Fine-grained and Non-intrusive LLM Training Monitoring via Microsecond-level Traffic Measurement

Yibo Xiao
State Key Laboratory of Novel Software Technology, Nanjing University
Nanjing, China

Hao Zheng
State Key Laboratory of Novel Software Technology, Nanjing University
Nanjing, China

Haifeng Sun
National University of Singapore
Singapore, Singapore

Qingkai Meng*
State Key Laboratory of Novel Software Technology, Nanjing University
Nanjing, China

Jiong Duan
State Key Laboratory of Novel Software Technology, Nanjing University
Nanjing, China

Xiaohe Hu
Infrawaves
Beijing, China

Rong Gu
State Key Laboratory of Novel Software Technology, Nanjing University
Nanjing, China

Guihai Chen
State Key Laboratory of Novel Software Technology, Nanjing University
Nanjing, China

Chen Tian
State Key Laboratory of Novel Software Technology, Nanjing University
Nanjing, China

Abstract

Large language model (LLM) training is prone to anomalies due to its long duration and large scale, which can lead to significant performance degradation or even training crashes. Due to the synchronization nature of LLM training, anomalies exhibit the cascading effect, making their diagnosis challenging. Existing approaches rely on collecting communication operator information via code instrumentation, which yields only coarse-grained monitoring data and requires modifications to training code or communication libraries. We propose Pulse, a fine-grained, non-intrusive, and easy-to-deploy monitoring system. Our key idea is to enable fine-grained monitoring via traffic measurement. Pulse conducts microsecond-level RDMA traffic measurement on NICs, and transforms flow-level measurements into communication operator measurements, thereby enabling fine-grained and non-intrusive monitoring. We deploy Pulse on a testbed with 64 H200 GPUs and evaluate its anomaly localization capability under common failure scenarios. Pulse achieves machine-level localization in 10 out of 12 scenarios, while existing methods succeed in only 4 and even misdiagnose 2 of the remaining scenarios. Additionally, Pulse achieves over 90% precision and 100% recall, supports up to 2000 concurrent RDMA flow measurements per NIC, and imposes negligible overhead on training performance, making it a practical solution for real-world LLM training environments.

CCS Concepts: • **Networks** → **Network monitoring**; *Programming interfaces*; • **Hardware** → *Networking hardware*.

Keywords: Network Measurement; LLM Training Monitoring; Smart NICs

ACM Reference Format:
Yibo Xiao, Hao Zheng, Haifeng Sun, Qingkai Meng, Jiong Duan, Xiaohe Hu, Rong Gu, Guihai Chen, and Chen Tian. 2026. Fine-grained and Non-intrusive LLM Training Monitoring via Microsecond-level Traffic Measurement. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '26), March 22–26, 2026, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3779212.3790163

1 Introduction

Public AI training services [2, 3, 6] have become increasingly prevalent with the rapid advancement of large language models [1, 7, 27, 51, 52]. In this paradigm, customers rent GPU servers, upload their training code and communication libraries, and then launch the training job, while the service providers assume responsibility for ensuring reliable computation and networking. When anomalies occur, providers should rapidly localize them to minimize disruption and reduce the overall impact on training progress.

However, the long duration and large scale of LLM training jobs make them more prone to anomalies [11, 13, 46, 54, 57]. Moreover, due to the synchronization nature of training,

*corresponding author

ASPLOS '26, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-2359-9/2026/03
https://doi.org/10.1145/3779212.3790163

a single-node anomaly can rapidly propagate and trigger cascading anomalies across the system, leading to substantial waste of computational and networking resources. The high frequency of anomalies and the cascading effect pose greater challenges in anomaly diagnosis compared to traditional network environments. Thus, traditional monitoring methods [5, 12, 16, 28] often struggle to accurately localize anomalies that occur during training [13].

State-of-the-art (SOTA) methods designed for LLM training can be broadly categorized into two types: offline tools and online monitors. Offline tools provide benchmarks for GPUs and RDMA NICs (RNIC) [23, 24, 29, 55]. However, benchmarking is time-consuming and may fail to reproduce the anomaly. Online monitors typically gather information on communication operators [11, 13, 22, 54, 57], including start and end timestamps, along with the average transmission rate of each operator, for real-time anomaly localization. This ***operator-level (OP-level)*** information provides contextual insights when an anomaly occurs, enabling more efficient diagnosis. However, OP-level monitoring is too coarse-grained to achieve machine-level localization (*e.g.*, identify communication stragglers). In such cases, ***sub-OP-level monitoring*** at the microsecond scale is required for accurate localization (see § 2.3). Furthermore, existing methods require code instrumentation in the training framework or communication library, which is unfeasible for cloud service providers, as it requires clients to modify code and libraries.

An ideal monitoring system for LLM training in cloud environments should offer fine-grained monitoring and remain non-intrusive. OP-level monitoring is too coarse to localize anomalies accurately. Collective communication is a highly synchronized process, and OP-level monitoring cannot offer visibility into its internal progression, making straggler identification difficult. In addition, since communication involves coordination among GPUs, CPUs, and NICs, OP-level monitoring incorporates the computation overhead into the communication duration, failing to distinguish between computation and communication anomalies. Alongside fine-grained, a non-intrusive design further eliminates the need for code instrumentation, greatly facilitating cloud providers.

In this paper, we propose Pulse, a fine-grained, non-intrusive, and easily deployable monitoring system for LLM training. Our key idea is to enable fine-grained and non-intrusive monitoring of LLM training through microsecond-level RDMA traffic measurement. By deploying traffic measurement directly on the NICs, we eliminate the need for code instrumentation. We then convert traffic measurements into transmission rates of communication operators by leveraging function hooking, enabling both OP-level and sub-OP-level monitoring for each operator in a non-intrusive manner. Our contributions can be summarized as follows:

First, we propose microsecond-level RDMA traffic measurement with full precision on mainstream RNICs (*i.e.*, Nvidia Connect-X 6 Dx, Nvidia BlueField-3). We introduce a novel three-layer design that performs lightweight aggregation in the packet processing pipeline and offloads primary measurement tasks to NIC-embedded microprocessors. This ensures that primary measurement operations remain outside the critical path of packet processing. By combining on-path aggregation and off-path measurement, our method enables microsecond-level monitoring of thousands of RDMA flows on RNICs with negligible performance overhead.

Table 1. Monitoring data of each communication operator.

Monitor Level	Metric
OP-level	start timestamp, end timestamp
Sub-OP-level	microsecond-level transmission rate

Second, we associate RDMA flows with each communication operator in an application-agnostic way. By hooking into the NCCL and RDMA APIs, we infer the communication peer and data volume of each initiated built-in collective and Point-to-Point (P2P) operator for each GPU. We then introduce an operator segmentation algorithm that combines this expected volume and time interval to accurately identify the start and end points of each communication operator on the RDMA rate curves. After segmentation, we obtain both OP-level and sub-OP-level monitoring data for both built-in collectives and custom collectives (*i.e.*, collectives implemented as a set of P2P operators) as shown in Table 1.

Third, we achieve fine-grained yet lightweight anomaly localization. Our analysis and experimental validation show that accurate localization requires sub-OP-level monitoring of communication operators, typically at microsecond granularity. Instead of collecting raw microsecond-level rate data directly for anomaly localization, we extract two key metrics from fine-grained data that effectively capture the performance of each operator, enabling accurate machine-level localization with minimal data collection overhead.

We implement Pulse's measurement component on NVIDIA BlueField-3, while the other components are on the host. We deploy Pulse on a testbed with 64 H200 GPUs and reproduce common failure scenarios to evaluate its monitoring capability. Pulse accurately pinpoints the anomalous node in 10 of 12 scenarios and achieves group-level localization for the remaining. SOTA methods only achieve machine-level localization for 4 scenarios and even misdiagnose 2 of the remaining. Further evaluation shows that Pulse achieves over 90% precision and 100% recall, with an average diagnosis latency of approximately 6 seconds. Moreover, Pulse imposes negligible overhead on training performance and supports up to 2000 concurrent RDMA flow measurements per NIC with minimal impact on throughput and latency.

2 Anomaly Localization in LLM Training

In this section, we first provide the necessary background, then introduce SOTA methods, and finally analyze the necessity of fine-grained and non-intrusive monitoring.

Table 2. Typical causes of different anomaly types. We reproduce cases highlighted in bold in our evaluation.

Anomaly Type		Typical Causes
fail-stop	computation	ECC error, **GPU execution error**, GPU card drop, CPU error, **CUDA execution error**
	communication	**NIC down**, Switch reboot, AOC error, NVLink error
fail-slow	computation	GPU overheat, **GPU throttling**, **CPU throttling**, **CPU contention**
	communication	**PCIe downgrade**, **Memory alignment**, **PCIe contention**, **Network congestion**, **NIC-GPU cross NUMA binding**

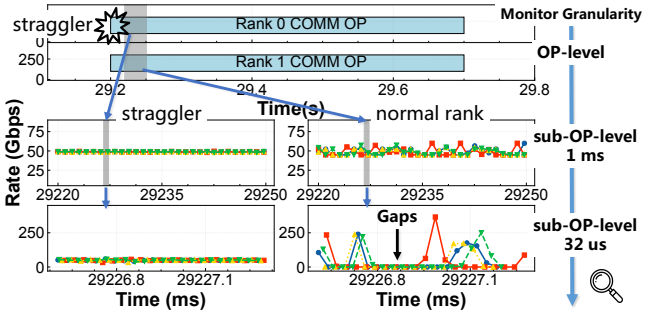

Figure 1. Communication straggler case under OP-level and sub-OP-level monitor. The testbed setup is detailed in § 7.1.

2.1 Background

Public AI training services. Cloud providers [2, 3, 6] offer on-demand public AI training services that allow users to rent GPU clusters and launch jobs, while providers are responsible for monitoring the underlying infrastructure. Upon detecting an anomaly, providers must promptly localize and replace the faulty node, and then resume the training job [13, 57]. Critically, since providers generally lack access to the users' code [13], monitoring should avoid intrusive code modifications. While anomaly localization is expected to operate online, root-cause analysis is generally conducted offline [13]. Our work focuses on anomaly localization.

Distributed training. As model sizes grow, a single GPU can no longer hold an entire model, motivating the use of parallelism to partition model parameters and data across nodes. Common approaches include tensor parallelism (TP) [34, 49], data parallelism (DP) [47], pipeline parallelism (PP) [19, 33], and expert parallelism (EP) [10, 25]. Training then relies on Collective Communication Libraries (CCLs) for synchronization, with NCCL [41] being the most widely used. CCLs utilize PCIe or NVLink for intra-node communication and RDMA for inter-node communication.

Anomaly taxonomy. Due to their extended duration and large scale, LLM training jobs are prone to anomalies. When anomalies occur, training typically manifests in two forms: fail-stop, which halts progress, and fail-slow, which prolongs iteration time. Anomalies can also be classified by origin into two types: computation and communication anomalies. Table 2 outlines common causes of each type based on prior studies [11, 13, 54, 57] and our operational experience.

2.2 State-of-the-Art Methods

Existing work broadly falls into offline and online methods. Offline methods run benchmarks on GPUs and RNICs to find abnormal devices. Superbench [55] offers end-to-end benchmarks for representative workloads and component-level benchmarks. Collie [24], Husky [23], and Hostping [29] focus on RDMA benchmarks for detecting communication anomalies. However, benchmarks may fail to reproduce the anomaly, and running benchmarks can be time-consuming.

