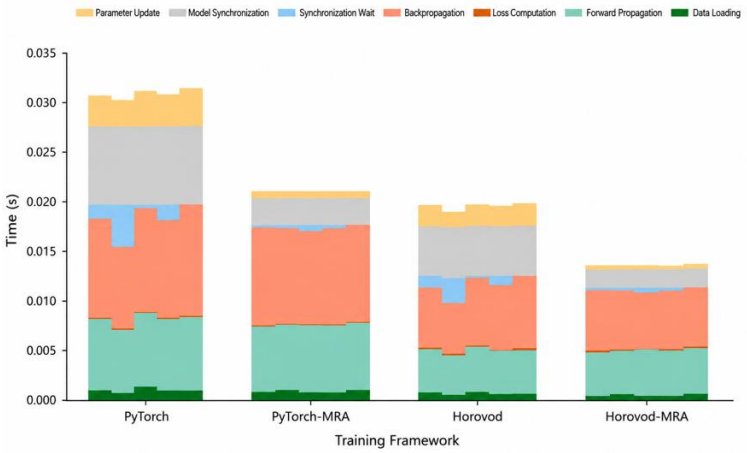


Figure 16. Breakdown of Single-Iteration Time for ResNet-50 (CIFAR-10, MRA).

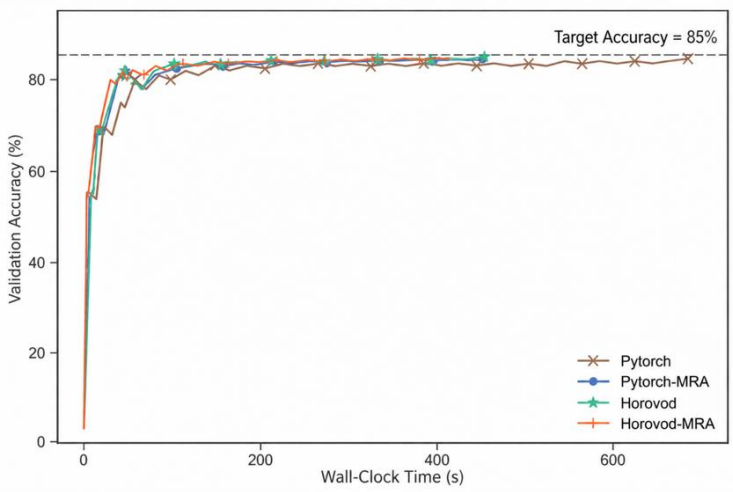


Figure 17. Validation accuracy versus wall-clock training time for ResNet-50 (CIFAR-10, MRA).

https://doi.org/10.3390/aerospace13111021

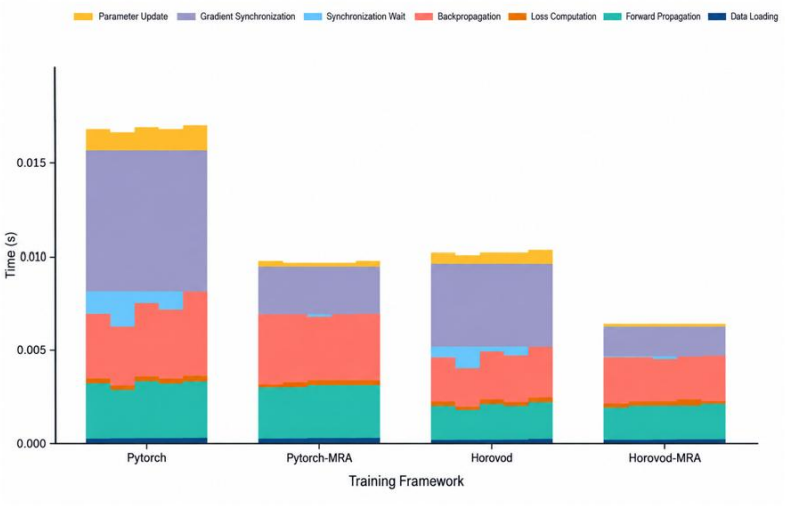


Figure 18. Breakdown of Single-Iteration Time for AlexNet (CIFAR-10, MRA).

