Holmes: Localizing Irregularities in LLM Training with Mega-scale GPU Clusters

Zhiyi Yao and Pengbo Hu, *Fudan University and Tencent;* Congcong Miao and Xuya Jia, *Tencent;* Zuning Liang and Yuedong Xu, *Fudan University;* Chunzhi He, Hao Lu, Mingzhuo Chen, Xiang Li, Zekun He, Yachen Wang, and Xianneng Zou, *Tencent;* Junchen Jiang, *University of Chicago*

https://www.usenix.org/conference/nsdi25/presentation/yao

This paper is included in the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation.

April 28–30, 2025 • Philadelphia, PA, USA

978-1-939133-46-5

Open access to the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

Holmes: Localizing Irregularities in LLM Training with Mega-scale GPU Clusters

Zhiyi Yao[1,2*], Pengbo Hu[1,2*], Congcong Miao[2*], Xuya Jia[2], Zuning Liang[1], Yuedong Xu[1§], Chunzhi He[2], Hao Lu[2], Mingzhuo Chen[2], Xiang Li [2], Zekun He[2], Yachen Wang [2], Xianneng Zou[2], Juncheng Jiang[3]
[1]*Fudan University,* [2]*Tencent,* [3]*University of Chicago*

Abstract

Training Large Language Models (LLMs) on large-scale GPU clusters requires numerous iterations over several months. Existing works mainly focus on addressing failures that interrupt the iterative training process to improve the utilization of GPU clusters. However, our large-scale measurements over tens of thousands of GPUs show that the training process exhibits an unstable state with some *irregular* iterations taking even more than twice the time of a normal iteration. Surprisingly, we find that these irregular iterations greatly extend the time of LLM training, which is even more severe than the impact of failures. Meanwhile, the irregular phenomenon is silent, making it challenging to be accurately localized. In this paper, we propose a first-of-its-kind system called Holmes, leveraging communication operators to accurately localize these irregularities in real-time. The core of Holmes's approach is to employ an enhanced abnormal operator detection model and a novel communication operator graph to perform efficient irregularity localization. Furthermore, Holmes conducts cross-iteration analysis to improve localization accuracy. We evaluate Holmes using large-scale trace-driven simulations and a production-level prototype. Large-scale simulation results demonstrate that Holmes achieves irregularity localization accuracy of 97.21%. Production-level prototype evaluation results show Holmes can localize irregularity within 30.3 seconds, achieving a speedup of 6.52× as compared to traditional approaches.

1 Introduction

Large language models (LLMs) [7] are propelling the rapid evolution of efficient and reliable AI infrastructure. Since the model size explosively increased to hundreds of billions of parameters such as GPT-4 [42] and PaLM [10], the training clusters have scaled out to encompass thousands to even 10,000 GPUs [25]. Training LLMs over large-scale GPU clusters always requires hundreds of thousands of iterations. Each *iteration* is a cycle of parameter update and synchronization, consisting of a mixture of massive compute- and communication- intensive operators. Executing these iterations in the GPU cluster can last for months, making investment in LLM training entail significant costs. Efficient utilization of expensive GPUs is crucial since even small improvements can save millions of dollars at such a scale [5].

Unfortunately, LLM training is *fragile* in the sense that even a small failure in the GPU clusters can explicitly interrupt the training process. These failures will finally extend the training time, resulting in a considerable waste of hardware resources. Therefore, there are several works focusing on failure detection and diagnosis to improve the system reliability [18, 21, 23, 32, 35, 43, 54, 62]. In these works, once a failure is diagnosed, the anomalous device is isolated to recover the training process.

Generally, the time of each iteration in the LLM training is expected to be stable and regular because the computing and communication patterns are predetermined and repeated in each iteration. However, our production measurements over tens of thousands of GPUs indicate that the training process exhibits an unstable state with the time of each iteration varying. Some *irregular* iterations take even more than twice the time as a normal iteration. Here, the term "*irregularity*" describes the time of an iteration exceeding the reference value by a certain margin, yet without interrupting the training process. We then accumulate additional training time introduced by irregularity. Surprisingly, we observe that the additional time introduced by the irregularity is even more severe than the impact of failures. Our manual post hoc analyses show that the root causes of these irregularities encompass anomalous GPUs, network link issues, unstable switches, etc. (§ 3.1).

However, efficient irregularity localization in a large GPU cluster is very challenging. Firstly, these irregularities are *silent* and do not interrupt the training process, making traditional failure diagnosis approaches [21, 43, 54, 62] relying on error logs ineffective. As there will be no failure logs during the irregular iteration, traditional systems will even not react

*The authors contributed equally to the paper.
§Corresponding author: Yuedong Xu (ydxu@fudan.edu.cn).

to begin the diagnosis process. Secondly, the anomaly in the device causing longer training iteration will propagate to impact other GPUs. Specifically, since there are thousands of collective communications across GPUs in the cluster, once an anomaly occurs in a device, all the GPUs connected to this device will be impacted to exhibit a longer training time in one iteration. When there are a lot of irregular iteration logs among these GPUs, it is hard to identify the anomalous device. Thirdly, the irregularity localization approach should be accurate. When the normal device is mistakenly identified as an anomaly, the isolation process of the device will lead to the waste of precious computing resources.

Holmes architecture. To address the aforementioned issues, we present a first-of-its-kind system, called Holmes, to accurately localize these irregularities for LLM training on mega-scale GPU clusters. Holmes is a lightweight approach that only relies on the data from the communication operator, thereby introducing little overhead during the LLM training. Specifically, to explicitly capture the silent irregularity, we introduce an enhanced abnormal operator detection model. We first leverage a machine learning method (i.e., random forest) to analyze communication tracing logs to detect abnormal communication operators. As the time distribution of iteration may shift over a long-term training process, we further introduce an auto-tuning approach for the machine learning method to adapt to the changes (§ 4.2). To identify the anomalous device using a set of irregular iteration logs from GPUs, we build a novel communication operator graph and design a combination of breadth-first and depth-first search methods to localize the irregularity in the graph. Note that Holmes only collects logs from these GPUs which are traversed by the search methods to reduce collected data volume and localization time (§ 4.3). To further improve the localization accuracy, we conduct cross-iteration analysis among all the devices including GPUs, links, and switches in the training clusters, and utilize Maximum A Posteriori estimation to find the most possible anomalous devices (§ 4.4).

Contributions. Our main contributions are summarized as follows:

- We identify the irregularity phenomenon and observe that these irregularities greatly extend the time of LLM training, which is even more severe than the impact of failures. To the best of our knowledge, we are the first to study the irregularity phenomenon in large-scale LLM training (§ 3).
- We design Holmes that leverages communication operators to localize irregularity over a 10,000 GPU cluster in real-time. Holmes constructs a communication operator graph combined with a domain-specific search method (§ 4).
- Large-scale simulation results demonstrate that Holmes achieves an overall localization accuracy of 97.21%. Production-level prototype evaluation results show that Holmes can localize irregularity within 30.3 seconds, which is 6.52× faster than the baseline (§ 5).

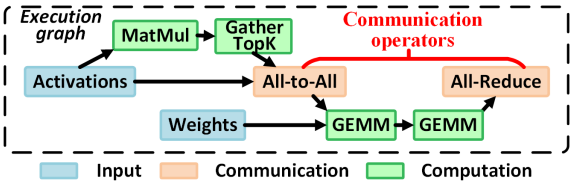


Figure 1: Compute- and communication- operators.

2 Background

Training Large Language Models (LLMs) relies on a large number of GPU machines and auxiliary resources to accelerate the execution of hundreds of thousands of iterations [21,60]. Each iteration is a cycle of parameter update and synchronization, consisting of a mixture of massive compute- and communication- intensive operators. These operators are repetitively executed following a fixed order, which can be represented in the form of an execution graph. Figure 1 depicts an example execution graph of a typical LLM training, where `Activations` and `Weights` are the input for the operators. The computation operators such as `MatMul` are executed on a single GPU without interference with other training nodes. In contrast, communication operators such as `All-to-All` involve a set of GPUs and will be executed only after all the GPUs have completed the execution of their respective dependent operators. Each GPU has its own execution graph and executes independently, and the communication operators will conduct synchronization among GPUs.