Online methods collect runtime information to localize anomalies. MegaScale [22] uses CUDA events to monitor the execution time of critical code segments, requiring instrumentation within training code. Aegis [13] and Holmes [57] collect execution information of each operator by modifying CCLs. Greyhound [54] adopts a non-intrusive way by combining function hooking with CUDA events. These methods primarily collect OP-level monitoring data, including the start and end timestamps, along with the throughput; Aegis further records the count of work requests and completions for each operator. These methods typically rely on code instrumentation, which is impractical for cloud providers. Moreover, they are limited to OP-level monitoring, which fails to accurately localize anomalies, as described below.

2.3 Why Fine-grained, Non-intrusive Monitoring

We focus on online methods, as they generally achieve more efficient diagnosis by utilizing runtime information from training. We argue that effective anomaly localization in LLM training requires fine-grained, non-intrusive monitoring.

Necessity of fine-grained. OP-level monitoring provides coarse-grained visibility and has the following limitations:

L1. Lack of visibility into the progression of communication processes. In CCLs, data is partitioned into slices, with transmission and synchronization performed at the slice level. Each rank receives data slices from its predecessor, performs reductions, and forwards them to its successor. If one rank slows down during data transfer, other ranks stall, resulting in transmission gaps. Consequently, both straggler and normal ranks show identical OP-level durations, making the straggler hard to distinguish. Figure 1 illustrates monitoring at different granularities for such a case: OP-level monitoring fails to localize the straggler as all nodes exhibit identical durations. Moreover, straggler-induced gaps are only visible at 32 us but not at 1 ms; thus, at 1 ms, both straggler and normal ranks show similar rates, whereas at 32 us, stragglers can be identified by their lower rates. In § 6, we further dive into the required monitoring granularity to observe different gaps that classical OP-level monitoring cannot identify.

L2. Fail to distinguish between computation and communication anomalies. CCLs orchestrate communication

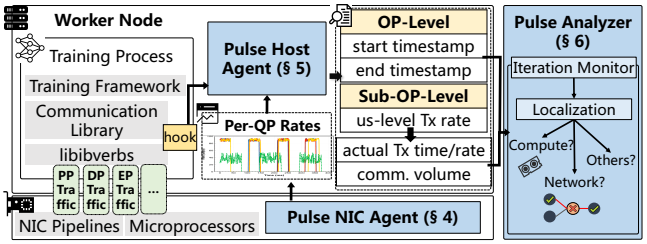

Figure 2. Pulse Architecture.

through three hardware components: the GPU, the CPU, and the NIC. GPUs perform reductions and move data between buffers, CPUs launch kernels and manage host-side coordination, and NICs transfer packets across nodes. Therefore, bottlenecks at CPUs or GPUs can also extend the overall duration, leading to misjudging a computation anomaly as a communication anomaly (see § 7.2.2).

Necessity of non-intrusive. Instrumentation-based monitoring is often impractical for cloud providers as it requires access to user code, and users may be unwilling to modify their code. A non-intrusive approach eliminates this dependency and facilitates deployment.

3 Design Overview

We propose Pulse, a traffic-centric monitoring system for LLM training. We aim to achieve the following three goals:

O1. High-precision, fine-grained monitoring of communication processes. We aim to provide sub-OP-level monitoring for each communication operator at the microsecond scale. This fine-grained monitoring captures critical details essential for anomaly localization.

O2. Application-agnostic monitoring without requiring code instrumentation. We aim to monitor each communication operator without requiring any modifications to training code or CCLs. This non-intrusive approach provides convenience for public cloud service providers.

O3. Low-overhead and easily deployable monitoring. We aim to monitor training with minimal overhead and utilize widely available hardware, enabling scalable deployment in large-scale production environments. This makes Pulse a practical solution for real-world LLM training scenarios.

3.1 Design Challenge

The core idea of Pulse is to provide monitoring via traffic measurement. Pulse conducts microsecond-level per-QP (Queue Pair) RDMA rate measurement and converts these measurements into the transmission rates of communication operators. This approach eliminates the need to instrument training code or CCLs and provides fine-grained monitoring with minimal performance interference, as it can run in an isolated process or directly on hardware. However, designing such a monitoring system presents three main challenges:

C1. How to conduct high-precision, microsecond-level RDMA measurement? RDMA's hardware offloading makes host-side measurement difficult. Existing approaches [8, 50, 60] are primarily deployed on P4 platforms, but they suffer from limited precision due to architectural constraints, and P4-compatible hardware is not yet widely available.

C2. How to bridge traffic layer information with application layer information in an application-agnostic way? Due to the decoupled nature of network stack layers, it is challenging to establish a clear mapping between RDMA traffic and collective communication operators directly.

C3. How to conduct lightweight anomaly localization with fine-grained data? Although fine-grained monitoring enables accurate anomaly localization, it also generates massive amounts of data. Directly collecting raw measurement data for localization introduces significant overhead.

3.2 Pulse Architecture

As shown in Figure 2, Pulse has three functional components: Pulse NIC Agent, Pulse Host Agent, and Pulse Analyzer. The workflow of the Pulse system is as follows:

1. To address **C1**, we exploit the programmability of modern RNICs and propose a three-layer on-NIC measurement design. Building on this design, **Pulse NIC Agent** conducts microsecond-level RDMA traffic measurement and periodically uploads the results to the host. (§ 4)
2. To address **C2**, we propose an operator segmentation algorithm that identifies the start and end points of each operator on RDMA rate curves. Based on this, **Pulse Host Agent** converts the per-QP measurements into the transmission rate of each communication operator. (§ 5)
3. To address **C3**, we extract two key metrics from the transmission rates of each built-in and custom collective, respectively, which accurately capture the network performance. **Pulse Analyzer** only collects them along with OP-level data for localization. (§ 6)

Assumptions and Limitations. First, Pulse relies on per-flow RDMA measurements and thus is limited to inter-node collective monitoring. Proprietary scale-up network protocols (*e.g.*, NVLink) do not expose per-flow transmission visibility. Second, Pulse does not currently support monitoring collectives using CollNet or NVLS, as CollNet requires specialized hardware and NVLS is designed for intra-node communication. Third, Pulse's parallelism identification method (see § 5.2) may require adaptation when the user's parallelism implementation deviates from mainstream frameworks (*i.e.*, Megatron [49] or DeepSpeed [48]).

4 Traffic Measurement on NIC Agent

In this section, we propose a novel three-layer measurement design based on mainstream RNICs (*i.e.*, Nvidia Connect-6 Dx, Nvidia Bluefield-3). We first analyze the rationale for deploying measurements on NICs, then describe the three-layer design, and finally discuss its scalability.

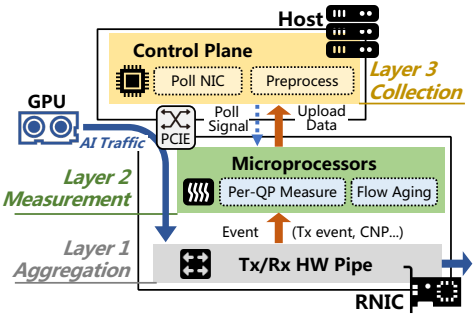

Figure 3. Three-layer measurement architecture.

4.1 Why Measure on NICs

SOTA host-based and P4 switch-based measurement works fail to provide high-precision RDMA traffic measurement [15, 17, 50, 56, 60]. Unlike TCP, RDMA's hardware offloading makes accurate host-side measurements difficult. P4-based solutions are constrained by switch architecture and limited on-chip memory, often requiring approximation techniques (*e.g.*, sketches) that reduce precision. This loss in precision degrades operator segmentation and, ultimately, monitoring effectiveness. These limitations motivate the exploration of NICs as the target platform for RDMA measurement.

To support customized capabilities, modern RNICs integrate embedded microprocessors, *e.g.*, RISC-V cores available on BlueField-3 [37] and ConnectX-6 Dx [14], which we identify as a promising platform for RDMA measurement. First, their proximity to the packet-processing pipeline enables access to on-NIC events (*e.g.*, packet transmission and NACK reception), and some even support direct packet manipulation [9, 40]. Second, these microprocessors typically follow the RISC architecture and the Run-To-Complete (RTC) execution model, which imposes fewer programming constraints. Finally, they are typically equipped with larger memory resources (*e.g.*, 1-GB DPA-accessible DDR region in BlueField-3 [9]), making them well-suited for lossless microsecond-level measurement. Currently, NVIDIA's ConnectX-6 Dx and BlueField-3 expose the programming interfaces of these microprocessors and have been widely deployed in modern data centers (*e.g.*, Alibaba implements a customized HPCC algorithm [26] on these RNICs). Therefore, we target these two RNICs as our measurement platform.

4.2 Three-Layer Measurement Architecture

On-NIC microprocessors provide a promising platform for measuring RDMA traffic. However, redirecting all packets to microprocessors imposes significant processing and memory overhead, and places the measurement tasks on the critical path of packet processing, degrading both throughput and latency. To achieve high-precision and low-overhead measurement, Pulse NIC Agent adopts a three-layer architecture: aggregation layer on the packet processing pipeline, measurement layer on the microprocessor, and collection layer on the host, as illustrated in Figure 3.

Aggregation layer aggregates per-flow packets by utilizing the event mechanism. Specifically, it registers a transmit event that triggers a handler on microprocessors for further measurement whenever a specified amount of data has been transmitted. Aggregation not only relieves microprocessors from processing every packet but also keeps measurement operations off the packet-processing critical path, thereby minimizing interference with application traffic.

Measurement layer performs per-QP rate measurement within the event handler. Upon the arrival of a transmission event for a given flow, it indexes this flow and records the transmitted data volume within the corresponding time window, *i.e.*, epoch. Since an RDMA RC (Reliable Connection) can be uniquely identified by its 24-bit source QPN (Queue Pair Number) on each RNIC, the measurement layer uses a direct-address table with 16M entries, enabling constant-time indexing without hash collisions. For memory allocation, although microprocessors provide large memory capacity, pre-allocating memory for all possible flows causes excessive usage. To mitigate this, we design an epoch pool to allocate memory for active flows in batches dynamically. Moreover, to promptly reclaim memory from silent or disconnected flows, we design a flow aging mechanism coordinated with the host-side collection layer, as described below.

Collection layer on the host is responsible for polling the microprocessors to retrieve measurement data via PCIe periodically. During each polling event, the measurement layer performs flow aging: flows that have not transmitted data since the last poll are marked as inactive, and their associated memory is reclaimed promptly. The uploads of measurement data are handled asynchronously to minimize host CPU overhead: the host CPU only needs to initiate the polling process and wait for the microprocessors to respond once they complete uploading the measurement data. The per-QP measurement data is first preprocessed and then handed over to Pulse Host Agent, where it is correlated with the corresponding communication operators.

4.3 Scalability Analysis

Our design targets measuring thousands of active RDMA flows per RNIC, sufficient for LLM training [11, 22, 46]. We analyze its scalability in terms of compute overhead and memory footprint. Compute overhead is mainly determined by event frequency. An event is triggered when a flow sends a predefined amount of data ϕ, rather than at every epoch. Thus, the event frequency depends on NIC throughput Thp and is calculated as $\frac{Thp}{\phi}$. For a 400 Gbps NIC with $\phi = 4$ KB, it triggers at most about 12.5M events/s. A single DPA thread can handle about 0.8M events/s in our evaluation, so 16 of 256 available DPA threads suffice for the maximum load.

For memory footprint, let e denote the epoch length, s the storage size per epoch, T the host polling period, N the number of flows, and E the number of epochs each flow

needs. The total memory consumption M is calculated as:

$$M = N \times E \times s = N \times T/e \times s \quad (1)$$

In practice, we set N, e, and T to 2000, 32 us, and 1 s, respectively. Each epoch records the transmitted data in units of 32 bytes, requiring 2 bytes per epoch to avoid overflow under a 400 Gbps network. Including the directly-address table (16M entries $*$ 4 = 64 MB), the total memory usage is $64\ \text{MB} + 2000 * (\frac{1}{32*10^{-6}}) * 2\ \text{B} \approx 184\ \text{MB}$, which is well within the limits of modern microprocessors (1 GB of memory [9]). Our evaluation in § 8 shows that Pulse supports microsecond-level measurement for 2000 active RDMA flows per NIC, with negligible impact on throughput and latency.