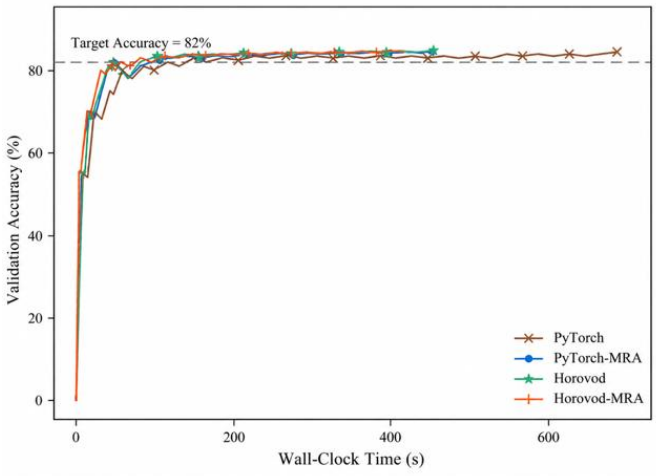


Figure 19. Validation accuracy versus wall-clock training time for AlexNet (CIFAR-10, MRA).

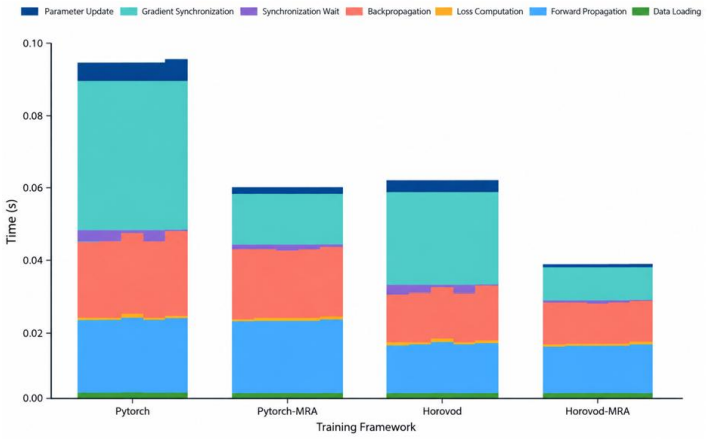


Figure 20. Breakdown of Single-Iteration Time for VGG-16 (CIFAR-10, MRA).

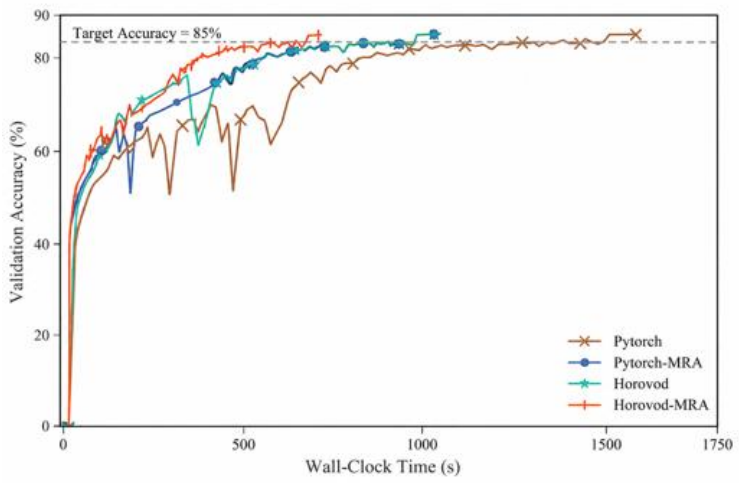


Figure 21. Validation accuracy versus wall-clock training time for VGG-16 (CIFAR-10, MRA).