Communication operators. The communication operators serve for various parallel training strategies, which aim to better utilize GPUs and accelerate the training. These communication operators can be classified into two categories [37]: collective communication (CC) and point-to-point (P2P) communication. Collective communication including `All-Reduce`, `All-Gather`, `All-to-All` and `Reduce-Scatter`, is widely used in strategies like data parallelism (DP) [11,46], tensor parallelism (TP) [41,51] and expert parallelism (EP) [14,28]. During the training process, collective communication brings tight synchronization to massive GPUs involved, as it forces all the GPUs to initiate data transmission simultaneously. Different from collective communication, P2P communications like `Send` and `Recv`, do not need synchronization among massive GPUs. They are adopted in the pipeline parallelism (PP) [22,40], used to transmit the activations between the sub-models. Figure 2 depicts an example of LLM distributed training. The black arrows represent the P2P communication of GPUs, which communication only involves one sender and one receiver. Bidirectional arrows are collective communication, in which GPUs involved transmit data to each other.

3 Observation & Motivation

In this section, we first describe the overall performance degradation caused by irregularities in our production cluster and classify them into three categories (§ 3.1). We then elaborate

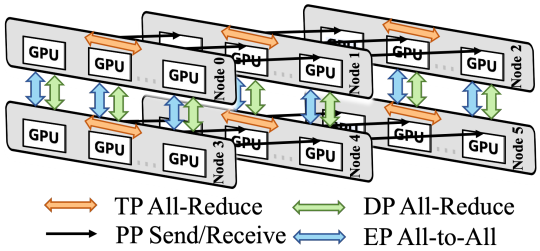


Figure 2: Communication of LLM distributed training.

on the challenges of localizing the anomalous devices, which will cause the irregularity (§ 3.2).

3.1 Irregularity in LLM Training

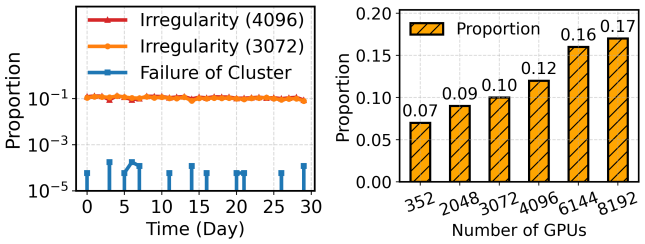
We define a novel metric to measure the irregularity in LLM training as below.

Definition 1 δ−*Irregularity refers to the phenomenon where training continues* ***uninterrupted****, but the duration of an iteration, i.e. time of every training step, is* δ *times longer than the average value within a given time window.*

The parameter δ (where $\delta > 1$) is tunable to account for subjective assessments of performance irregularities (by default δ is set to 1.1). The production cluster in our study comprises over 10,000 NVIDIA H800 GPUs. We keep tracing LLM training using a total number of GPUs ranging from 352 to 8192 in three months. The models of LLM training include LLaMA [56], LLaMA2 [57] and GPT3 [7] which vary in size. Their parallel training strategies employ hybrid parallelism incorporating tensor- [41], data- [11, 46], pipeline- [22, 40], sequence- [27], and expert- parallelism [28].

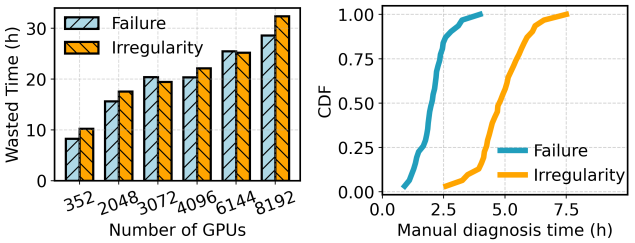
Irregularity occurs frequently. In Figure 3(a), we compare the numbers of irregularities and failures during the training of LLaMA+MoE using 3072 and 4096 GPUs for a month, respectively. The frequency is defined as the irregular iterations compared to the total iterations per day. On average, irregularities occur in 778 and 1047 iterations daily, in contrast to 6646 and 10165 total iterations per day. No more than three failures occur in one day throughout the entire training process, and only 40% of the days in a month encounter failure with a failure probability of less than 10^{-3}. It is evident that the frequency of failures is several orders of magnitude lower than that of irregularities. In Figure 3(b), we perform a statistical analysis of the frequencies of irregularities across various training. The frequency of irregularities increases as more GPUs are used, since more devices in training can lead to more potentially anomalous devices. Specifically, when training with 8192 GPUs, the irregularity frequency is 0.17, nearly three times that of training with 352 GPUs. More details and analysis of Figure 3(b) can be referred to appendix A.

Irregularity extends the training time. We highlight the *time wastage* caused by potential irregularities compared to training failures for a month in Figure 4(a). The wasted time due to failures is defined as the time interval between the



(a) Frequency across time (b) Frequency across training

Figure 3: Frequency of irregularity.



(a) Wasted time of irregularity (b) CDF of irregularity diagnosis time

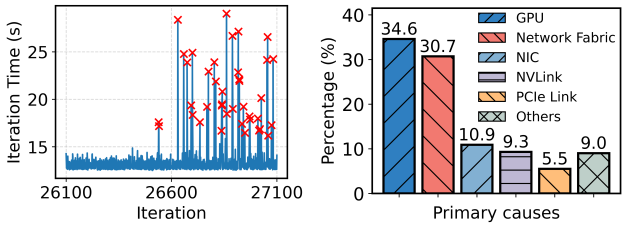
Figure 4: Impact of irregularity.

pause and resumption of training. As reported in [21], a training failure state usually lasts no more than three hours. The wasted time due to irregularities is calculated as the difference between the total actual training time and the hypothetical training time, where each irregular iteration is replaced with the time it would have taken without irregularities. An observation is that the time wastage of irregularities is comparable to or even greater than that of failures, with wasted time increasing as the number of GPUs scales up. For instance, when training with 8192 GPUs, the wasted time due to irregularities is as long as 32.38 hours, leading to a significant wastage of valuable GPU resources. This motivates us to explore the key behaviors of irregularities and identify the anomalous devices that induce such irregularities.

Irregularity localization is time-consuming. We then compare the manual irregularity localization time with the failure diagnosis time. As shown in Figure 4(b), around 86.2% of the irregularity localization time is larger than 4 hours. This is much longer than most of the time consumption on failure diagnosis, where approximately 84.7% of the diagnosis time is within 2.5 hours. This is attributed to that failure diagnosis has been extensively studied and standardized. In contrast, irregularity localization highly depends on human experience, leading to uncertainties in the time required for resolution. This motivates us to build an efficient system for automatic irregularity localization.

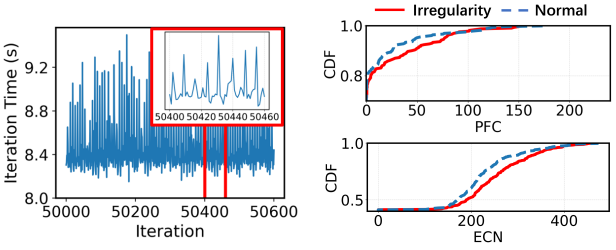
3.1.1 Abnormal Spikes

In this paper, we classify the irregularities into three categories according to their duration and frequency. The first type of irregularity, namely *abnormal spike*, occurs randomly and less frequently but significantly extends per-iteration training time. We study the pattern of abnormal spikes while training



(a) Iterations with abnormal spike (b) Causes of abnormal spike

Figure 5: Irregularity: abnormal spike.



(a) Iterations with fluctuation (b) CDF of PFC and ECN

Figure 6: Irregularity: performance fluctuation.

a LLaMA+MoE model using 3072 GPUs. The per-iteration times are shown in Figure 5(a), with a total of 1000 iterations recorded. Within this time window, abnormal spikes, marked with crosses, occur in 39 iterations. The iteration time with a spike is much longer than the normal value. For instance, at the 26803-rd iteration, the training time is 1.96× longer than the average. Over the entire training process for a month, the wasted time due to spikes amounts to 8.93 hours.