5 Flow-Operator Association on Host Agent

Pulse Host Agent intercepts NCCL operator API calls and associates them with per-QP measurements, *i.e.*, identifying the start and end points of each operator in the measurement data. Our key insight is that, given the expected data volume and communication peer of each operator, we can segment the measurement data of the corresponding peer based on the expected volume, thereby determining the operator's execution boundaries. In this section, we detail how to derive the expected data volume and peer of each operator (§ 5.1) and conduct operator segmentation (§ 5.2). Then we discuss support for monitoring custom collectives in § 5.3.

5.1 Expected Volume and Peer Derivation

Figure 4 provides an overview of how to derive the expected volume and peer. During training initialization, Pulse hooks into RDMA and NCCL APIs to perform two preparatory steps for each GPU: ① determine source and destination GPUs of each flow, since per-QP measurement only provides Global Identifier (GID) and QPN information. ② Obtain the membership of the communication groups to which each GPU belongs. During training runtime, Pulse intercepts each operator and infers its expected data volume and peers.

Preparatory steps during training initialization. Pulse hooks into RDMA and NCCL initialization functions to collect GPU-to-QP mappings and group membership. For ①, by intercepting the function `ibv_modify_qp`, we trace the creation process of each QP. Combining the process-to-GPU usage information provided by `nvidia-smi`, we identify which QPs are used by each GPU. An all-gather operation is then performed to collect GPU-QP mapping from other ranks, enabling us to determine the destination GPU for each QP. For ②, by hooking into the initialization API of NCCL, we can obtain the `commHash` (the communication group ID in CCLs) and the corresponding rank of the current GPU within each group. Another all-gather operation, followed by grouping based on `commHash`, is used to retrieve the membership information of each communication group.

Expected volume and peer derivation during training. Pulse intercepts each operator and derives its expected

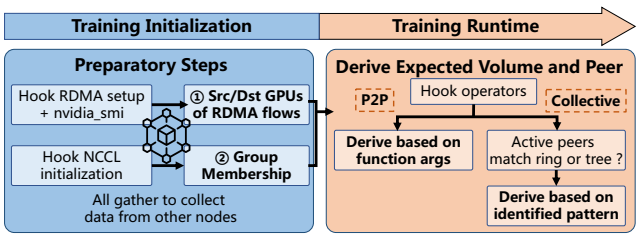

Figure 4. Expected volume and peer derivation overview.

volume and peer. For **P2P operators** (*i.e.*, send, recv), the communication peer and volume are explicitly specified via function arguments (*i.e.*, `peer`, `count`, and `datatype`). For **collective operators**, although the total number of elements is provided (*i.e.*, `count`), the data volume each GPU needs to transmit to other GPUs depends on the communication algorithm used. This algorithm is dynamically selected based on `count` and cannot be directly determined through function hooking. To address this, **our key observation is that different communication algorithms require a GPU to transmit data to different sets of peers.**

Pulse infers the communication algorithm of a collective operator by analyzing each GPU's active peer pattern (*i.e.*, the peers to which the GPU is transmitting data). Excluding algorithms that require specialized hardware (*i.e.*, CollNet and NVLS), NCCL adopts the ring-based and tree-based [20, 36] algorithms. The ring-based algorithm adopts a logical ring topology, where each rank sends data to its predecessor and successor. The tree-based algorithms utilize a logical tree topology, where each node communicates with its parent and child nodes. Therefore, when a GPU initiates a collective operation within a group, Pulse first checks RDMA rate measurements to identify its active peers. Pulse then matches the active peer pattern against the logical topology of each algorithm, inferring the algorithm in use and calculating the expected volume accordingly. We provide the expected volume of each algorithm in Appendix B. Furthermore, due to the periodicity of communication operators in LLM training, identification results can be reused for subsequent ones.

5.2 Communication Operation Partition

Pulse Host Agent segments the RDMA rate curve based on the expected volume to identify the start and end of each operation. However, several factors cause the actual communication volume to deviate from its theoretical expectation:

- Collective communication operations often involve additional synchronization messages between nodes.
- The theoretical communication volume is computed without considering the packet headers overhead.
- In the case of packet loss, RNICs need retransmissions, thereby increasing the overall data volume.

We propose an operator segmentation algorithm (Alg. 1) to handle deviations between the expected and actual communication volume. This design combines expected volume and

Algorithm 1 Segmentation Algorithm

Input: [(time, rate)], size
Output: starts, ends
1: **Function** segmentation **do**
2: $last\_t \leftarrow 0, tmp \leftarrow 0$
3: **for** (t, r) in [(time, rate)] **do**
4: **if** $t - last\_t \geq \delta \ \wedge \ tmp \geq size$ **then**
5: $ends.append(last\_t)$
6: $tmp \leftarrow 0$
7: **end if**
8: **if** $tmp == 0$ **then**
9: $starts.append(t)$
10: **end if**
11: $last\_t \leftarrow t, \ tmp \leftarrow tmp + epoch\_length * rate$
12: **end for**
13: return $starts, ends$
14: **end Function**

time interval, motivated by two observations: (1) the theoretical volume provides a lower bound of the actual volume, and (2) communication operators occur periodically rather than continuously. Pulse identifies the end of a communication operator based on two conditions (Line 4): the time interval exceeds a threshold δ, and the transmitted volume exceeds the expected value. Although NCCL introduces gaps during transmission, these are typically at the microsecond scale or occur at the start of transmission (see § 6). By setting the threshold to $O(\text{ms})$ and enforcing the volume condition, Pulse effectively excludes the interference caused by NCCL gaps. After segmentation, Pulse obtains both OP-level and sub-OP-level monitoring data for each operator.

Besides, we propose a decision-tree-based algorithm to infer the parallelism strategy. While per-operator monitoring data suffices for anomaly localization in LLM training by utilizing node similarity to detect outliers [11, 13], users may require higher-level training metrics, *e.g.*, DP communication time or per-microbatch duration in PP, which can be derived once the parallelism strategy is identified. Figure 5 illustrates our algorithm, which leverages the communication characteristic of each parallelism. DP and TP rely on collective operators, while PP and EP rely on P2P operators[1]. DP and TP are distinguished by their relative data volume and operation count: DP synchronizes GB-scale model gradients per iteration, while TP synchronizes intermediate activations at every layer. Small-volume operations, typically used to gather node-wide information (*e.g.*, loss), are labeled as None. EP and PP are differentiated by P2P communication patterns: a sequential send pattern (1->..->N->..->1) is labeled as PP, and a full-mesh pattern is labeled as EP.

We acknowledge that our algorithm may require adaptation when a user's parallelism implementation deviates from our assumptions. Parallelism identification is primarily needed when users care about higher-level metrics; in such cases, we can collaborate with users to obtain framework-specific information and accordingly refine the decision tree.

[1]NCCL does not provide an all-to-all operator before NCCL 2.28. In mainstream frameworks, it is implemented using the P2P operators.

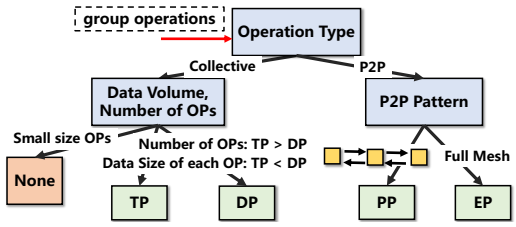

Figure 5. Identification of the parallelism strategy used.

```
void all2all_single_unequal_split() {
  NCCL_CHECK(ncclGroupStart());
  for (const auto r : c10::irange(numranks)) {
    if (_nccl_should_send_recv(sendcounts[r]))
      NCCL_CHECK(ncclSend(...);
    if (_nccl_should_send_recv(recvcounts[r]))
      NCCL_CHECK(ncclRecv(...));
  }
  NCCL_CHECK(ncclGroupStart());
}
```

Figure 6. The implementation of all-to-all in PyTorch [45].

5.3 Custom Collective

Pulse can monitor the transmission rates of custom collectives because they ultimately execute as a set of P2P operators. The standard way to implement a custom collective is using multiple P2P operators [43] and then encapsulating these operators within NCCL Group calls [42] (*i.e.*, `ncclGroupStart` and `ncclGroupEnd`), to avoid deadlocks and reduce launch overhead. Figure 6 presents a PyTorch code snippet [45], which utilizes P2P operators to implement the `all-to-all`, a widely used custom collective in LLM training [10, 52]. To monitor the transmission rates of such collectives, Pulse first hooks into the NCCL group calls and identifies the set of P2P operators that belong to the same custom collective. Then, as described in § 5.1 and § 5.2, Pulse monitors the rates of each P2P operator, thereby obtaining the transmission rate of the custom collective.

6 Pulse Analyzer

In this section, we first analyze the required monitoring granularity. Then, we present how to conduct fine-grained yet lightweight anomaly localization that avoids uploading all sub-OP-level data to the analyzer.

6.1 Monitoring Granularity Analysis

We discuss two common transmission gaps that OP-level design cannot capture. These gaps manifest at millisecond and microsecond timescales and therefore ultimately require microsecond-level monitoring for accurate localization.

Millisecond-level gaps due to receiver-side execution delays. Gaps occur when the delayed execution of receiver communication kernels prevents timely buffer consumption,

thereby stalling sender-side data transmission. These gaps are common in MoE training (see § 7.2.4) and have not been reported by prior works. When the application initiates communication operators, NCCL notifies proxy threads and launches the kernel. **On the sender side**, the proxy thread first waits for the GPU to copy data into the pre-allocated buffer, then waits until the receiver is ready. It subsequently sends the data, and upon completion, notifies the GPU that the buffer is available. **On the receiver side**, the proxy thread first advertises buffer readiness to the sender, receives data into the buffer, and then relies on the receiver communication kernel to consume the buffered data.

The above process repeats when the data size exceeds the buffer capacity. Consequently, if the receiver communication kernel is delayed, the sender can only transmit a buffer-sized portion of the data and stalls until the receiver makes progress, thereby causing transmission gaps. The scale of these gaps depends on the severity of the kernel delay, which is typically millisecond-scale in our evaluation (see § 7.2.4).

Microsecond-level gaps due to stragglers. Although millisecond-level monitoring can capture receiver-induced transmission gaps that OP-level design fails to observe, we demonstrate in § 2.3 that microsecond-level monitoring is necessary for straggler localization. To mitigate transmission bubbles [35], NCCL subdivides each collective operation into multiple channels (at least two) for parallel data transmission. For inter-node communication, each channel typically utilizes one QP. The basic unit of transmission and synchronization is the slice, typically no larger than 1 MB. On a 400 Gbps network, assuming NCCL utilizes two channels, transmitting a slice takes approximately 40 us. If a straggler's bandwidth drops to $\frac{1}{p}$ of the original, it requires $40p$ us to transmit a slice, creating a gap of $40(p-1)$ us on the normal nodes. When the monitoring granularity exceeds $40(p-1)$ us, such gaps become invisible due to insufficient temporal resolution. Therefore, microsecond-level monitoring is necessary unless a performance degradation to $\frac{1}{25}$ of the normal rate (*i.e.*, $p = 25$) can be tolerated. As network speeds scale to 800 Gbps, the requirements on monitoring granularity will become more stringent. We validate this analysis in § 7.2.1.

6.2 Anomaly Localization

Communication Anomaly Localization. For NCCL built-in collective (*e.g.*, all-reduce), We extract two key metrics from the microsecond-level rate data to efficiently pinpoint communication anomalies: actual communication time and communication volume.