5.4.2. Quantitative Performance and Generalization Across Dataset Scales

To evaluate the individual and joint contributions of IDAT and MRA, Tables 5 and 6 compare the training time and convergence accuracy of four configurations (Native, IDAT-only, MRA-only, and IDAT+MRA) on CIFAR-10 and Mini-ImageNet. The results show that IDAT alone achieves approximately 1.03×–1.04× speedup through adaptive workload balancing, whereas MRA provides greater acceleration by reducing gradient communication overhead and redundant parameter updates. Their combination consistently achieves the shortest training time across all evaluated configurations. Specifically, IDAT+MRA achieves speedups of 1.41×–1.64× on CIFAR-10 and 1.44×–1.82× on Mini-ImageNet, yielding an overall end-to-end speedup of 1.41×–1.82× over the corresponding native baselines.

Table 5. Training Time (s) and Accuracy (%) of Different Optimization Configurations on CIFAR-10.

Framework	Configuration	ResNet-50 Time (s)	Accuracy (%)	AlexNet Time (s)	Accuracy (%)
PyTorch DDP	Native	699.1	85.36	445.7	82.67
PyTorch DDP	IDAT	673.9	85.35	433.2	82.67
PyTorch DDP	MRA	495.3	85.37	279.8	82.66
PyTorch DDP	**IDAT+MRA**	**478.0**	**85.36**	**272.2**	**82.66**
Horovod	Native	454.4	85.36	289.7	82.66
Horovod	IDAT	440.3	85.36	282.2	82.65
Horovod	MRA	331.8	85.35	189.2	82.66
Horovod	**IDAT+MRA**	**321.5**	**85.35**	**184.3**	**82.65**

Note: Bold values highlight the results obtained using the proposed joint IDAT+MRA optimization method.

Table 6. Training Time (min) and Accuracy (%) of Different Optimization Configurations on the Mini-ImageNet Dataset.

Framework	Configuration	ResNet-50 Time (min)	Accuracy (%)	AlexNet Time (min)	Accuracy (%)
PyTorch DDP	Native	238.6	76.32	153.6	63.36
PyTorch DDP	IDAT	229.3	76.31	148.9	63.36
PyTorch DDP	MRA	136.2	76.31	92.5	63.36
PyTorch DDP	**IDAT+MRA**	**130.9**	**76.31**	**89.7**	**63.35**

Horovod	Native	149.2	76.30	91.6	63.35
Horovod	IDAT	144.3	76.32	88.9	63.36
Horovod	MRA	106.9	76.32	58.8	63.34
Horovod	**IDAT+MRA**	**103.4**	**76.31**	**57.1**	**63.34**

Note: Bold values highlight the results obtained using the proposed joint IDAT+MRA optimization method.

The performance gains vary with model-specific computation and communication characteristics. MRA is particularly effective for communication-intensive workloads, while IDAT primarily mitigates synchronization delays caused by heterogeneous worker performance. Their complementary effects explain the additional acceleration achieved by the joint framework. Furthermore, Tables 5 and 6 show that the final convergence accuracy remains comparable across all configurations, indicating that IDAT+MRA improves training efficiency without noticeable degradation in model accuracy.

As shown in Table 7, both IDAT and MRA reduce the wall-clock time required to reach the predefined target accuracy, although their contributions differ substantially. IDAT alone provides a moderate but consistent reduction in TTA. Under PyTorch DDP, the TTA decreases from 582 s to 561 s for ResNet-50, from 376 s to 365 s for AlexNet, and from 1392 s to 1343 s for VGG-16. Similar improvements are observed under Horovod, where the corresponding TTA values decrease from 381 s to 369 s, from 247 s to 241 s, and from 912 s to 884 s, respectively. These results are consistent with the approximately 3–4% training-time improvement of IDAT reported in Section 5.3.

Table 7. Time-to-Accuracy (TTA) on CIFAR-10 under Different Optimization Configurations.

Framework	Configuration	ResNet-50 TTA (s), 85%	AlexNet TTA (s), 82%	VGG-16 TTA (s), 85%
PyTorch DDP	Native	582	376	1392
PyTorch DDP	IDAT	561	365	1343
PyTorch DDP	MRA	415	237	901
PyTorch DDP	**IDAT+MRA**	**401**	**231**	**871**
Horovod	Native	381	247	912
Horovod	IDAT	369	241	884
Horovod	MRA	279	161	621
Horovod	**IDAT+MRA**	**270**	**157**	**602**

Note: Bold values highlight the results obtained using the proposed joint IDAT+MRA optimization method.