Manual diagnosis. When a *serious* abnormal spike occurs, our engineers will examine the states of training resources such as GPU and CPU utilization, ECN marks. Through comparison with empirical normal values, the suspicious anomalous devices will be isolated with further task-specific stress tests. In general, the entire process takes nearly 3-7 hours to resolve the abnormal spikes, while they continue to occur randomly until being resolved. We illustrate the primary causes of abnormal spikes and their respective percentages in Figure 5(b). Among them, GPU execution ranks first and is responsible for 34.6% of all abnormal spikes. The second major cause is the inter-machine networking issue, comprising network fabric problems, link flapping, network link failures, and NIC anomalies. Together, they account for 41.6% of the spikes, highlighting the severity of networking faults. This suggests that monitoring inter-machine transmission could be the key to identifying these faults.

3.1.2 Performance Fluctuation

Another type of irregularity is called *performance fluctuation* that is usually ignored due to their inconspicuous magnitudes of iteration time changes. Figure 6(a) shows a sequence of 600 iteration times of training with 4096 GPUs, where δ in this case is set to 1.05. The iteration times at the top may reach 9.45 seconds and those at the bottom are around 8.18 seconds, and the average iteration time is 8.40 seconds. As a result, there are 85 iterations with abnormal fluctuation among total 600 iterations. Even though the impact of performance fluctuation is smaller than abnormal spike on an individual iteration, the former is much more frequent than the latter (see appendix B). Overall, the training in Figure 6(a) is extended by performance fluctuation for 9.32 hours over a month, which is even more severe than abnormal spike. We also study the performance of network fabric when irregularities occur. Figure 6(b) shows the CDF of packets with ECN mark and pause frames within an iteration on all the ToR switches. The irregular iteration shows a clear shift to more ECN marks and PFCs, compared to the normal iteration. This phenomenon indicates that irregularities could affect the network traffic, which also inspires us to monitor irregularities through the network.

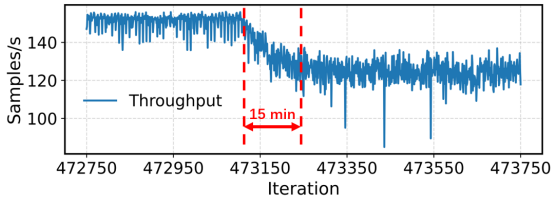


Figure 7: Irregularity: persistent degradation.

Manual diagnosis. The issues that could lead to fluctuations include link flapping and shared network fabrics for multiple jobs (e.g., periodic fluctuation occurs when jobs with different iteration times share spine or ToR switches [45]). Other causes that may also make the training process unstable with small and frequent fluctuations include load imbalance issues of MoE training [20, 31] and congestion [17]. However, this kind of analysis highly relies on the experience of engineers, leading to high and uncertain analytic costs.

3.1.3 Persistent Degradation

Different from the above irregularities, the third type of irregularity degrades the training efficiency persistently, which we call *persistent degradation*. Figure 7 shows the training throughput of 1000 iterations within a 4096 GPUs training. An observation is that the throughput decreases dramatically from around 150 samples per second to 125 samples per second, which means a persistent degradation occurs and results in a 16.72% training throughput degradation. Although the decreasing process of throughput lasts for only 15 minutes, the training will execute at a low efficiency if the irregularity is not addressed in time.

Manual diagnosis. Most of the root causes of persistent degradation are training-independent, such as GPU overheating [21] and abnormal CPU occupation of back-end services. Engineers have to analyze massive resource states and even operation logs of non-training systems to find the causes. For example, after several hours of analysis for the irregularity in Figure 7, we find that the CPU utilization in part of the machines is abnormally high. By tracking the operation logs,

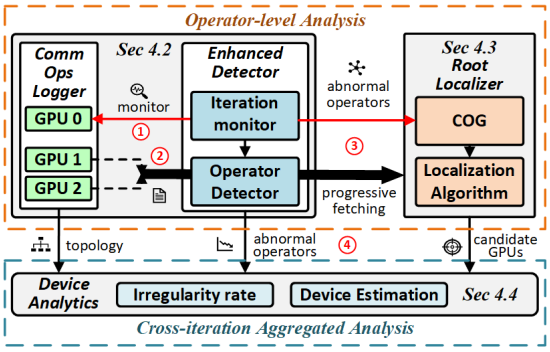


Figure 8: System overview.

the final root cause we identified is that the latest release of our network monitor system frequently scans all the NCCL ports abnormally, competing for the CPU cores. In our training cluster, each persistent degradation could result in 18.2% of training throughput decrease on average.

3.2 Challenges

Our goal is to design a system to accurately localize the above irregularities for LLM training. However, designing such a system in large GPU clusters is quite challenging.

Irregularity behaves silently. Since irregularities do not interrupt training, no error logs are generated. Network monitoring systems track real-time link throughput, which does not reflect the computing anomalies and the traffic interdependency across machines. GPU monitoring tools provide only coarse and noisy runtime states that limit their range of application. The training debugging system usually logs the execution process, while the logging granularity remains unresolved. For example, coarsely logging the iteration time [41,47] is insufficient. Fine-grained logging such as the torch profiler [3] may assist in localization. However, they record all the operators, which considerably slows down the training process [25].

Irregularities's impact propagates among GPUs. The irregularity tends to *propagate* its impact. For instance, if one GPU computes slowly before a DP `All-Reduce`, the other GPUs participating in the same `All-Reduce` must wait for it. As a result, similar irregular phenomena, i.e. extended iteration time, are observed across multiple GPUs, which can be confusing to diagnose. The irregularity can propagate *more than one hop*, further complicating its localization. Independently analyzing all the GPUs will introduce a lot of effort and is time-consuming since a lot of irregular training logs will be detected and hard to localize the anomalous device.

Misidentify the anomalous device is costly. The irregularity localization needs to be accurate, as isolating potentially anomalous devices is a costly countermeasure. While the irregularity occurs frequently, misidentifying anomalous devices causes the waste of precious computing resources, and leaves the irregularity unresolved. It is crucial to ensure high system accuracy while minimizing misidentification.

4 System Design

In this section, we first introduce the design principle and overall design of Holmes (§ 4.1). Then, we elaborate on the three key aspects of Holmes: enhanced abnormal operators detection model (§ 4.2), graph-based root causes localization (§ 4.3), and cross-iteration analysis (§ 4.4).

4.1 Overall Design

The goal of this paper is to accurately localize the anomalous devices that lead to the irregularity during LLM training.

Design principles. Our system design should follow these basic principles.

• *Accuracy.* High accuracy in pinpointing the irregularities and diagnose the root causes.

• *Real-time detection.* Capable of detecting and localizing irregularity in real time.

• *Low overhead.* Acceptable overhead brought by monitoring and localization, which has little impact on the LLM training.

• *Flexibility.* Capable of adapting to various parallel training strategies without any modifications.

System Overview. Following the above design goals, we design Holmes, a first-of-its-kind system to accurately localize the anomalous devices in a large-scale GPU cluster. Holmes is a lightweight approach that only relies on data from communication operators. More concretely, to explicitly observe silent irregularity, we propose an enhanced abnormal operator detection model. This model firstly leverages a communication operator logger to log the detailed process of communication operators (§ 4.2.1). Upon observing the irregularity, we propose an enhanced random forest model (RF) to analyze communication operator logs and capture the specific operator that causes the irregularity. Since the time distribution of iteration may shift over a long-term training and result in the RF performance degradation, we further introduce an auto-tuning approach to adapt to the changes (§ 4.2.2). To localize the anomalous device according to abnormal operators among GPUs, we construct a novel communication operator graph (COG), and design an effective algorithm that combines breadth-first search and depth-first search for traversing the collective communication and point-to-point operators respectively. Holmes only collects logs from the GPUs traversed by the search algorithm, minimizing the amount of data transmitted and localization time (§ 4.3). To ensure high localization accuracy, we introduce cross-iteration analysis among all devices within the training clusters, including GPUs, links, and switches, and employ Maximum A Posteriori estimation to identify the most possible anomalous device (§ 4.4).