- In cases of communication fail-slow, stragglers cause gaps on non-straggler ranks. Therefore, actual communication time is calculated by summing the epochs with a nonzero rate, explicitly excluding gaps. Stragglers can be identified by their larger actual communication times. Besides, inter-node communication relies on CPU proxy threads to coordinate data movement and synchronization between GPUs and NICs. When CPU bottlenecks occur, time spent in proxy threads increases, extending the overall duration. Using actual communication time avoids misjudging such computation bottlenecks as communication anomalies.
- In cases of communication fail-stop, the faulty rank halts transmission first and consequently sends the least data during the current operator. Communication volume is calculated as the amount of data each node sends during the current operator. Thus, faulty nodes can be identified by their smallest communication volumes.

Each node only uploads these two metrics along with OP-level data to Pulse Analyzer while retaining the microsecond-level rate data from the most recent iterations locally, which significantly reduces the overhead of data collection. The retained data can later be used for root-cause analysis of anomalies (*e.g.*, bugs in congestion control algorithms [13]).

For custom collectives, we extract two key metrics from the microsecond-level rate data: rank-level actual transmission rate and completion status of each P2P operator.

- In cases of communication fail-slow, as custom collectives are composed of a set of P2P operators, we focus on the performance of the operator set as a whole. We first calculate the rank-level actual communication time by summing up the epochs during which at least one P2P operator is actively transmitting. Then, since traffic volumes can vary across P2P operators within a custom collective (*e.g.*, `all-to-all` in MoE models), we further calculate the rank-level actual transmission rate by dividing the total traffic volume of all constituent P2P operators by the rank-level actual communication time. In § 7.2.5, we observe that per-P2P metrics exhibit noticeable jitter: even P2P operators on non-straggling ranks may exhibit low rates since each P2P operator may encounter varying degrees of congestion. Using rank-level metrics helps mitigate this noise. Pulse Analyzer further mitigates such jitter by comparing the average of this metric over the most recent k (we set to 10) collectives to localize stragglers.
- In cases of communication fail-stop, P2P operators involving the faulty node fail to complete. Accordingly, we determine the completion status of each P2P operator by comparing its actual traffic volume with its expected traffic volume (as determined in § 5.1). Pulse Analyzer then localizes the faulty node by identifying which node exhibits such uncompleted operators.

Computation Anomaly Localization. With start and end timestamps of each communication operator, we can reconstruct the timeline of computation and communication during training. For a custom collective composed of a set of P2P operators, we take the earliest start timestamp and the latest end timestamp among these operators as the start and end timestamps, respectively. When computation anomalies occur, the subsequent communication operator is the first

to be affected. In cases of computation fail-slow, this effect appears as a delayed start timestamp, while in computation fail-stop, it appears as a missing start timestamp. Therefore, Pulse Analyzer scans operators in chronological order to find the first one exhibiting the corresponding behavior, similar to the approach adopted in prior work [13, 57].

In addition to the forward and backward passes, computation fail-slow can also affect communication. Bottlenecks on CPU or GPU may prolong the execution of proxy threads or GPU kernels, thereby increasing the overall duration. Pulse Analyzer identifies a node as computation fail-slow when it observes an extended duration while the start timestamp and actual communication time remain normal.

7 Case Study

In this section, we first introduce the testbed setup. Then, we present several anomaly localization cases using Pulse.

7.1 Implementation and Testbed Setup

We implement Pulse NIC Agent on the BlueField-3 SuperNIC with DOCA 2.9.0 [38], and develop Pulse Host Agent and Pulse Analyzer on the host. Pulse Host Agent hooks functions using the `LD_PRELOAD` mechanism. The implementation details of Pulse NIC Agent are provided in Appendix A. For the testbed setup, we rented eight machines from a cloud provider. Each machine is equipped with two AMD EPYC 9575F 64-core processors, eight NVIDIA H200 GPUs, and eight BlueField-3 SuperNICs. All SuperNICs are interconnected via 400 Gbps RoCEv2 links to a switch, with GPUDirect RDMA enabled. We adopt the NTP protocol [31] for clock synchronization across servers. Each server utilizes the PCC mailbox [39] to communicate with the DPA, synchronizing the host and NIC clocks, thereby ensuring all NICs are aligned on a unified timeline. For LLM training, we employ the Megatron framework [49].

7.2 Anomaly Localization Cases

To evaluate Pulse's localization capability on both built-in and custom collective operators, we inject 12 failure types listed in Table 3 during the training of GPT-2 and Mixtral 8×7B [21], ensuring coverage of all anomaly types in Table 2. Mixtral 8×7B relies on `all-to-all` for token dispatch and combination, which represent the most widely used custom collectives in LLM training. For other types of custom collectives, we run standalone collective workloads and inject failures. The epoch length is set to 32 us. We compare Pulse with state-of-the-art methods (*i.e.*, Aegis [13], Holmes [57], GrayHound [54]) that rely on OP-level monitoring for localization. The results are summarized in Table 3, with several representative cases discussed below and the remaining detailed in Appendix D. In production environments, various failures may occur, some of which are difficult to reproduce in our testbed (*e.g.*, hardware faults).

Table 3. Anomaly localization results.

●: machine-level localization. ◐: group-level localization.
○: misjudging computation anomaly as communication anomaly.

Case	Failure injection	Pulse	SOTAs
Network congestion	Launch Background RDMA traffic	●	◐
CPU contention	Use `stress` tool	◐	○
NIC down	Disable one RNIC	●	◐
MoE expert imbalance	None. Inherent compute stragglers	●	●
PCIe contention	Launch loopback RDMA traffic	●	◐
GPU throttling	Use `nvidia-smi` tool	●	●
GPU execution error	Invalid address access	●	●
CUDA execution error	Allocate excessive memory	●	●
CPU throttling	Adjust `scaling_governor`	◐	○
PCIe downgrade	Use `setpci` tool	●	◐
NIC-GPU cross NUMA binding	Manually bind	●	◐
Memory alignment	Allocate misaligned memory buffer	●	◐

7.2.1 Network Congestion (GPT-2). We induce congestion by launching background traffic, reducing the bandwidth of a network path by half. We train the GPT-2 70B model with 8-way TP, 4-way ZeRO-2 DP, and 2-way PP. Pulse finds that the third iteration is 500 ms slower than the others and starts anomaly localization. Figures 7 and 8 show the transmission rates of the first GPU at each node in the third iteration for PP and DP groups, respectively. The forward and backward computations of the i-th microbatch are annotated as "Fi" and "Bi," respectively. Note that Pulse Analyzer does not need to collect rate data from each node (see § 6); this is presented here only for better illustration.

Pulse precisely localizes the straggler, while SOTA methods are limited to group-level localization. Pulse first checks per-operator monitoring data within the PP group, which are uniform across GPUs and indicate no computation anomaly. It then examines the DP group. Using actual communication time, Pulse precisely identifies the straggler: as shown in Figure 9, GPU 1 on Node 1 exhibits a longer actual time than the others. SOTA methods only observe that GPUs in Group 1 have identical durations, requiring additional benchmarking [54, 57] to identify the straggler. Since DP groups are typically large (*e.g.*, 128-way [27]), narrowing diagnosis from the group level to the node level is essential for accelerating diagnosis and reducing resource waste.

Validation of the granularity requirement. We set the epoch length to 64 us to validate the monitor granularity requirement analysis in § 6. Our analysis indicates that a 40 us granularity is required when a straggler's bandwidth is halved. In Figure 9, with a 64 us granularity, the straggler indeed exhibits the longest actual communication time, but other nodes in Group 1 appear 150 ms longer than normal. This discrepancy arises because insufficient monitor granularity folds some gaps into the actual communication time, aligning with our analysis.

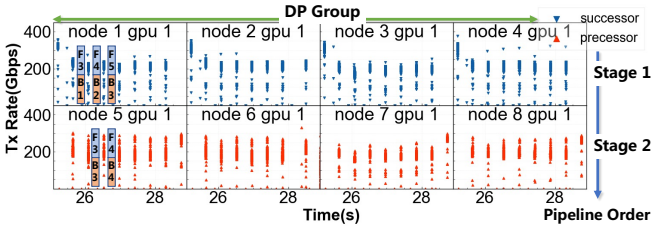

Figure 7. PP groups Tx rates in network congestion case.

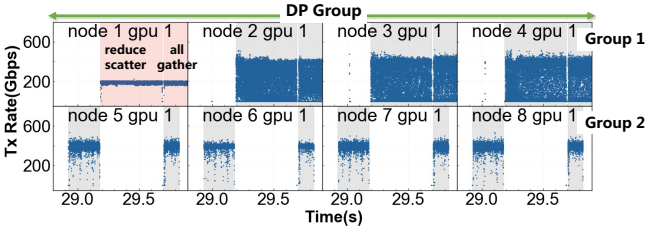

Figure 8. DP groups Tx rates in network congestion case.

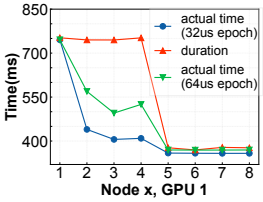

Figure 9. Metrics in network congestion case.

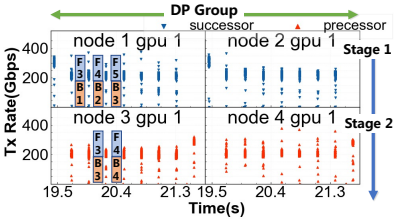

Figure 10. PP group Tx rates in CPU contention case

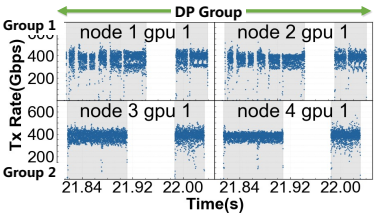

Figure 11. DP group Tx rates in CPU contention case

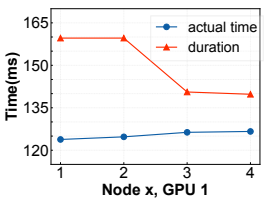

Figure 12. Metrics in CPU contention case.

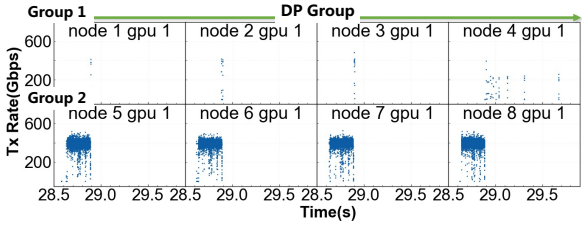

Figure 13. DP groups Tx rates in NIC down case.

7.2.2 CPU Contention (GPT-2). We use the tool `stress` to saturate all CPU cores on one node. Pulse identifies this computation bottleneck, while SOTA methods misjudge it as a communication anomaly. We train the GPT-2 32B with 8-way TP, 2-way ZeRO-2 DP, and 2-way PP. First, durations of forward and backward passes are regular and uniform in Figure 10, indicating no anomalies in these phases. Pulse then checks the DP groups. Figures 11 and 12 show that Group 1 exhibits longer durations but maintains normal transmission rates and actual communication times. Existing SOTA methods can only observe extended durations and misjudge them as communication anomalies. Since Pulse observes normal actual communication times across GPUs, it rules out communication issues and attributes the extended durations to the computational bottleneck.

7.2.3 NIC Down (GPT-2). We disable one NIC during training. Pulse accurately localizes the faulty node while SOTA methods only identify a group of nodes. We train the GPT-2 70B model with 8-way TP, 4-way ZeRO-2 DP, and 2-way PP. The localization process follows a similar approach to prior cases, and we present only the key step here. As shown in Figure 13, a communication hang occurs in Group 1. By examining the per-node communication volume of the affected operator, Pulse identifies Node 1, GPU 1 as the anomalous device, since it transmits the least amount of data. SOTA methods only observe that the entire communication group has hung, failing to achieve machine-level localization.