MRA provides a more pronounced reduction in TTA because it directly reduces gradient communication and parameter-update overhead. Under PyTorch DDP, MRA reduces TTA to 415 s, 237 s, and 901 s for ResNet-50, AlexNet, and VGG-16, respectively. Under Horovod, the corresponding values decrease to 279 s, 161 s, and 621 s. The larger improvement observed for AlexNet and VGG-16 is consistent with their greater sensitivity to communication and parameter-update overhead.

More importantly, the complete IDAT+MRA configuration achieves the lowest TTA across all evaluated models and frameworks. Under PyTorch DDP, IDAT+MRA reaches the predefined target accuracy in 401 s, 231 s, and 871 s for ResNet-50, AlexNet, and VGG-16, respectively. Under Horovod, the corresponding TTA values are further reduced to 270 s, 157 s, and 602 s. These results demonstrate that the computation-side workload balancing provided by IDAT remains complementary to the communication-side optimization provided by MRA.

5.5. Validation on Aerospace Workloads

Although the preceding experiments demonstrate the effectiveness of IDAT and MRA across different neural network architectures and dataset scales, CIFAR-10 and Mini-ImageNet are general-purpose computer vision benchmarks and therefore cannot directly demonstrate the applicability of the proposed framework to aerospace-related workloads. To address this limitation, we further evaluate the proposed framework on two representative learning tasks: automatic modulation recognition (AMR) based on I/Q communication signals and SAR ship detection based on remote-sensing imagery [31,32]. These tasks involve substantially different data modalities, model architectures, and optimization characteristics, enabling us to evaluate whether the proposed computation- and communication-side optimizations generalize beyond conventional image classification.

5.5.1. Automatic Modulation Recognition

Automatic modulation recognition is an important signal-processing task for intelligent wireless receivers and is relevant to signal perception, spectrum monitoring, adaptive transmission, and interference identification in space communication systems. To construct a representative communication-signal learning workload, we use the RadioML 2016.10A dataset. The dataset contains 11 modulation types over SNR levels ranging from −20 dB to +18 dB, with each sample represented by a 128-point two-channel in-phase/quadrature (I/Q) sequence.

A 1D-CNN-based classifier is employed to extract temporal and local features from the I/Q sequences. The model is trained for 80 epochs with a global batch size of 512. To ensure a fair comparison, the same model architecture, global batch size, optimizer, learning-rate configuration, and convergence criterion are maintained across all experiments. Four distributed training configurations are evaluated: PyTorch DDP (Native), PyTorch-IDAT, PyTorch-MRA, and PyTorch-IDAT+MRA. Total training time, speedup relative to native PyTorch DDP, synchronization waiting time per iteration, and final classification accuracy are recorded.

5.5.2. SAR Ship Detection

SAR ship detection is a representative aerospace remote-sensing task with important applications in maritime surveillance, target monitoring, and situational awareness. Recent studies have demonstrated the effectiveness of enhanced YOLO architectures for small-ship detection and oriented detection architectures based on edge deformable convolution and point-set representations for handling arbitrary ship orientations and geometric variations. These learning-based approaches also introduce substantial computation and parameter-synchronization workloads during distributed model training.

To evaluate the proposed framework on an aerospace remote-sensing workload, we employ a SAR ship detection dataset containing 1160 SAR images and 2536 annotated ship instances and adopt YOLOv5s as the detection model. The model is trained for 100 epochs with a global batch size of 64. The Native, IDAT, MRA, and IDAT+MRA configurations use identical model architectures and training hyperparameters. Total training time, speedup, synchronization waiting time per iteration, and mean average precision are used as evaluation metrics.