Workflow. As shown in Figure 8, Holmes firstly embeds a communication operator logger for each GPU in the training cluster to record the data locally. Then we deploy our system modules including enhanced detector, root localizer and device analytics in an independent CPU server. ① The iteration

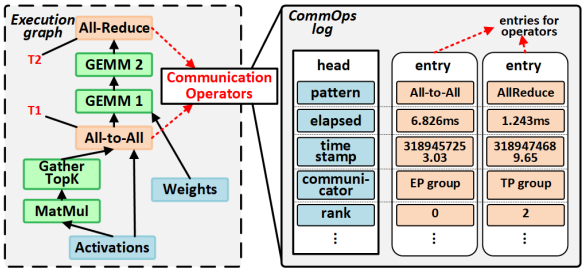
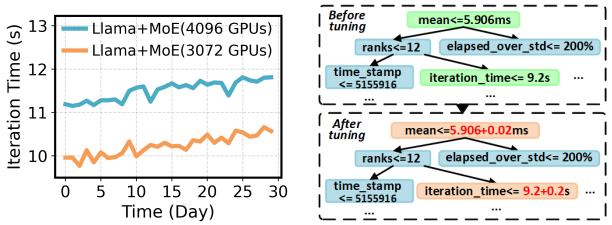


Figure 9: Communication operators and logging scheme.



(a) Distribution drift (b) Auto-tuning against mean shift

Figure 10: (a) Distribution drift and (b) Holmes's auto-tuning.

monitor keeps tracing the log of one pivot GPU for analysis in real time. ② Once the iteration monitor observes an irregularity, the operator detector will perform enhanced abnormal operator detection on the logs. ③ Then root localizer module searches the GPUs in COG and collects the log data of these GPUs that are correlated to the irregularity. ④ The device analytics module calculates the probability of anomaly for every device and identifies the top-K anomalous devices. Finally, the on-call technicians can further confirm and replace the anomalous devices.

4.2 Abnormal Operators Detection

4.2.1 Communication Operator Tracing

Since the irregularity does not generate any error logs, we are motivated to implement a logger to capture this phenomenon. However, logging all information during LLM training can significantly interfere with the training process. Therefore, it is challenging to develop a lightweight approach with minimal overhead on LLM training.

CommOps log. Figure 9 shows a typical execution graph of a single iteration when training LLMs. There are two communications, i.e., `All-to-All` ending at T_1 and `All-Reduce` starting at T_2. The computation operators are executed during T_1 and T_2. We find that both the computation issues and communication issues can be observed by monitoring communication operators. For instance, if the `GEMM` operator executes abnormally with longer computation time, the start time of the `All-Reduce` operator will be later than T_2. Even when computation and communication overlap during training, interdependent communication operators remain vulnerable to cascading delays. Furthermore, communication operators indicate data transmission among GPUs, which we can use to trace the propagation of irregularity's impact. However, popular communication libraries such as NCCL [1] do not provide the interface to record the communication operators directly. In Holmes, we introduce a logging scheme called CommOps log to keep tracing the communication operators as shown in the right part of Figure 9. Each entry of the log records the features of a specific communication operator, which includes the communication pattern, the elapsed time, time stamps, communicator, GPU rank, etc. Since the number of communication operators is much less than that of computation operators in the typical LLM training, CommOps log could trace the fine-grained training process with almost negligible overhead.

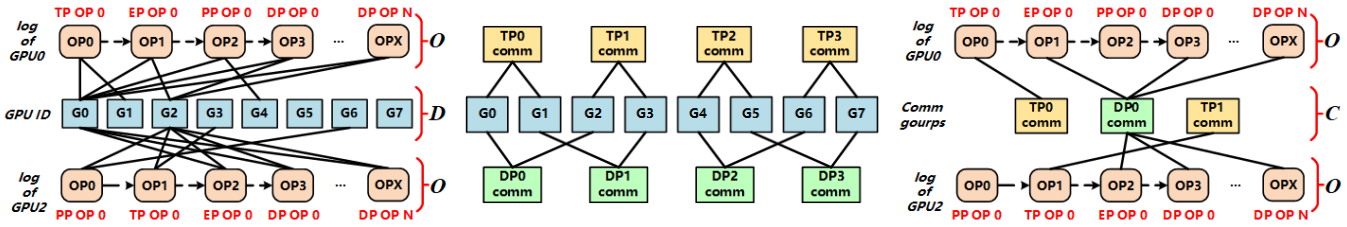
4.2.2 Automatic Operator Detection

Iteration monitor. As we present in § 3.1, the irregularities have apparent characteristics in end-to-end measurement, e.g. prolonged iteration time. For simplicity, we perform irregular iteration detection strictly according to our definition of δ-irregularity.

Unified operator detection. Typically, an iteration always contains tens of thousands of operators. Once an irregular iteration is detected, we need to trace the relevant operators. However, due to the operators' execution time varying dramatically, we can hardly leverage traditional anomaly detection methods such as the 3-sigma rule to detect them. For example, if the duration of an abnormal iteration increases from 7 seconds to 9 seconds, potentially caused by two abnormal operators. The elapsed time of one operator increases from 0.02 ms to 1 ms (by 50×), while the elapsed time of the other operator increases from 1000 ms to 3000 ms (by 3×). In this case, if we only consider the increasing ratio, the former operator will be detected as an abnormal operator rather than the latter. In Holmes, we choose to use Random Forest (RF) to train a prediction model for abnormal operator detection. We consider each entry in the operator log as an individual sample of the RF, while the features are the records of the entry. Since RF can effectively capture multiple features, we also incorporate global information, such as iteration time and average operator elapsed time within a time window into the features. In this way, Holmes can detect abnormal operators with the consideration of the importance and variation of operators.

Detection features. Each operator has specific features in a given iteration, however, detection requires analysis of global information. Below is a list of features we construct:

- *Mean and standard deviation (std).* The average value and standard deviation of operator execution time.
- *Z-Score.* For each point X in the time window, Z-Score is calculated as: $Z = (X - mean)/std$.
- *Quartiles.* The first quartile ($Q1$), the second quartile ($Q2$, i.e., median), and the third quartile ($Q3$) of feature.
- *IQR.* The interquartile range (IQR), calculated as $Q3 - Q1$.
- *Fixed features.* Ranks, data size, communication pattern.



(a) Original COG, (b) Communication grouping. (c) Reconstructed COG.

Figure 11: Communication operator graph.

Distribution drift adaptation. The RF can predict the abnormal operators based on training samples. However, due to the non-stationarity inherent in production, the trained RF model is prone to *distribution drift* [13, 19, 63], which subsequently degrades its forecasting performance over time. For example, we depict the average iteration times per day of two LLM training instances in Figure 10(a). An observation reveals that the average iteration time exhibits slight daily variations. Both iteration times increased by 5.92% and 7.03% respectively within a month, indicating a drift in the distribution of features (e.g., iteration time). Periodic tree model retraining mitigates drift, but RF training requires costly manual data labeling. Holmes counters this drift by enhancing tree-based predictions with an automatic tuning approach. In tree-based models such as RF, there are numerous interconnected tree nodes. Each node is capable of classifying data samples by utilizing trained attribute values associated with a specific feature. We directly tune the attribute values in the trained model periodically. For instance, as shown in Figure 10(b), a trained RF model contains nodes with specific features. We detect that the mean of iteration time right shifts for 0.2 seconds (from 9.0 seconds to 9.2 seconds). To cope with this mean shift, we adjust all attribute values associated with the shifted features, e.g. adding 0.2 seconds to the attribute value 9.0s in the node "iteration_time". This tuning approach allows Holmes to adapt to distribution drift during training without the need for additional efforts such as incremental training.

4.3 Graph-based Causes Localization

During the training, an anomaly in the device causing longer training iteration will propagate to impact other GPUs, appearing longer iteration time. When there are a lot of irregular iteration logs among these GPUs, it is challenging to identify the anomalous device. To address this, Holmes constructs a novel communication operator graph (COG), and then designs an effective algorithm to localize the irregularity.

4.3.1 Communication Operator Graph

Constructing the COG. Holmes designs a novel graph called the communication operator graph (COG) $G = (O, D, E)$, where O and D are the sets of operators and GPUs respectively, and E denotes the edges between O and D. Specifically, Figure 11(a) illustrates each $o \in O$ represents a communication operator in a log and each $g \in D$ represents an individual GPU. An edge $e = (o, g) \in E$ exists between o and g if g is participated in o. In addition, the operators in the same log are connected in the form of a linked list to represent the execution order, as the dash lines in Figure 11(a) show. However, this naive format of COG may result in an unacceptable complexity. For example, if an LLM training includes 4096 GPUs ($|D| = 4096$) and 4000 operators in each log ($|O| = 4000 * 4096$), the number of connections $|E| = |O| * |D|$ can be 6.71×10^{10}, which is prohibitively complicated.