7.2.4 MoE Expert Imbalance (Mixtral 8×7B). The uneven routing in MoE models induces inherent computation stragglers, which are correctly identified by both Pulse and the SOTA methods. However, it induces transmission gaps on other ranks, which can be misdiagnosed as communication anomalies by SOTA methods. We train the Mixtral 8×7B model with 8-way TP, 4-way DP on attention layers, and 8-way TP, 4-way EP on expert layers, without injecting failures. Figure 14 shows the transmission rates of each P2P operator for two `all-to-all` operations within one EP group, corresponding to token dispatch and token combination. Pulse and SOTA methods identify rank 1 as a straggler based on its latest start timestamp of the second operator, which is consistent with the token dispatch traffic map in Figure 16. Furthermore, uneven routing causes varying computation durations (blue arrows) and thus varying start times of the second `all-to-all` kernel, resulting in gaps of varying lengths. Ranks with larger gaps can be misdiagnosed as communication anomalies by SOTA methods due to their longer duration. Pulse explicitly excludes the gaps, thereby avoiding such false positives (see § 7.2.5).

7.2.5 PCIe Contention (Mixtral 8×7B). We induce PCIe contention by launching RDMA loopback traffic on one RNIC, which halves the normal sending bandwidth. Pulse identifies this communication straggler, while SOTA methods misdiagnose other normal ranks. We train the Mixtral 8×7B model with 8-way TP, 4-way DP on attention layers, and 8-way TP, 4-way EP on expert layers. Figures 15 and 17 show the transmission rates of each P2P operator and each rank within one EP group, respectively. Although lower-rate P2P operators appear on rank 3 (red ellipses) and rank 1,

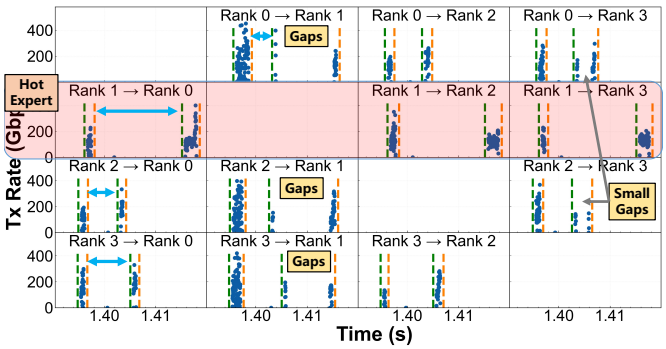

Figure 14. EP group Tx rate in expert imbalance case

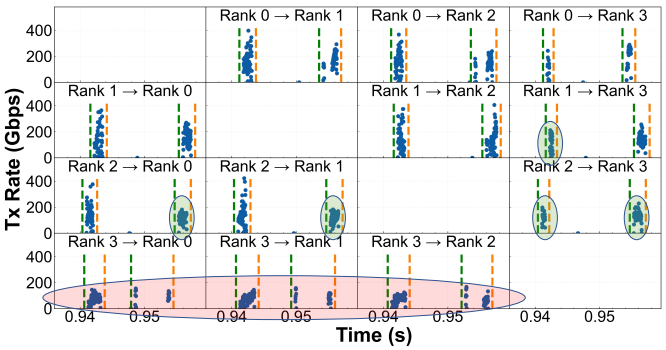

Figure 15. EP group Tx rate in PCIe contention (MoE) case

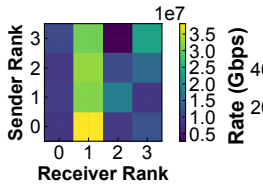

Figure 16. Imbalance routing.

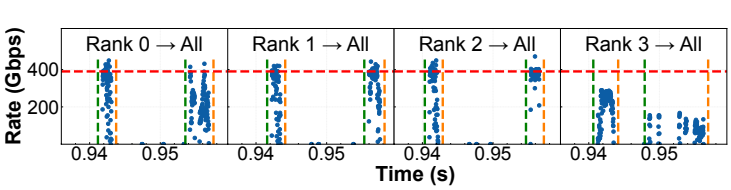

Figure 17. Rank-level Tx rate in PCIe contention (MoE) case

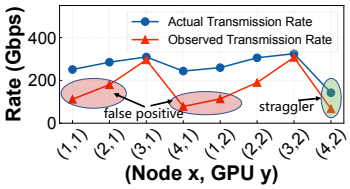

Figure 18. Metrics in PCIe contention (MoE) case.

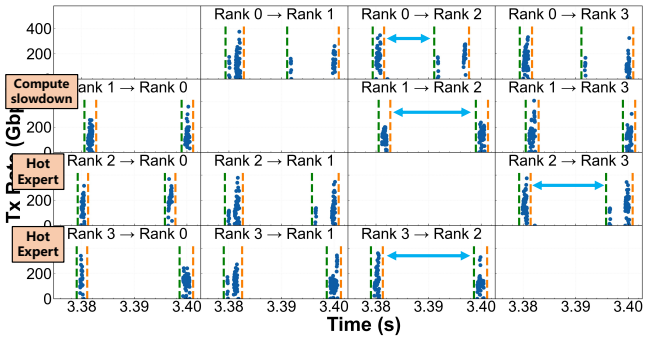

Figure 19. EP group Tx rate in GPU throttling (MoE) case

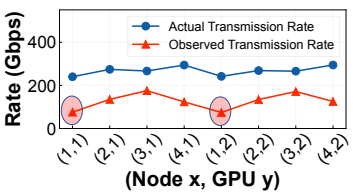

Figure 20. Metrics in GPU throttling (MoE) case.

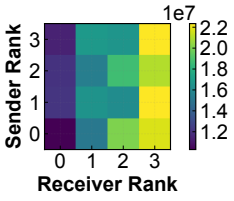

Figure 21. Heatmap in GPU throttling

2 (green ellipses) in Figure 15, rank-level rates in Figure 17 clearly show that only rank 3 is the straggler, highlighting the need for rank-level transmission metrics. Figure 18 shows the rank-level actual rates and observed rates (computed as data size divided by duration) for the last 10 recent `all-to-all` operations across two EP groups. Using actual rates, Pulse identifies Node 4 GPU 2 as the straggler. SOTA methods incur false positives as some ranks (*e.g.*, Node 4 GPU 1) exhibit lower observed rates due to their larger transmission gaps.

7.2.6 GPU Throttling (Mixtral 8×7B). We reduce one GPU's power limit from 700W to 350W using `nvidia-smi`. Both Pulse and SOTA methods localize stragglers caused by reduced power or hot experts, but SOTA methods incur false positives due to transmission gaps. We train the Mixtral 8×7B model using 8-way TP and 4-way DP for attention layers, and 8-way TP and 4-way EP for expert layers. As shown in Figure 19, Pulse and SOTA methods identify that rank 1, 2, and 3 are computation stragglers by their longer computation durations (blue arrows). These stragglers cause transmission gaps on rank 0, causing a lower rank-level observed rate on Node 1 GPU 1 (*i.e.*, rank 0) in Figure 20. SOTA methods misdiagnose this as a communication fail-slow. Furthermore, combining with the traffic map in Figure 21, we can infer that rank 2 and 3 are hotspots, while rank 1 slows down due to reduced computation efficiency as it receives fewer tokens than rank 2 but has a longer computation duration.

8 Evaluation

We first evaluate the diagnosis accuracy and latency of Pulse. We then evaluate its overhead, including impacts on RDMA traffic and training performance, and PCIe overhead. Finally, we evaluate the effectiveness of the operator segmentation and the parallelism strategy identification.

8.1 Diagnosis Accuracy and Latency

We conduct 57 anomaly localization experiments by injecting the failures listed in Table 3 under three scenarios: (1) training a 64-GPU GPT-2 70B model with 8-way TP, 4-way DP, and 2-way PP; (2) training a 32-GPU Mixtral 8×7B model, using 8-way TP and 4-way DP on attention layers, and 8-way TP and 4-way EP on expert layers; and (3) a 4-GPU neighbor-exchange collective. For performance-degrading failures, we vary the failure severity (*e.g.*, PCIe bandwidth reduced to

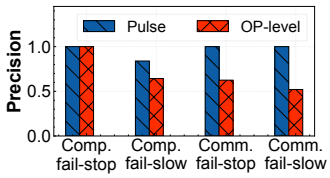

Figure 22. Precision.

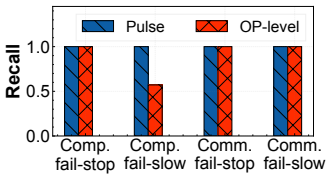

Figure 23. Recall.

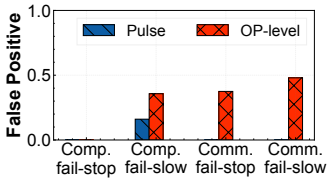

Figure 24. False positive rate.

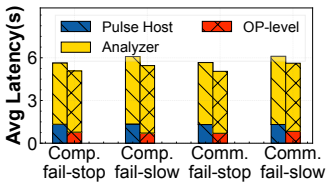

Figure 25. Diagnosis latency.

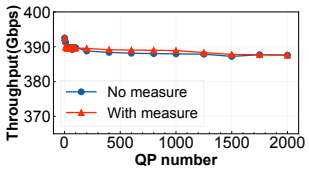

Figure 26. Throughput.

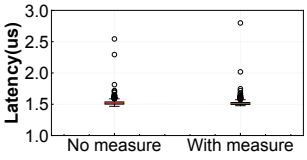

Figure 27. Latency.

50%, 60%, 70%, and 80% of the original). Appendix E details all failure-scenario combinations and severity levels.

Diagnosis accuracy. Pulse consistently outperforms SOTA OP-level monitoring methods [13, 54, 57] across all anomaly types, achieving over 90% precision and 100% recall, with false positives arising only in specific CPU-bottleneck scenarios. We define a true positive as localizing faulty nodes with the correct anomaly type. A false positive occurs when a normal node is reported as anomalous or a faulty node is identified with an incorrect anomaly type (*e.g.*, identify CPU bottlenecks as communication anomalies). Figures 22, 23, and 24 show the average precision, recall, and false positive rate across the four anomaly types. SOTA methods based on OP-level monitoring achieve an over 60% precision for communication anomalies and only 64% precision and 57% recall for computation fail-slow, as they cannot exclude gaps or pinpoint computation bottlenecks that occur during communication. Pulse achieves nearly 100% precision and recall, except for a 16% false positive rate under computation fail-slow (0% under other scenarios). These false positives mainly arise from CPU-bottleneck failures: when using built-in collectives (*e.g.*, all-reduce), a CPU bottleneck at a single node can introduce gaps across the entire group (see § 7.2.2). Pulse detects such anomalies at the group level while SOTA methods misdiagnose them as communication anomalies.

Diagnosis latency. Pulse incurs approximately 0.7s higher latency than SOTA methods due to the flow-operator association. Diagnosis latency comprises two main components: (1) *collection time* that each host spends collecting monitoring data of operators; and (2) *analysis time* that Analyzer takes to gather data from hosts, store the monitoring data, and localize the anomaly. Figure 25 shows the average diagnosis latency across all anomaly types. Pulse incurs approximately 0.7s higher latency than SOTA methods, primarily due to a longer collection time. Specifically, Pulse Host Agent must wait for the RNIC to upload measurement data at one-second intervals and then perform flow-operator association, whereas SOTA methods can record the overall execution time immediately after each operator completes.