5.5.3. Acceleration Performance on Aerospace-Related Tasks

Table 8 shows consistent optimization trends across the two aerospace-related workloads. For RadioML AMR, IDAT reduces the training time from 827.3 s to 796.5 s and the synchronization waiting time from 12.6 ms to 5.8 ms per iteration, while MRA achieves a 1.35× speedup. Combining IDAT and MRA further reduces the training time to 589.7 s, yielding a 1.40× speedup, with the classification accuracy remaining within 92.12–92.15%.

https://doi.org/10.3390/aerospace13111021

For SAR ship detection, IDAT reduces the synchronization waiting time from 18.3 ms to 8.7 ms, while MRA achieves a 1.35× speedup. The complete IDAT+MRA configuration reduces the training time from 41.2 min to 29.3 min, corresponding to a 1.41× speedup, while maintaining mAP within 76.79–76.85%.

Table 8. Performance Comparison on Aerospace Learning Workloads.

Workload	Configuration	Training Time	Speedup vs. Native	Sync. Wait	Task Performance
RadioML AMR	Native	827.3 s	1.00×	12.6 ms	92.14%
	IDAT	796.5 s	1.04×	5.8 ms	92.12%
	MRA	612.4 s	1.35×	12.1 ms	92.15%
	IDAT+MRA	**589.7 s**	**1.40×**	**5.5 ms**	**92.13%**
SAR Ship Detection	Native	41.2 min	1.00×	18.3 ms	76.82%
	IDAT	39.7 min	1.04×	8.7 ms	76.79%
	MRA	30.5 min	1.35×	17.8 ms	76.85%
	IDAT+MRA	**29.3 min**	**1.41×**	**8.2 ms**	**76.81%**

Note: Bold values highlight the results obtained using the proposed joint IDAT+MRA optimization method.

These results confirm the complementary roles of IDAT and MRA: IDAT mitigates synchronization waiting caused by computational heterogeneity, whereas MRA reduces communication and parameter-update overhead. Their combination consistently achieves the best training efficiency without degrading task performance. The consistent gains on both I/Q signal classification and SAR object detection further demonstrate the applicability of the proposed framework beyond conventional image classification to communication-signal processing and aerospace remote-sensing workloads.

6. Conclusions

In this paper, we investigate the efficiency of synchronous data-parallel training in heterogeneous clusters and propose a joint optimization framework integrating IDAT for dynamic workload balancing and MRA for communication and parameter-update optimization. IDAT dynamically adjusts local batch sizes according to worker processing capabilities, while MRA reorganizes gradient synchronization and sharded parameter updates to reduce communication and redundant computation overhead. Experiments using ResNet-50, AlexNet, and VGG-16 on CIFAR-10 and Mini-ImageNet demonstrate the complementary benefits of the two mechanisms. IDAT achieves approximately 1.03×–1.04× speedup over the native baselines, while the complete IDAT+MRA framework achieves 1.41×–1.82× speedup with comparable convergence accuracy. Additional experiments on RadioML modulation recognition and SAR ship detection achieve approximately 1.40× and 1.41× speedup, respectively, further demonstrating applicability to aerospace-related workloads. Nevertheless, the current evaluation is limited to a relatively small heterogeneous cluster, and direct comparisons with ZeRO and FSDP remain to be conducted. Future work will focus on larger heterogeneous clusters, systematic comparisons with modern distributed-training frameworks, and additional space communication tasks, including channel estimation, interference detection, and beam management.

Author Contributions:. Conceptualization, R.W. and W.H.; methodology, R.W., W.H. and W.T.; software, R.W. and H.W.; validation, R.W. and H.W.; formal analysis, R.W., W.H. and H.W.; investigation, R.W. and H.W.; resources, W.H. and W.T.; data curation, R.W. and H.W.; visualization, R.W. and H.W.; writing—original draft preparation, R.W.; writing—review and editing, W.H., H.W.

and W.T.; supervision, W.H. and W.T.; project administration, W.H.; funding acquisition, W.H. and W.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (NSFC) under Grant Nos. W2432043 and 62371106, and the Sichuan Provincial "Posting and Bidding" Major Science and Technology Special Project under Grant No. 2025ZDZX0059.