Reconstructing the COG using communication grouping. To address the curse of dimensionality, Holmes leverages the grouping feature of communication operators to reduce the connections. In Figure 11(b), we take 8 GPUs training as an example, where the degrees of data, pipeline, and tensor parallelism are 2. With the communication pattern defined, all the communication operators are executed on fixed groups of GPUs. For instance, all the communication of data parallelism in GPU G0 are executed with G2, then G0 and G2 are called a DP group (labeled as "DP0 comm"). This group conception is used in various parallelism implementations and communication libraries, such as the *communicator* in NCCL [1]. In Holmes, we maintain a mapping of groups such as "DP comm" and their corresponding GPU sets. By substituting GPUs with groups, we reconstruct COG $\hat{G} = (O, C, E)$, where C represents the set of GPU groups. As shown in Figure 11(c)[1], the number of edges in the $\hat{G}$ is much lower than in G. Furthermore, since all the connections of O are sparse (only one to C), Holmes could use sparse storage for the edges, reducing the memory and processing overhead.

4.3.2 Graph-based Localization Algorithm

To localize the irregularity, we design a recursive algorithm (Algorithm 1) to identify the GPU or GPU sets where the root cause occurs. When a set of abnormal operators *OPs* is detected in the pivot GPU, we call the recursive `locateNeighbor` function for each operator (line 2). Given the low probability that multiple types of irregularities occur in the same operator, our algorithm treats input abnormal

[1] Typically, the GPUs performing DP communication will always perform the same EP communication. We group them together to further reduce the COG complexity. Furthermore, we do not group GPUs in PP communication as it is only executed among 2 GPUs.

Algorithm 1: Root cause localization

```
   Input: Ĝ = (O,C,E), OPs
   Output: OPERATORS, CAUSES
1  OPERATORS ← ∅, CAUSES ← ∅;
2  for all o of OPs do locateNeighbor(o) ;
3  Function locateNeighbor(o):
4      if o.type ∈ CC_OP then
5          ▷ Breadth First for CC
6          for all neighbors group of o do
7              if any neighbor operator n of group is normal then
8                  ▷ Track operators of the GPU g where n belongs to
9                  for all neighbors operators po of g before n do
10                     if any po is abnormal then
11                         locateNeighbor(po), break
12                     else
13                         OPERATORS.add(po),
                             CAUSES.add(COMPUTE), return;
14                     end
15                 end
16             else
17                 OPERATORS.add(o), CAUSES.add(NETWORK)
                     return;
18             end
19         end
20     else
21         ▷ Depth First for P2P
22         if o.peer = ∅ or o.peer operator is abnormal then
23             OPERATORS.add(o), CAUSES.add(NETWORK);
24             return;
25         end
26         ▷ Track operators of peer GPU
27         for all neighbors operator po of o.peer GPU do
28             if po before o.peer operator is abnormal then
                   locateNeighbor(po);
29         end
30     end
31 end
```

operators separately. For each communication operator, we use breadth-first search (BFS) and depth-first search (DFS) to traverse the operators in collective communication (CC) and point-to-point communication (P2P), respectively. For each GPU, we track the operators belonging to it and detect whether they are abnormal. We initialize the set of root cause operators and types of causes to $\emptyset$ (line 1).

BFS for Collective Communication. We perform a breadth-first search to traverse all the GPUs within a collective communication group. For an abnormal collective communication operator o, we identify its corresponding group c and verify all associated operators across group GPUs. These operators represent the same communication but are recorded from different GPUs. If all the operators are reported as abnormal, we confirm the communication itself as the root cause and terminate the analysis (line 17). Otherwise, there exists an operator n that is executed normally, i.e. normal execution duration. However, since the collective communication requires all the GPUs to be ready, this indicates that the abnormal operators in the rest of the GPUs take too much time waiting for the "normal" operator n to start. We can infer that irregularity occurs and delays n. In this case, we will track all the previous

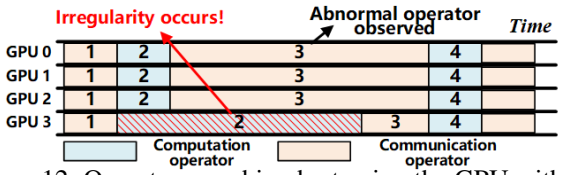


Figure 12: Operator searching by tracing the GPU with the normal collective operator.

operators before n (line 9). If any operator po is abnormal, we recursively analyze po (line 10), otherwise, we identify the causes as the computation prolongs and impacts n (line 13). Figure 12 demonstrates this principle through a 4-GPU scenario where GPU3's prolonged operator2 execution forces others to wait at operator3. Our algorithm correctly attributes the anomaly to GPU3's computational delay by observing abnormal operator3 executions in other GPUs versus GPU3's normal operator3.

DFS for P2P communication. In P2P communication, each operator involves two GPUs with peer relationships. We perform a depth-first search along with the pipeline stages. For each GPU, the other GPU is referred to as the peer GPU, and the corresponding operator in this peer GPU is termed the peer operator. For a specific P2P operator o, we directly detect its peer operator due to its inherent dependency. For example, if a `Recv` is abnormal, we can find its peer `Send` according to the COG. If both operator o and its peer operator are abnormal, then it can be confirmed that the cause of the irregularity is related to the abnormal P2P network communication (line 22). If only o is abnormal, its prolonged execution implies synchronization delays while awaiting the peer operator's readiness. In this case, we track the operators before the peer operator on the peer GPU and further locate the abnormal operators among them (lines 27-29). The readers can refer to appendix C for more details.

4.4 Cross-iteration Analysis

By combining the detection (§ 4.2) and localization (§ 4.3), we are able to find the root abnormal operators, involved GPUs and the cause type of an irregularity. However, there may be false positives if we localize the causes only based on a single iteration. Meanwhile, we can hardly identify multiple concurrent irregularities. To address these challenges, Holmes leverages a cross-iteration analysis in a time window T to find out the most possible anomalous devices.

Influence assessment of irregularities. Given detected irregularities $X = \{x\}$ within T, we first define a quality metric to evaluate the irregularity's influence on training. Since the irregularity prolongs the iteration time for δ times, Holmes calculates the impact of irregularity on the overall iteration time as a metric called *irregularity rate* $\mathcal{R}(x, t_s, T)$, where the time window starts at t_s. The impact of irregularity can be reflected in its corresponding abnormal operators. $\mathcal{R}(x, t_s, T)$ is calculated as the Pearson Correlation Coefficient [4, 59] of

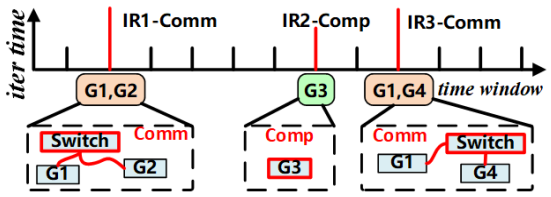


Figure 13: Cross-iteration analysis.

the abnormal operator duration and the iteration time:

$$\mathcal{R}(x,t_s,T) = \frac{Cov\left(\mathbb{T}^T_{t_s}(x), \mathbb{T}^T_{t_s}(I)\right)}{\sigma_{\mathbb{T}^T_{t_s}(x)}\sigma_{\mathbb{T}^T_{t_s}(I)}}$$

where $\mathbb{T}^T_{t_s}(x)$ denotes the series of the execution time of the abnormal operator involved in x, and $\mathbb{T}^T_{t_s}(I)$ denotes the series of the overall iteration time. The larger $\mathcal{R}$ an irregularity has, the more severe influence it has on training.