8.2 Overhead of Pulse

RDMA traffic. We evaluate RDMA throughput and latency with measurement enabled and disabled, using `ib_send_bw` to generate up to 2000 RDMA flows, and `ib_send_lat` to collect 1000 latency samples. Figure 26 and 27 show the NIC throughput and latency distribution for both settings. Pulse supports concurrent measurement of 2000 flows per RNIC without performance degradation and has minimal impact on latency, with average latencies of 1.52 us and 1.53 us under the two settings, as its three-layer design places most measurement tasks off the packet-processing critical path.

LLM training. We evaluate the training performance under three conditions: without monitoring, with Pulse monitoring, and with log-based monitoring. We train three models, GPT-2 32B, GPT-2 70B, and Llama-70B for 100 iterations. For GPT-2 models, we apply 8-way TP, 2-way PP, and 4-way DP, while for the Llama model, we apply 2-way CP (Context Parallelism), 4-way TP, 2-way PP, and 4-way DP. The epoch length is 32 us. Figure 28 shows that the average iteration time is almost identical across all three configurations. Pulse introduces negligible overhead, as it conducts lightweight measurements on NICs and only intercepts APIs on host CPUs to collect argument information. We also evaluate the impact of epoch length on performance. Figure 29 shows the average iteration time of GPT-2 70B under two parallelism settings across varying epoch lengths. The results indicate that epoch length has little impact on training performance.

PCIe overhead. We evaluate the PCIe overhead incurred when Pulse Host Agent periodically pulls measurement data from Pulse NIC Agent over PCIe. We monitor 100 training iterations of GPT-2 70B and calculate the communication volume between the two agents. The PCIe overhead is estimated by dividing this volume by the total duration. We test on two parallelism settings and varying epoch lengths. Figure 30 shows that the maximum PCIe bandwidth usage is approximately 0.3 MB/s, which is negligible, since flows in LLM training are sparse compared to other workloads.

8.3 Effectiveness of Pulse Host Agent Design

Operator segmentation algorithm. Figure 31 shows the operator segmentation results for one DP group, with operator boundaries marked by red dotted lines. Using communication volume alone is insufficient, as it only provides a lower bound and causes error accumulation over time, leading to increasing deviations from the ground truth. By combining time intervals with communication volume, our algorithm

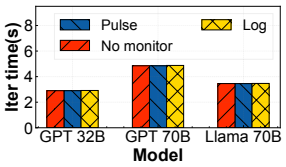

Figure 28. Avg. Iter. time with different models

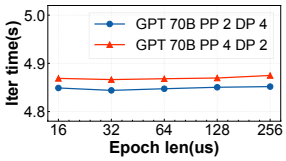

Figure 29. Avg. Iter. time vs. epoch lengths.

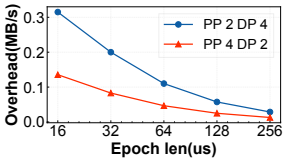

Figure 30. PCIe overhead vs. epoch lengths.

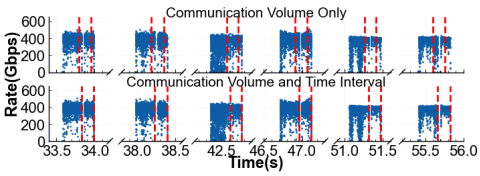

Figure 31. Operator segmentation results.

accurately identifies each operator. We manually validate the segmentation results in all evaluations and confirm that every communication operator is correctly identified.

Parallelism strategy identification. The classification accuracy of this algorithm remains 100% across all our cases. We further evaluate it on a 512-GPU simulation testbed using SimAI [53], testing various parallelism settings (DP, TP, PP, EP), where it consistently achieves full accuracy.

9 Discussion

Application to other SmartNIC architectures. Our measurement design can generalize to other SmartNICs that meet two requirements: ① visibility into packet transmissions and ② sufficient memory resources. For example, AMD Pensando [4] supports line-rate per-packet processing (①) and is equipped with on-board DDR (②). Similarly, Pensando's DDR incurs high access latency, and frequent accesses can degrade performance. Our aggregation layer mitigates this and can be adapted to Pensando in two ways: (1) Maintain small per-QP counters in in-pipeline SRAM and flush to DDR only when thresholds are reached. (2) Use verb-level aggregation by allowing only RDMA WRITE FIRST/LAST packets to update DDR. This has negligible impact on measurement accuracy for LLM training, as a typical write size is 1 MB, corresponding to roughly 20 us over a 400 Gbps network.

Scope of anomaly localization. Pulse can localize anomalous nodes for non-group anomalies. For group anomalies (*e.g.*, when the entire network degrades), Pulse can determine whether the source is computation or network-related by analyzing traffic measurements.

Other application scenarios of RDMA measurement. RDMA measurement applies to a range of scenarios, including distributed model inference [61] monitoring, congestion control bugs diagnosis [13], and collecting high-precision traffic data for complex communication patterns (*e.g.*, all-to-all [59]) to support simulation and performance modeling.

Future work. As scale-up networks play an increasingly important role, extending Pulse to support intra-node communication monitoring is a natural direction for future work, potentially enabled by combining eBPF-like GPU kernel profiling [18] and intra-node link performance counters.

10 Related Work

LLM training monitor. State-of-the-art monitoring systems for LLM training [11, 13, 22, 30, 44, 54, 57] primarily gather monitoring data from software logs and hardware counters. MegaScale [22] uses CUDA events to track the durations of critical code segments. Aegis [13] initially relies on training logs and hardware counters, later evolving to gather data from CCLs for monitoring. Similarly, Holmes [57] traces operator information in CCLs and focuses on straggler identification. GreyHound [54] combines function hooking with CUDA events to achieve non-intrusive monitoring. Different from these works, Pulse offers fine-grained monitoring through traffic measurement, enabling accurate anomaly localization in a non-intrusive way.

High-resolution measurement. State-of-the-art measurement systems typically work at millisecond granularity and are deployed on hosts or P4 switches [17, 32, 50, 56, 60]. uMon [60] proposes a WaveSketch to conduct memory-efficient microsecond-level measurements on hosts. Lumina [58] mirrors every packet on switches for in-depth analysis of NICs. Millisampler [15] measures microbursts at interface granularity. Different from these works, Pulse designs a three-layer measurement on mainstream RNICs, enabling per-QP microsecond-level RDMA measurement.

11 Conclusion

In this paper, we propose Pulse, a traffic-centric LLM training monitoring system. Pulse conducts microsecond-level RDMA traffic measurement with no loss of precision and transforms this low-layer data into measurements of collective communication operators in a non-intrusive way. This enables fine-grained monitoring of the LLM training process. Our evaluation demonstrates that Pulse can provide machine-level anomaly localization in an application-agnostic way with minimal impact on training performance overhead.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This research is supported by the National Natural Science Foundation of China under Grant Numbers 62325205, U25B2035, and 62502197, the Key Program of Natural Science Foundation of Jiangsu under grant No. BK20243053, the Basic Research Program of Jiangsu under Grant Number BK20251206, the Nanjing University-China Mobile Communications Group Co.,Ltd. Joint Institute. Qingkai Meng is the corresponding author.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Aliyun. 2025. Aliyun Web Services. https://www.aliyun.com.

[3] Amazon. 2025. Amazon Web Services. https://aws.amazon.com.

[4] AMD. 2025. AMD Pensando. https://www.amd.com/en/solutions/data-center/networking.html.

[5] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Liu, Jitu Padhye, Geoff Outhred, and Boon Thau Loo. 2017. Closing the network diagnostics gap with vigil. In *Proceedings of the SIGCOMM Posters and Demos*. 40–42.

[6] Azure. 2025. Microsoft Azure. https://azure.microsoft.com.

[7] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).

[8] Hyunseok Chang, Walid A Hanafy, Sarit Mukherjee, and Limin Wang. 2024. INSERT: In-Network Stateful End-to-End RDMA Telemetry. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 1061–1070.

[9] Xuzheng Chen, Jie Zhang, Ting Fu, Yifan Shen, Shu Ma, Kun Qian, Lingjun Zhu, Chao Shi, Yin Zhang, Ming Liu, et al. 2024. Demystifying datapath accelerator enhanced off-path smartnic. In *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*. IEEE, 1–12.

[10] Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066* (2024).

[11] Yangtao Deng, Xiang Shi, Zhuo Jiang, Xingjian Zhang, Lei Zhang, Zhang Zhang, Bo Li, Zuquan Song, Hang Zhu, Gaohong Liu, et al. 2025. Minder: Faulty Machine Detection for Large-scale Distributed Model Training. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*.

[12] Rui Ding, Xunpeng Liu, Shibo Yang, Qun Huang, Baoshu Xie, Ronghua Sun, Zhi Zhang, and Bolong Cui. 2024. RD-Probe: Scalable Monitoring With Sufficient Coverage In Complex Datacenter Networks. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 258–273.

[13] Jianbo Dong, Kun Qian, Pengcheng Zhang, Zhilong Zheng, Liang Chen, Fei Feng, Yikai Zhu, Gang Lu, Zhihui Ren, Xue Li, et al. 2025. Evolution of Aegis: Fault Diagnosis for AI Model Training Cloud Service in Production (Experience Track). In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*.

[14] Benjamin Fuhrer, Yuval Shpigelman, Chen Tessler, Shie Mannor, Gal Chechik, Eitan Zahavi, and Gal Dalal. 2023. Implementing reinforcement learning datacenter congestion control in NVIDIA NICs. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 331–343.

[15] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. 2022. A microscopic view of bursts, buffer contention, and loss in data centers. In *Proceedings of the 22nd ACM Internet Measurement Conference*. 567–580.

[16] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 139–152.

[17] Qun Huang, Haifeng Sun, Patrick PC Lee, Wei Bai, Feng Zhu, and Yungang Bao. 2020. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 404–421.

[18] Songlin Huang and Chenshu Wu. 2025. Neutrino: Fine-grained {GPU} Kernel Profiling via Programmable Probing. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*. 331–355.

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

[20] Sylvain Jeaugey. 2025. PAT: a new algorithm for all-gather and reduce-scatter operations at scale. *arXiv preprint arXiv:2506.20252* (2025).

[21] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of Experts. arXiv:2401.04088 [cs.LG] https://arxiv.org/abs/2401.04088

[22] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. 2024. {MegaScale}: Scaling large language model training to more than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 745–760.

[23] Xinhao Kong, Jingrong Chen, Wei Bai, Yechen Xu, Mahmoud Elhaddad, Shachar Raindel, Jitendra Padhye, Alvin R Lebeck, and Danyang Zhuo. 2023. Understanding {RDMA} microarchitecture resources for performance isolation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 31–48.

[24] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. 2022. Collie: Finding performance anomalies in {RDMA} subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 287–305.

[25] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).

[26] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High precision congestion control. In *Proceedings of the ACM special interest group on data communication*. 44–58.

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

[28] Kefei Liu, Zhuo Jiang, Jiao Zhang, Shixian Guo, Xuan Zhang, Yangyang Bai, Yongbin Dong, Feng Luo, Zhang Zhang, Lei Wang, et al. 2024. R-pingmesh: A service-aware roce network monitoring and diagnostic system. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 554–567.

[29] Kefei Liu, Zhuo Jiang, Jiao Zhang, Haoran Wei, Xiaolong Zhong, Lizhuang Tan, Tian Pan, and Tao Huang. 2023. Hostping: Diagnosing intra-host network bottlenecks in {RDMA} servers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 15–29.