Data Availability Statement: The data supporting the findings of this study are not publicly available because they include experimental configurations, performance measurements, and benchmark data generated during distributed training experiments, which are subject to confidentiality restrictions.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bhattacharyya, A.; Nambiar, S.M.; Ojha, R.; Gyaneshwar, A.; Chadha, U.; Srinivasan, K. Machine learning and deep learning powered satellite communications: Enabling technologies, applications, open challenges, and future research directions. *Int. J. Satell. Commun. Netw.* **2023**, *41*, 539–588.
2. Anjum, M.J.; Anees, T.; Tariq, F.; Shaheen, M.; Amjad, S.; Iftikhar, F.; Ahmad, F. Space-air-ground integrated network for disaster management: Systematic literature review. *Appl. Comput. Intell. Soft Comput.* **2023**, *2023*, 6037882.
3. Zhang, R.; Du, H.; Niyato, D.; Kang, J.; Xiong, Z.; Jamalipour, A.; Zhang, P.; Kim, D.I. Generative AI for space-air-ground integrated networks. *IEEE Wirel. Commun.* **2024**, *31*, 10–20.
4. Zhang, Y.; Wu, Y.; Liu, A.; Xia, X.; Pan, T.; Liu, X. Deep learning-based channel prediction for LEO satellite massive MIMO communication system. *IEEE Wirel. Commun. Lett.* **2021**, *10*, 1835–1839.
5. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*; NeurIPS: Vancouver, BC, Canada, 2019; Volume 32, pp. 8024–8035.
6. Sergeev, A.; Del Balso, M. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv* **2018**, arXiv:1802.05799.
7. Peng, Y.; Zhu, Y.; Chen, Y.; Bao, Y.; Yi, B.; Lan, C.; Wu, C.; Guo, C. BytePS: A high performance and generic framework for distributed DNN training. In Proceedings of Machine Learning and Systems, Austin, TX, USA, 2–4 March 2020; Volume 2, pp. 512–531.
8. Li, S.; Zhao, Y.; Varma, R.; Salpekar, O.; Noordhuis, P.; Li, T.; Paszke, A.; Smith, J.; Vaughan, B.; Damania, P.; et al. PyTorch Distributed: Experiences on accelerating data parallel training. *Proc. VLDB Endow.* **2020**, *13*, 3005–3018.
9. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
10. Subramanya, S.J.; Arfeen, D.; Lin, S.; Qiao, A.; Jia, Z.; Ganger, G.R. Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling. In Proceedings of the 29th Symposium on Operating Systems Principles, Koblenz, Germany, 23–26 October 2023; pp. 642–657.
11. Yao, F.; Zhang, Z.; Ji, Z.; Liu, B.; Gao, H. LBB: Load-balanced batching for efficient distributed learning on heterogeneous GPU cluster. *J. Supercomput.* **2024**, *80*, 12247–12272.
12. Narayanan, D.; Santhanam, K.; Kazhamiaka, F.; Phanishayee, A.; Zaharia, M. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation, Online, 4–6 November 2020; pp. 689–705.
13. Qiao, A.; Choe, S.K.; Subramanya, S.J.; Neiswanger, W.; Ho, Q.; Zhang, H.; Ganger, G.R.; Xing, E.P. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation, Online, 14–16 July 2021; pp. 855–870.
14. Ilhan, F.; Su, G.; Liu, L. ScaleFL: Resource-adaptive federated learning with heterogeneous clients. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Vancouver, BC, Canada, 17–24 June 2023.
15. Li, Y.; Zhang, X.; Zeng, T.; Duan, J.; Wu, C.; Wu, D.; Chen, X. Task placement and resource allocation for edge machine learning: A GNN-based multi-agent reinforcement learning paradigm. *arXiv* **2023**, arXiv:2302.00571.