Anomalous device estimation. Any anomalous device in the cluster leads to the irregularities, we here to calculate the anomalous probability of each device. Noting that the abnormal operators may be different among iterations, they can be caused by the same anomalous device. We first define the device set $D = \{d\}$, including such as GPUs, links connecting GPU, NVLinks, and switches. Each device has an *accumulated index* $I(d)$ to indicate its possibility of being anomalous. With the irregularity rate $\mathcal{R}$ and involved GPUs of the irregularities, we can calculate all the $I(d)$ by accumulating $\mathcal{R}$. Specifically, for the irregularity that is identified as computation issues in a set of GPUs, we directly accumulate $\mathcal{R}$ into I of the GPUs. While for irregularity stemming from the network issues, we traverse all the network devices that lie on the communication paths among involved GPUs, and accumulate the $\mathcal{R}$ into their I. Given a device d and irregularities set X, we use the MAP estimation to perform the accumulation as described in appendix D, which exploits the prior probability of the devices being anomalous. Figure 13 depicts the communication irregularity "IR" involves GPU1 and GPU2. We accumulate the irregularity rate to all the communication devices between them, i.e. a switch and two links. While for the computation irregularity "IR" involving GPU3, we directly accumulate the irregularity rate to GPU3. Finally, Holmes ranks $I(d)$ and identifies the top K devices with the highest $I(d)$, which means that they are most likely to be anomalous and severely affect the training.

5 Evaluation

In this section, we present a series of experiments on Holmes. Experimental results manifest that Holmes can achieve the localization accuracy of 97.21% in the models (§ 5.1) trained in a mega-scale GPU cluster. Our production-level testbed simulations show that the irregularity localization can be completed within 30.3 seconds (§ 5.2). Finally, we study Holmes's localization details by a case from production (§ 5.3).

5.1 Large-scale Trace-driven Analysis

We collect CommOps logs over a three-month period from our production training cluster, which is equipped with over 10,000 GPUs, including NVIDIA H800. Each GPU is paired with an NVIDIA ConnectX-7 Dx NIC, providing a bandwidth of 400Gbps. The software systems include Megatron-LM [41], DeepSpeed [47], and the NCCL [1]. The training process adopts multi-dimensional parallelism strategies that encompass tensor- [41], data- [11,46], pipeline- [22,40], and expert- parallelism [28]. The LLMs are carefully designed models based on LLaMA [56], LLaMA2 [57] and GPT3 [7], with thousands of billions of parameters.

Dataset. We integrate our CommOps logger into our production cluster and generate the communication logs accordingly. Manual diagnosis is conducted to identify anomalous devices and corresponding irregularities. Firstly, we rely on expert knowledge to enumerate all the possible anomalous devices. Subsequent manual inspection evaluates device resource metrics (e.g., GPU usage). Suspected devices are isolated upon confirmation through stress testing.

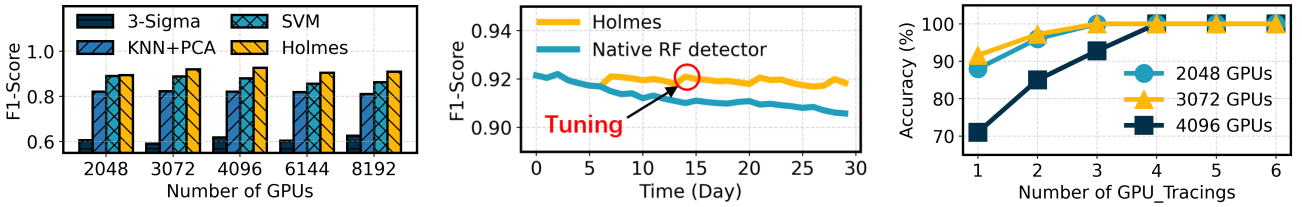
System configuration. The training set of the detection tree model is the CommOps log data of LLM training. Specifically, we train a distinct tree model for every LLM training using their CommOps logs. Each sample represents the feature of a specific communication operator. Most of the LLM training spans several months and we manually construct the training dataset using a few hours' logs. For cross-iteration analysis, given the low probability of concurrent failures in multiple devices, K is defaulted to 2. The irregularity rate calculation spans 100 iterations by default.

5.1.1 Modular study

We present the key modular study results below and other supplementary experiments can be found in appendix E.

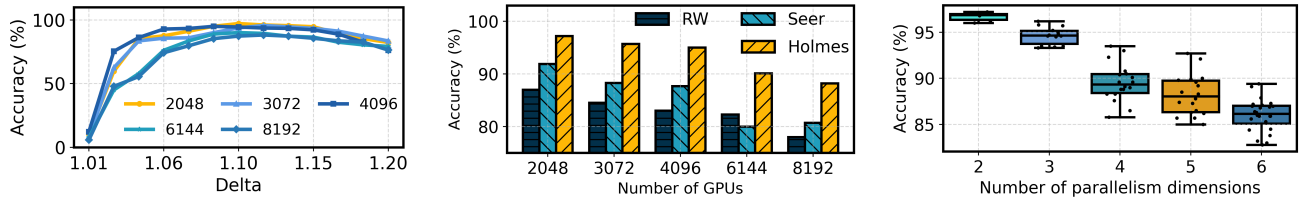
Abnormal operator detection accuracy. We first study the performance of abnormal operator detection in Holmes. Figure 14(a) compares our RF detector with three baseline methods, where higher F1-Score indicates better detection performance, the details of the setting can be referred to appendix F. For various scales of the training, the F1-Score of RF exceeds 0.89. Specifically, when training with 4096 GPUs, the F1-Score of RF reaches 0.93, surpassing the scores of the 3-sigma rule and K-Nearest Neighbor (KNN) by 0.31 and 0.11, respectively. Even when compared to SVM, Holmes still achieves a higher F1-Score by 0.07. These phenomena manifest that our RF detector could detect abnormal operators correctly even in large-scale training.

Distribution drift adaptation. We then study the auto-tuning of Holmes against the distribution drift. For a 3072 GPUs training lasting one month, we set the auto-tuning interval to one week and train the same RF model for Holmes and a naive RF detector without tuning. Figure 14(b) illustrates how the F1-Scores evolve over time. Starting from the second



(a) F1-Score of different detection methods (b) Distribution drift adaption (c) Localization accuracy

Figure 14: Performance of detection.



(a) Localization accuracy of δ (b) Scalability to number of GPUs (c) Scalability to number of parallel dimensions

Figure 15: End-to-end localization accuracy.

week, Holmes consistently outperforms the native RF. The F1-Score of Holmes is always above 0.917, while the score of native RF declines from 0.922 on the first day to 0.906 by the end of the month. Meanwhile, an increase in the F1-Score is clearly observed on each auto-tuning date. Holmes automatically collects the feature shift such as iteration time shift and tunes the attributed value for these features weekly, keeping high detection performance. Furthermore, the tuning frequency can be adapted according to the training.

Localization algorithm accuracy. To evaluate the localization algorithm, we present per-iteration localization accuracy on three training with thousands of GPUs. As shown in Figure 14(c), we run the localization algorithm several times independently, beginning with 1 to 6 randomly selected monitored GPUs. The GPUs that have the earliest executed abnormal operators are then chosen as the localization results. When monitoring a single GPU, Holmes achieves localization accuracy of 91.6%, 88.0%, and 71.1% with 2048, 3072 and 4096 GPUs, respectively. As the training involves more GPUs, GPUs are more likely to interfere with each other, which limits the localization accuracy. However, since the localization algorithm starts from few GPUs, we can monitor more GPUs to improve the localization accuracy in production. For example, in training with 4096 GPUs, the accuracy increases from 71.1% to 100% as the monitored GPUs increase.

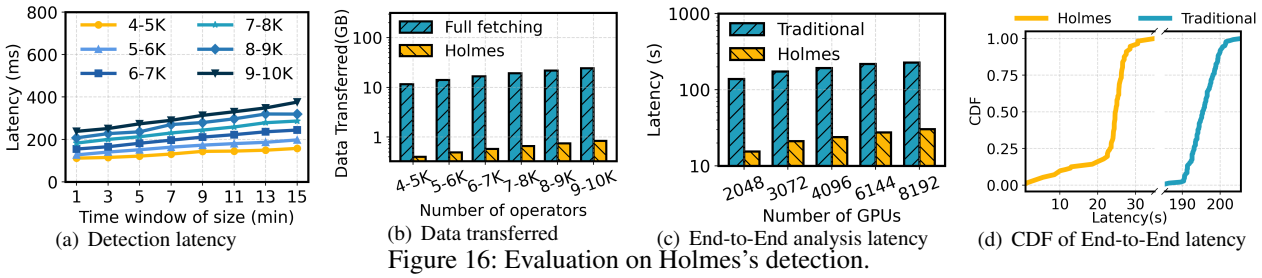
5.1.2 End-to-end Performance

Baselines. Since there are no existing systems for irregularity localization during training, we adapt the idea of two anomaly diagnosis counterparts [16, 59] from the field of (micro-)service computing to irregularity localization. "Seer" is a neural network based method [16] that directly localizes the potential anomalous device using the CommOps logs. We further replace our graph-search algorithm with the ranking and random walk method [59], called "RW".