[30] Qingkai Meng, Hao Zheng, Zhenhui Zhang, ChonLam Lao, Chengyuan Huang, Baojia Li, Ziyuan Zhu, Hao Lu, Weizhen Dang, Zitong Lin, Weifeng Zhang, Lingfeng Liu, Yuanyuan Gong, Chunzhi He, Xiaoyuan Hu, Yinben Xia, Xiang Li, Zekun He, Yachen Wang, Xianneng Zou, Kun Yang, Gianni Antichi, Guihai Chen, and Chen Tian. 2025. Astral: A Datacenter Infrastructure for Large Language Model Training at Scale. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) *(SIGCOMM '25)*. Association

for Computing Machinery, New York, NY, USA, 609–625. doi:10.1145/3718958.3750521

[31] David L Mills. 2002. Internet time synchronization: the network time protocol. *IEEE Transactions on communications* 39, 10 (2002), 1482–1493.

[32] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2016. Trumpet: Timely and Precise Triggers in Data Centers. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 129–143. doi:10.1145/2934872.2934879

[33] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM symposium on operating systems principles*. 1–15.

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

[35] NVIDIA. [n. d.]. NCCL GitHub Issue #578: Ring AllReduce performance discrepancy. https://github.com/NVIDIA/nccl/issues/578.

[36] Nvidia. 2019. NCCL Tree. https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/.

[37] Nvidia. 2025. Data-Path Accelerator Subsystem. https://docs.nvidia.com/doca/sdk/dpa+subsystem/index.html.

[38] Nvidia. 2025. DOCA Documentation v2.9.0. https://docs.nvidia.com/doca/archive/2-9-0/index.html.

[39] Nvidia. 2025. DOCA PCC. https://docs.nvidia.com/doca/sdk/doca+pcc/index.html.

[40] Nvidia. 2025. DPA L2 Reflector. https://docs.nvidia.com/doca/archive/doca-v2-5-0/nvidia+doca+dpa+l2+reflector+application+guide/index.html.

[41] Nvidia. 2025. NCCL. https://github.com/nvidia/nccl.

[42] Nvidia. 2025. NCCL Group calls. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/groups.html.

[43] Nvidia. 2025. NCCL Point-to-point communication. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/p2p.html.

[44] Nvidia. 2025. NCCL Profiler Plugin. https://github.com/NVIDIA/nccl/blob/master/ext-profiler/README.md.

[45] Pytorch. 2025. Pytorch NCCL backend. https://github.com/pytorch/pytorch/blob/viable/strict/1766582867/torch/csrc/cuda/nccl.cpp.

[46] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. 2024. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 691–706.

[47] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[48] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 3505–3506.

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

[50] Haifeng Sun, Jiaheng Li, Jintao He, Jie Gui, and Qun Huang. 2023. Omniwindow: A general and efficient window mechanism framework for network telemetry. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 867–880.

[51] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599* (2025).

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

[53] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, Yang Liu, Pengcheng Zhang, Kun Qian, Kunling He, Jiaqi Gao, Ennan Zhai, Dennis Cai, and Binzhang Fu. 2025. SimAI: Unifying Architecture Design and Performance Tuning for Large-Scale Large Language Model Training with Scalability and Precision. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 541–558. https://www.usenix.org/conference/nsdi25/presentation/wang-xizheng-simai

[54] Tianyuan Wu, Wei Wang, Yinghao Yu, Siran Yang, Wenchao Wu, Qinkai Duan, Guodong Yang, Jiamang Wang, Lin Qu, and Liping Zhang. 2025. {GREYHOUND}: Hunting {Fail-Slows} in {Hybrid-Parallel} Training at Scale. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. 731–747.

[55] Yifan Xiong, Yuting Jiang, Ziyue Yang, Lei Qu, Guoshuai Zhao, Shuguang Liu, Dong Zhong, Boris Pinzur, Jie Zhang, Yang Wang, et al. 2024. {SuperBench}: Improving Cloud {AI} Infrastructure Reliability with Proactive Validation. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 835–850.

[56] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 561–575.

[57] Zhiyi Yao, Pengbo Hu, CongCong Miao, Xuya Jia, Zuning Liang, Yuedong Xu, Chunzhia He, Hao Lu, Mingzhuo Chen, Xiang Li, Zekun He, Yachen Wang, Xianneng Zou, and Junchen Jiang. 2025. Holmes: Localizing Irregularities in LLM Training with Mega-scale GPU Clusters. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*.

[58] Zhuolong Yu, Bowen Su, Wei Bai, Shachar Raindel, Vladimir Braverman, and Xin Jin. 2023. Understanding the micro-behaviors of hardware offloaded network stacks with lumina. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 1074–1087.

[59] Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. 2025. DeepEP: an efficient expert-parallel communication library. https://github.com/deepseek-ai/DeepEP.

[60] Hao Zheng, Chengyuan Huang, Xiangyu Han, Jiaqi Zheng, Xiaoliang Wang, Chen Tian, Wanchun Dou, and Guihai Chen. 2024. μMon: Empowering Microsecond-level Network Monitoring with Wavelets. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 274–290.

[61] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 193–210.

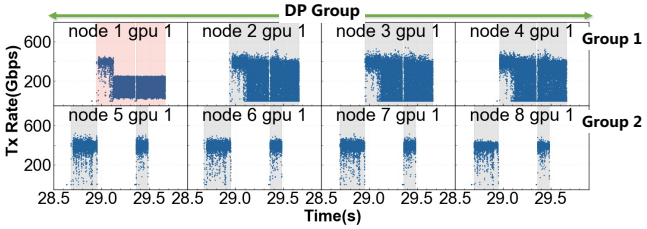

Figure 32. DP group rate in PCIe downgrade case.

A Implementation of Pulse NIC Agent

We implement the Pulse NIC Agent on Nvidia Bluefield-3, utilizing DOCA 2.9.0 [38]. For the aggregation layer, we leverage the DOCA PCC library [39] to access on-NIC events. The library supports various events, including transmission events, ACK events, NACK events, and CNP events. For our use case, we only register transmission events. Each time the NIC transmits 4KB of data for a given flow, it triggers an event in the measurement layer. The measurement layer is implemented on the DPA [37], an embedded processor within the Bluefield-3. DPA utilizes the C programming model, which simplifies the development process. The measurement function is embedded within the event handler. When DPA receives an event, it activates the handler and executes it on one of the available threads, where the necessary measurement is performed. The collection layer is deployed on the host CPU. We use the mailbox mechanism provided by the PCC library to implement the interaction between the collection layer and the measurement layer. Specifically, the host CPU periodically polls the DPA for flow aging and to gather the measurement data. This communication is ultimately conducted via PCIe.

B Expected Communication Volume

All-Reduce. Let s denote the group size, *count* denote the size of data elements that each device contributes and ultimately receives from the reduction. In the ring algorithm, each node sends to its successor, and the expected communication volume v is:

$$v = 2 * count * (s-1)/s$$

In the tree algorithm, each node communicates with its parent and children, and the expected communication volume is c. NCCL also supports constructing two independent trees, in which case the data is evenly split across the two trees.

All-Gather. Let s denote the group size, *sendcount* denote the size of data elements that each rank sends to all other ranks. In the ring algorithm, each node sends to its successor, and the expected communication volume v is:

$$v = sendcount * s * (s-1)/s = sendcount * (s-1)$$

Reduce-Scatter. Let s denote the group size, *recvcount* denote the size of data elements each rank ultimately receives from the reduction. In the ring algorithm, each node sends to its successor, and the expected communication volume v is:

$$v = recvcount * s * (s-1)/s = recvcount * (s-1)$$

C Performance Impact Analysis of Pulse Host Agent

The potential performance impact on the training arises primarily from two sources: function interception and all-gather operations. The former is lightweight [54], as we merely extract certain parameter information. For all-gather operations, the overhead primarily depends on the number of RDMA flows and the number of groups associated with each GPU, which generally do not exceed the order of hundreds [46]. Therefore, even in a cluster with tens of thousands of GPUs, the bandwidth overhead per GPU remains on the order of O(MB) ($k \times 10^2 \times 10^5$ bytes, where k is a constant). Moreover, these operations use management NICs for data transfer and occur only once, thereby imposing minimal impact on training performance.

D Anomaly Localization Case

D.1 PCIe Downgrade, PCIe Contention (GPT-2)

We train the GPT-2 70B model with 8-way TP, 4-way ZeRO-2 DP, and 2-way PP. For PCIe downgrade, we use `setpci` to reduce the PCIe link to half of its original bandwidth. For PCIe contention, we generate continuous loopback RDMA traffic to maintain the PCIe bandwidth at approximately half capacity. As shown in Figure 32, Node 1 GPU 1 exhibits a lower transmission rate than the others. Pulse localizes this GPU based on its extended actual communication time, as shown in Figure 33. SOTA methods can only observe a prolonged duration in Group 1 and fail to achieve machine-level localization.

D.2 NIC-GPU Cross NUMA Binding (GPT-2)

We train GPT-2 32B model with 4-way TP and 4-way ZeRO-2 DP across four nodes, utilizing four of the eight GPUs available on each node. On Node 1, the four GPUs are bound to four RNICs that are distributed across different NUMA nodes. Cross-NUMA data transfer incurs higher latency and limited bandwidth, resulting in degraded communication performance on Node 1. We observe that this performance degradation does not manifest as consistently lower transmission rates, as seen in network congestion or PCIe degradation, but rather as significant fluctuations in the transmission rate, as shown in Figure 34. While it is difficult to identify the straggler from the rate curve due to these fluctuations, Pulse successfully pinpoints the straggler by utilizing the actual communication time, as shown in Figure 35.

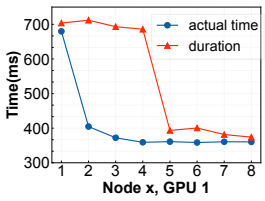

Figure 33. Actual communication time and duration in PCIe downgrade case.

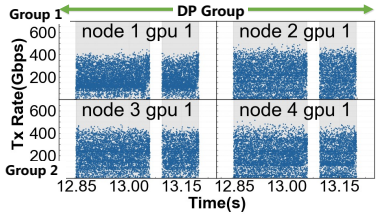

Figure 34. DP group rate in GPU-NIC cross NUMA binding

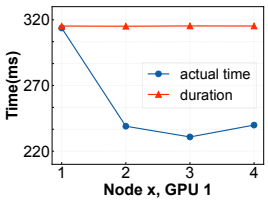

Figure 35. Actual communication time and duration in GPU-NIC cross NUMA binding case.

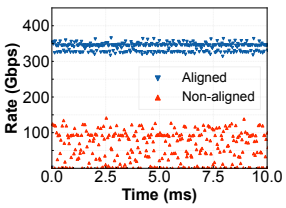

Figure 36. Impact of memory alignment

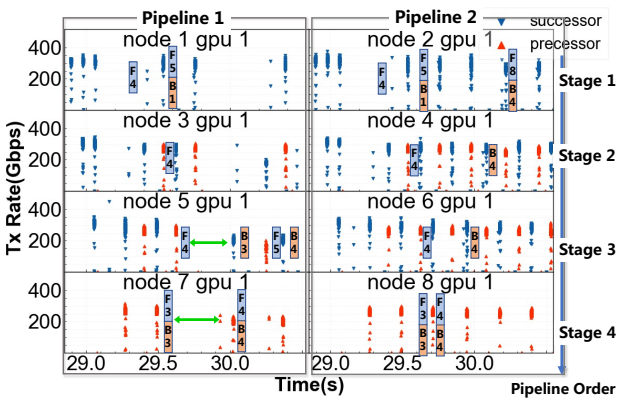

Figure 37. PP group Tx rate in GPU throttling case

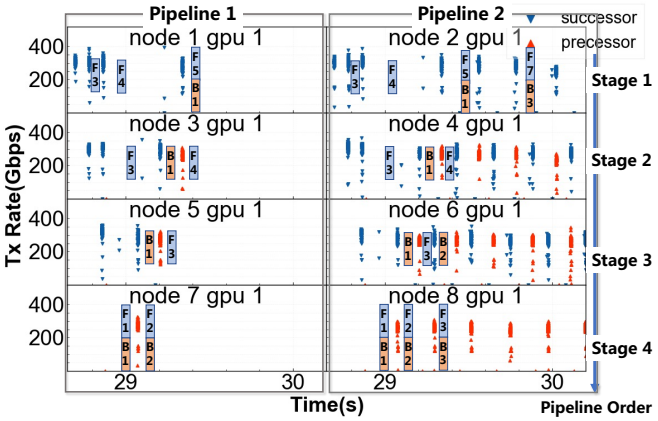

Figure 38. PP group Tx rate in GPU execution error case.