16. Zhang, T.; Gao, L.; Lee, S.; Zhang, M.; Avest Imehr, S. TimelyFL: Heterogeneity-aware asynchronous federated learning with adaptive partial training. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Vancouver, BC, Canada, 17–24 June 2023.
17. De Sensi, D.; Costa Molero, E.; Di Girolamo, S.; Vanbever, L.; Hoefler, T. Canary: Congestion-aware in-network AllReduce using dynamic trees. *arXiv* **2024**, arXiv:2309.16214.
18. Luo, S.; Wang, R.; Xing, H. Efficient inter-datacenter AllReduce with multiple trees. *IEEE Trans. Netw. Serv. Eng.* **2024**, *21*, 4793–4806.
19. Wang, Z.; Wu, X.; Xu, Z.; Ng, T.S.E. Cupcake: A compression scheduler for scalable communication-efficient distributed training. *Proc. Mach. Learn. Syst.* **2023**, *5*, 373–386.
20. Rajbhandari, S.; Rasley, J.; Ruwase, O.; He, Y. ZeRO: Memory optimizations toward training trillion parameter models. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Atlanta, GA, USA, 9 November 2020; pp. 1–16.
21. Xu, H.; Zhang, W.; Fei, J.; Wu, Y.; Xie, T.; Huang, J.; Xie, Y.; Elhoseiny, M.; Kalnis, P. SLAMB: Accelerated large batch training with sparse communication. In Proceedings of the 40th International Conference on Machine Learning, Honolulu, HI, USA, 23–29 July 2023; Volume 202, pp. 38801–38825.
22. Zhao, M.; Yin, Y.; Mao, Y.; Liu, Q.; Chen, L.; Gao, Y. SparDL: Distributed deep learning training with efficient sparse communication. *arXiv* **2023**, arXiv:2304.00737.
23. Alimohammadi, M.; Markov, I.; Frantar, E.; Alistarh, D. L-GreCo: Layerwise-adaptive gradient compression for efficient and accurate deep learning. *arXiv* **2022**, arXiv:2210.17357.
24. Zheng, L.; Li, Z.; Zhang, H.; Zhuang, Y.; Chen, Z.; Huang, Y.; Wang, Y.; Xu, Y.; Zhuo, D.; Xing, E.P.; et al. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. In Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation, Carlsbad, CA, USA, 11–13 July 2022; pp. 559–578.
25. Thakur, R.; Rabenseifner, R.; Gropp, W. Optimization of collective communication operations in MPICH. *Int. J. High Perform. Comput. Appl.* **2005**, *19*, 49–66.
26. Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Ranzato, M.A.; Senior, A.; Tucker, P.; Yang, K.; et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*; NeurIPS: Lake Tahoe, NV, USA, 2012; Volume 25.
27. Zheng, S.; Meng, Q.; Wang, T.; Chen, W.; Yu, N.; Ma, Z.-M.; Liu, T.-Y. Asynchronous stochastic gradient descent with delay compensation. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 4120–4129.
28. Jiang, C.; Tian, Y.; Jia, Z.; Zheng, S.; Wu, C.; Wang, Y. Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping. *arXiv* **2024**, arXiv:2404.19429.
29. Chen, C.; Li, X.; Zhu, Q.; Duan, J.; Sun, P.; Zhang, X.; Yang, C. Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, La Jolla, CA, USA, 27 April–1 May 2024. https://doi.org/10.1145/3620666.3651379.
30. Wang, G.; Qin, H.; Jacobs, S.A.; Holmes, C.; Rajbhandari, S.; Ruwase, O.; Yan, F.; Yang, L.; He, Y. ZeRO++: Extremely efficient collective communication for giant model training. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024.
31. Guan, T.; Chang, S.; Wang, C.; Jia, X. SAR Small Ship Detection Based on Enhanced YOLO Network. *Remote Sens.* **2025**, *17*, 839.
32. Guan, T.; Chang, S.; Deng, Y.; Xue, F.; Wang, C.; Jia, X. Oriented SAR Ship Detection Based on Edge Deformable Convolution and Point Set Representation. *Remote Sens.* **2025**, *17*, 1612.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.