Accuracy of delta. The irregularity refers to the training iteration time prolongs for δ times. Hereby we evaluate Holmes's performance when δ varies. As shown in Figure 15(a), Holmes can detect the irregularities in most of the training when δ varies from 1.06 to 1.12. When the δ is less than 1.04, Holmes can not tell the normal iterations and irregular iterations since the training iteration time fluctuates. While δ is larger than 1.12, Holmes can not detect slight irregularities like performance fluctuations. We empirically set our default δ of the detection system as 1.1.

Scalability to GPU number. Figure 15(b) shows the end-to-end localization accuracy when training LLMs at various GPU scales. With 2048 GPUs, Holmes achieves a localization accuracy of 97.2%. Even in the extreme scenario of 8192 GPUs, Holmes maintains an accuracy of 88.2%, while RW and Seer attain only 78.0% and 80.7%, respectively. RW's random walk on COG effectively localizes anomalous *communication* devices but cannot detect computational anomalies, as CommOps logs do not explicitly provide their information. Seer exhibits rapid accuracy degradation with GPU scaling, as its CNN+LSTM architecture struggles to model inter-operator dependencies at extreme scales. Holmes explicitly models the operator relationships, thereby achieving superior localization accuracy and interpretability.

Scalability to parallelism dimension. We assess the localization accuracy of Holmes when the training parallelism varies. The parallelization strategies include the aforementioned strategies and customized approaches. The LLMs in our experiments implement various combinations of these strategies, so each test case is designated by the number of parallelism dimensions rather than the specific combinations, as shown in Figure 15(c). For the commonly used 3D and 4D parallelisms, Holmes achieves a median localization accuracy of 94.6% and 89.6%, with a first quartile of 93.6% and 88.3%, respectively. The more complex parallelism introduces the more complicated communication patterns, leading to an accuracy decline. However, even with 6D parallelism, Holmes's median localization accuracy reaches 86.0%, with a



(a) Detection latency (b) Data transferred (c) End-to-End analysis latency (d) CDF of End-to-End latency

Figure 16: Evaluation on Holmes's detection.

first quartile of 85.2%, which shows that Holmes is adaptable to existing parallelization strategies.

5.2 Production-level Testbed

Localization latency is evaluated in our production network environments through a simulation framework that enables dynamic GPU configuration adjustments. Holmes's prototype operates on production network infrastructure, employing CPU-based log generation to emulate GPU training processes while preserving real collective communication operator logs. This approach maintains temporal fidelity by substituting GPU computation durations with simulated equivalents, with log transmission mechanisms mirroring production environments. Our testbed consists of nine machines: one acts as the Holmes analytics server, while the others serve as training machines. For example, to simulate a virtual cluster with 2048 GPUs, each training machine runs 256 log writers, acting as 256 GPU training processes. Each machine is equipped with an AMD EPYC 7K83 64-Core Processor and a 200 Gbps NVIDIA ConnectX-6 Dx NIC. Machines are connected to a Clos network using 10 TCS8400 switches to simulate the data plane network in production (Figure 22). We compare Holmes to the 3-sigma rule approach, labeled as "Traditional", which uses all GPUs' logs.

Detection latency. We study the detection latency by adjusting the detection time window size and the number of communication operators. A larger window size means that more iterations are processed in Holmes. As shown in 16(a), the overall detection latency for a single GPU log remains within 400 ms. When the window size increases from 1 minute to 15 minutes, the detection latency increases from 111.7 ms to 157.6 ms with 4-5K operators inside a log. The detection latency increases linearly as more operators are involved, since each operator needs to be processed by the RF model. An observation is that even when the time window extends to 15 minutes and the number of operators reaches 9-10K, the detection latency remains as low as 375.7 ms.

Localization with progressive fetching. To reduce collected data volume and localization time, Holmes implements a progressive fetching scheme that only collects logs from the GPUs that are traversed by our localization algorithm. Figure 16(b) shows the collected logs from the training machines to the analytics server for one localization process. While full log analysis is used as our baseline. For cases with 9-10K operators, the average per-iteration traffic load is 24.36 GB, whereas Holmes's load is only 0.84 GB, with a reduction of 96.6%. This manifests that our progressive fetching scheme effectively utilizes only the necessary data, preventing network overload due to excessive CommOps log traffic.

End-to-end localization latency. We evaluate the overall localization latency of Holmes when training with different cluster scales. As shown in Figure 16(c), Holmes significantly outperforms the traditional approach in all scenarios. For instance, the average latency in Holmes is 21.2 seconds when training with 3072 GPUs, exhibiting an 87.8% reduction compared to the baseline. As the number of GPUs scales from 2048 to 8192, the overall latency of Holmes increases from 15.5 seconds to 30.3 seconds. This increase is due to the prolonged detection time for irregular operators and the increased transmission time for CommOps logs. Figure 16(d) shows the cumulative distribution of the latency for the training using 3072 GPUs. Nearly 96.2% of the overall latency values fall below 30.0 seconds, while the 99th percentile is at 32.8 seconds. This implies the absence of a heavy tail, which is advantageous for Holmes's practical operation.

5.3 Case Study

We use an irregularity example from the production to illustrate the workflow of Holmes in Figure 17. The training in this case is executed under typical 3D parallelism, where the degrees of DP, PP and TP are 96, 8 and 4, respectively.

1) Irregularity detection. At the beginning, Holmes keeps monitoring the training in GPU 0 (DP:0, PP:0, TP:0). Upon the detection of an irregular iteration (whose time extends to 10.1s), we perform the abnormal operator detection on GPU 0, and find a `Send` operator with extended execution time.

2) DFS inside PP group 0. As the `Send` is a P2P operator in PP group 0, Holmes fetches the CommOps log in the peer GPU (DP:0, PP:1, TP:0) according to the COG. We find that the P2P operator `Receive` is normal while another `Send` operator is abnormal. We keep searching on this `Send`. After repeating multiple rounds, we transit to GPU 2304 (DP:0, PP:6, TP:0) whose `All-Reduce` operator is found abnormal.

3) BFS inside DP group 24. The `All-Reduce` belongs to DP group 24, we then detect all the GPUs in this DP group according to the COG. Only one GPU's (GPU 2572) corresponding `All-Reduce` is normal, while the other GPUs have the same abnormal DP `All-Reduce` operator. This indicates

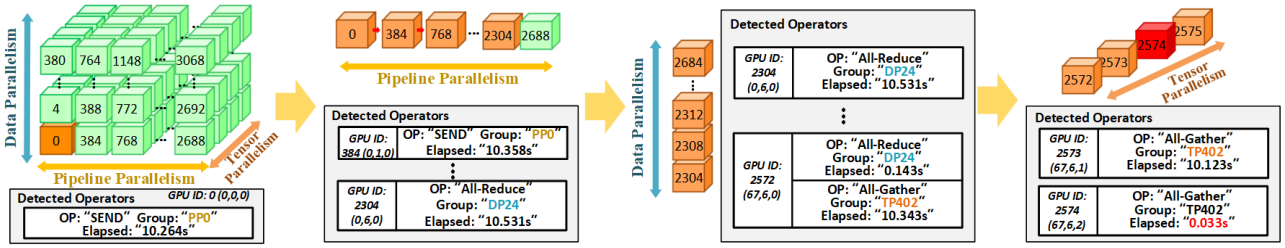


Figure 17: An example of irregularities detection with 3072 GPUs.

that GPU 2572 is the straggler and the rest of the GPUs start the `All-Reduce` ahead of GPU 2572 and wait till it is prepared. In GPU 2572, the abnormal operator we can detect is an `All-Gather` belonging to TP group 402.

4) BFS inside TP group 402. In this group, we find that the same `All-Gather` is normal only in GPU 2574, whereas no other abnormal operator in this GPU. We can infer that the irregularity is caused by the GPU 2574 computing slowly.

Summary. In this example, the number of GPUs we access is 106 (i.e., $6+96+4$), which is less than the total number of 3072. Combining with a long-term analysis during hundreds of iterations, Holmes suggests substituting the GPU 2574.