D.3 CPU Throttling (GPT-2)

We set the CPU's `scaling_governor` to `powersave` mode to reduce its operating frequency. Similar to the CPU contention case, this primarily impacts the proxy thread; therefore, we omit the detailed analysis.

D.4 GPU Throttling (GPT-2)

We use the `nvidia-smi` to reduce the power limit of a GPU from 700W to 350W. Both Pulse and SOTA methods successfully identify the anomalous node. We train the GPT-2 70B model with 8-way TP, 2-way ZeRO-2 DP, and 4-way PP. Figure 37 shows the transmission rates for two of the PP groups (*i.e.*, Pipeline 1 and Pipeline 2). Pulse checks the timestamp of each operation in chronological order and finds that the third P2P operator on Node 7, GPU 1 is the first to occur later than those in other groups. This delay can also be caused by Node 5 GPU 1, since Node 7 GPU 1 must wait for Node 5 GPU 1 to complete the forward pass of the fourth microbatch (F4) before it can send the backward result of the third microbatch (B3) to Node 5 GPU 1. Therefore, Pulse marks both nodes as anomalous. This case only uses OP-level data for localization, so existing SOTA methods can also localize these two devices.

Further analysis of other PP groups on Nodes 1, 3, 5, and 7 shows patterns similar to Figure 37, even though these GPUs operate normally. This is because GPUs within the same TP group synchronize during computation. As a result, while analyzing the monitoring data of PP groups can identify the straggler machine, localizing the specific straggler GPU requires analysis at the TP-group level. Since Pulse monitors only inter-node communication, and TP groups are typically confined to a single node, Pulse's localization granularity for computation anomalies is limited to the machine level.

D.5 GPU Execution Error (GPT-2)

We train GPT-2 70B model with 8-way TP, 4-way ZeRO-2 DP, and 2-way PP. We induce a GPU execution error by accessing an invalid address. Figure 38 presents the transmission rate details for two of the PP groups. Pulse detects that the third send-forward operator on Node 5, GPU 1 (*i.e.*, Stage 3 of Pipeline 1) is the first to miss, which can be caused by Node 5 GPU 1 during the computation of F3 or Node 7 GPU 1 during the computation of F2 and B2. Therefore, Pulse Analyzer marks both nodes as anomalous. This case utilizes only OP-level data for localization, so existing SOTA methods can also localize these two devices.

D.6 CUDA Execution Error (GPT-2)

We induce a CUDA out-of-memory error by allocating excessive memory on a GPU during training. This case has a

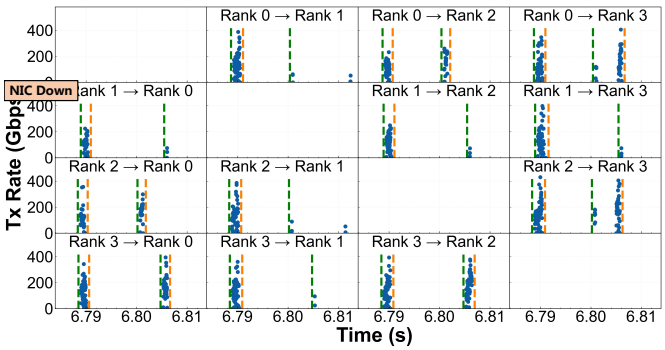

Figure 39. EP group Tx rate in NIC Down (MoE) case

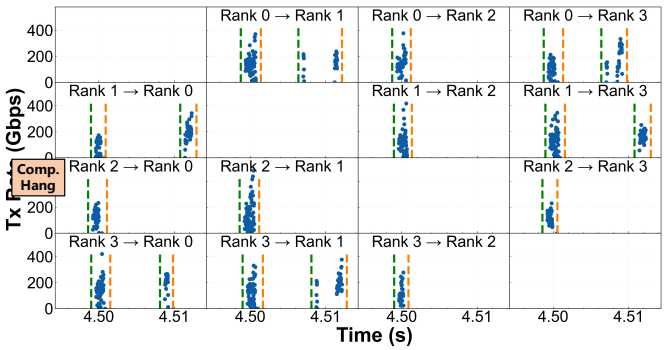

Figure 40. EP group Tx rate in GPU execution error (MoE)

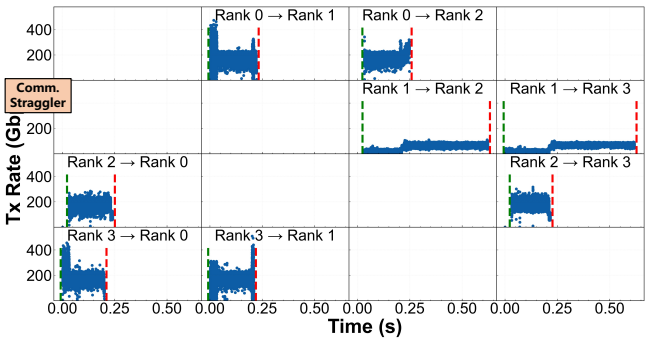

Figure 41. P2P Tx rate in neighbor-exchange case

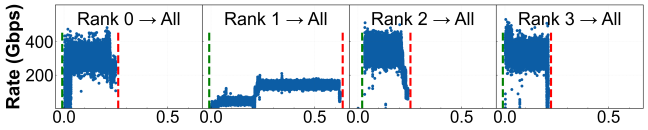

Figure 42. Rank-level Tx rate in neighbor-exchange case

similar behavior to the GPU execution error case. Therefore, we omit the detailed analysis.

D.7 Memory Alignment

This case originates from our development of an RDMA traffic generator for LLM training, where we observe that memory alignment has a significant impact on RDMA performance. As illustrated in Figure 36, misaligned memory results in significantly lower throughput compared to aligned memory. In practice, this degradation will extend the overall duration of the communication operation, and Pulse can effectively localize this straggler by utilizing the actual communication time.

D.8 NIC Down (Mixtral 8×7B)

We disable one NIC during training. Both Pulse and SOTA methods correctly localize the faulty node. We train the Mixtral 8×7B model using 8-way TP and 4-way DP for attention layers, and 8-way TP and 4-way EP for expert layers. Figure 39 shows the transmission rates of each P2P operator for two `all-to-all` operations within one EP group. Unlike built-in collective primitives, the P2P operators in the custom collective execute independently. As a result, both Pulse and SOTA methods observe all incomplete P2P operators involving rank 2 during the second `all-to-all`, and correctly identify rank 2 as the faulty node.

D.9 GPU Execution Error (Mixtral 8×7B)

We induce a GPU execution error by accessing an invalid address. Both Pulse and SOTA methods correctly localize the faulty node. We train the Mixtral 8×7B model using 8-way TP and 4-way DP for attention layers, and 8-way TP and 4-way EP for expert layers. Figure 40 shows the transmission rates of each P2P operator for two `all-to-all` operations within one EP group. During the second `all-to-all`, all P2P operators involving rank 2 lack a start timestamp (missing a green dotted line), indicating a computation hang on rank 2.

D.10 PCIe Contention (neighbor-exchange)

We induce PCIe contention by launching RDMA loopback traffic on one RNIC, throttling its sending bandwidth to 60% of its original. Both Pulse and SOTA methods can accurately identify this straggler. We run a 4-GPU neighbor-exchange collective in a two-dimensional space with one GPU per node, ensuring that all communication is carried out over the RDMA network. Each rank sends 1 GB data to its neighbors. Figure 41 and 42 show the transmission rates of each P2P operator and each rank, respectively. Since there is no computation workload, communication kernels start nearly simultaneously with no gaps. Consequently, both Pulse and SOTA methods identify rank 1 as the straggler based on the rank-level transmission rates.

D.11 PCIe Downgrade (all-to-one)

We use `setpci` to throttle the PCIe link to 70% of its original bandwidth. Both Pulse and SOTA methods can accurately identify this straggler. We run a 4-GPU all-to-one collective and set rank 0 as the root rank. Each rank sends 1 GB data to the root. Figure 43 shows the transmission rates of each P2P operator. Similar to § D.10, as all communication kernels start nearly simultaneously without gaps, both Pulse and the SOTA method correctly identify rank 1 as the straggler due to its lower transmission rates.

Table 4. The detailed failure-scenario combinations and severity level in § 8.1.

Type	Failure injection	Test scenarios
Comp. fail-stop	GPU execution error, invaild address access	GPT-2, Mixtral 8×7B
	CUDA execution error, allocate excessive memory	GPT-2, Mixtral 8×7B
Comp. fail-slow	GPU throttling, with power limit set to 350W, 300W, 250W, 200W	GPT-2, Mixtral 8×7B
	CPU throttling, adjust `scaling_governor` to `powersave`	GPT-2, Mixtral 8×7B
	CPU contention, using `stress` to occupy 90%, 100% of cores	GPT-2, Mixtral 8×7B
Comm. fail-stop	NIC down, disable one RNIC	GPT-2, Mixtral 8×7B neighbor-exchange
Comm. fail-slow	PCIe downgrade, using `setpci` to throttle the PCIe link to 50%, 40%, 30%, 20% of its original bandwidth	GPT-2, Mixtral 8×7B neighbor-exchange
	PCIe contention, launching RDMA loopback traffic to occupy 50%, 60%, 70%, 80% of PCIe bandwidth	GPT-2, Mixtral 8×7B neighbor-exchange
	Network congestion, launching background RDMA traffic to reduce the bandwidth of one path to 50%, 60%, 70%, 80%	GPT-2, Mixtral 8×7B neighbor-exchange
	NIC-GPU cross NUMA binding	GPT-2

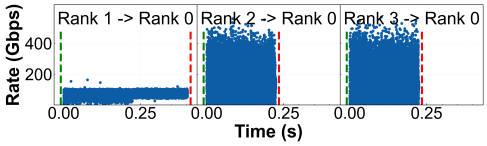

Figure 43. P2P Tx rate in all-to-one case

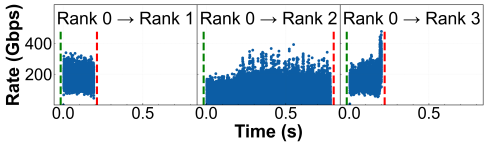

Figure 44. P2P Tx rate in one-to-all case

D.12 Network Congestion (one-to-all)

We induce congestion by injecting background traffic that halves the bandwidth of a network path. Both Pulse and SOTA methods can accurately identify the communication straggler. We run a 4-GPU one-to-all collective with rank 0 as the root rank. The root sends 1 GB data to the other ranks. Figure 44 shows the transmission rates of each P2P operator. Similarly, as all communication kernels start nearly simultaneously without gaps, both Pulse and the SOTA method correctly identify rank 1 as the straggler due to its lower transmission rates.

E Failure-Scenario Combinations and Severity Level

In § 8.1, we evaluate the diagnosis accuracy of Pulse under various failure types and test scenarios. Table 4 summarizes the evaluated failure–scenario combinations and the severity levels of performance-degrading failures. For neighbor-exchange, we use one GPU per node to ensure that all communication is carried out over RDMA.