6 Discussion

Considering operator implementation. Communication operators have been implemented by various algorithms (e.g., All-Reduce variants [1, 50]) including ring [44, 55], recursive doubling [49, 55], and topology-aware optimizations [15, 34]. While operator-level behavioral discrepancies enable Holmes's straggler detection, integrating implementation-specific operator tracking could enhance network device localization. This refinement, however, risks excessive logging overhead during multi-hop transmission monitoring – a critical trade-off requiring future exploration between diagnostic precision and runtime efficiency.

Resource metric integration. Traditional hardware monitors provide coarse-grained metrics (GPU utilization, temperature) with inherent noise and limited LLM context awareness. Though augmenting Holmes with real-time resource traces could improve localization – for instance, validating GPU status during diagnostic – our current focus remains anomaly characterization and core detection framework development, reserving multi-source correlation for subsequent research.

Experience in using network telemetry. Initial experiments with RNIC queue-pair (QP) telemetry achieved full communication reconstruction via per-chunk logging, yet proved impractical for large clusters: 10,000 RNICs generated over 10TB/iteration of QP data, exceeding real-time processing capabilities. This scalability constraint motivated Holmes's communication library (CCL)-centric design, balancing observability with operational feasibility.

7 Related Work

Failure diagnosis in GPU cluster. Recent research on anomaly analysis in machine learning has focused on failure diagnosis. A series of works [21, 25, 43, 54, 62] employed error logs for diagnosis. For example, MegaScale [25] diagnosed the failure by visualizing the data collected with a custom CUDA event monitor. Different from them, Holmes localizes the irregularity according to the communication relationship between GPUs without error logs. Another failure diagnosis approach was made by resource monitoring, such as GPU utilization, memory usage, etc [2, 18, 23, 32]. However, they lacked awareness of the detailed training process, while the diagnosis performance highly depends on the granularity of resource monitoring. Holmes can learn the training process precisely at the operator level from the communication logs. Some works [8, 9, 48, 52, 53, 58] focus on develop performance debugging tools for high-performance computing (HPC). Holmes leverages domain-specific knowledge like parallel training strategies to detect irregularity.

Anomaly diagnosis in distributed applications. Research has focused on anomaly diagnosis in other distribution applications, which typically involve detection and localization. In anomaly detection, some work [24, 38, 39] directly analyzed the operational logs, while the others [12, 16, 26, 33, 36, 61] focused on leveraging machine learning. For example, Seer [16] trained a neural network to detect QoS violations. Holmes enhances the machine learning detection model to adapt to the distribution drift in LLM training. In anomaly localization, a series of research [6, 29, 30, 33, 59] proposed the idea of building a graph based on specific applications. MicroHECL [30] analyzed anomaly propagation chains based on a dynamically constructed service call graph. Holmes continues the idea of graph analysis and proposes a novel graph to model the communication relationship within LLM training.

8 Conclusion

In this paper, we systematically analyze the irregularity phenomenon in LLM training and propose a first-of-its-kind system called Holmes for irregularity localization. Holmes employs an enhanced abnormal operator detection model and a novel communication operator graph to perform efficient irregularity localization. Evaluation of production traces shows that Holmes achieves localization accuracy of 97.21%.

Acknowledgements

We thank our shepherd Fang Zhou and the anonymous reviewers for their constructive feedback. This work was supported in part by the Natural Science Foundation of China under Grant 62072117, the Shanghai Natural Science Foundation under the Grant 22ZR1407000, and the Tencent Research Program RBFR20230724.

References

[1] Nvidia NCCL, 2024. https://github.com/nvidia/nccl.

[2] Nvidia Nsight Systems, 2024. https://developer.nvidia.com/nsight-systems.

[3] Torch Profiler, 2024. https://pytorch.org/docs/stable/profiler.html.

[4] H. Abe and S. Tsumoto. Analyzing behavior of objective rule evaluation indices based on a correlation coefficient. In *Knowledge-Based Intelligent Information and Engineering Systems, 12th International Conference, KES 2008, Zagreb, Croatia, September 3-5, 2008, Proceedings, Part II*, volume 5178 of *Lecture Notes in Computer Science*, pages 758–765. Springer, 2008.

[5] L. A. Barroso, U. Hölzle, and P. Ranganathan. *The Datacenter as a Computer: Designing Warehouse-Scale Machines, Third Edition*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2018.

[6] Á. Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero. Graph-based root cause analysis for service-oriented and microservice architectures. *J. Syst. Softw.*, 159, 2020.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

[8] M. Burtscher, B. Kim, J. R. Diamond, J. D. McCalpin, L. Koesterke, and J. C. Browne. Perfexpert: An easy-to-use performance diagnosis tool for HPC applications. In *Conference on High Performance Computing Networking, Storage and Analysis, SC 2010, New Orleans, LA, USA, November 13-19, 2010*, pages 1–11. IEEE, 2010.

[9] C. Cascaval, E. Duesterwald, P. F. Sweeney, and R. W. Wisniewski. Performance and environment monitoring for continuous program optimization. *IBM Journal of Research and Development*, 50(2.3):239–248, 2006.

[10] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023.

[11] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1232–1240, 2012.

[12] Q. Du, T. Xie, and Y. He. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In *Algorithms and Architectures for Parallel Processing - 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part IV*, volume 11337 of *Lecture Notes in Computer Science*, pages 560–572. Springer, 2018.

[13] W. Fan, P. Wang, D. Wang, D. Wang, Y. Zhou, and Y. Fu. Dish-ts: A general paradigm for alleviating distribution shift in time series forecasting. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 7522–7529. AAAI Press, 2023.

[14] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23:120:1–120:39, 2022.

[15] G. Feng, D. Dong, and Y. Lu. Optimized MPI collective algorithms for dragonfly topology. In L. Rauchwerger, K. W. Cameron, D. S. Nikolopoulos, and D. N. Pnevmatikatos, editors, *ICS '22: 2022 International Conference on Supercomputing, Virtual Event, June 28 - 30, 2022*, pages 14:1–14:11. ACM, 2022.

[16] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, pages 19–33. ACM, 2019.

[17] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang, S. Zhang, M. J. Fernandez, S. Gandham, and H. Zeng. RDMA over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference, ACM SIGCOMM 2024, Sydney, NSW, Australia, August 4-8, 2024*, pages 57–70. ACM, 2024.

[18] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, pages 350–361. ACM, 2011.

[19] H. He, Q. Zhang, K. Yi, K. Shi, Z. Niu, and L. Cao. Distributional drift adaptation with temporal conditional variational autoencoder for multivariate time series forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2024.

[20] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*, pages 120–134. ACM, 2022.

[21] Q. Hu, Z. Ye, Z. Wang, G. Wang, M. Zhang, Q. Chen, P. Sun, D. Lin, X. Wang, Y. Luo, Y. Wen, and T. Zhang. Characterization of large language model development in the datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*, pages 709–729. USENIX Association, 2024.

[22] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. X. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 103–112, 2019.

[23] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, pages 947–960. USENIX Association, 2019.

[24] T. Jia, P. Chen, L. Yang, Y. Li, F. Meng, and J. Xu. An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services. In *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*, pages 25–32. IEEE, 2017.

[25] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong, Y. Jia, S. He, H. Chen, Z. Bai, Q. Hou, S. Yan, D. Zhou, Y. Sheng, Z. Jiang, H. Xu, H. Wei, Z. Zhang, P. Nie, L. Zou, S. Zhao, L. Xiang, Z. Liu, Z. Li, X. Jia, J. Ye, X. Jin, and X. Liu. Megascale: Scaling large language model training to more than 10, 000 gpus. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*, pages 745–760. USENIX Association, 2024.

[26] M. Jin, A. Lv, Y. Zhu, Z. Wen, Y. Zhong, Z. Zhao, J. Wu, H. Li, H. He, and F. Chen. An anomaly detection algorithm for microservice architecture based on robust principal component analysis. *IEEE Access*, 8:226397–226408, 2020.

[27] V. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro. Reducing activation recomputation in large transformer models. *CoRR*, abs/2205.05198, 2022.

[28] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[29] Y. Li, J. Tan, B. Wu, X. He, and F. Li. Shapleyiq: Influence quantification by shapley values for performance debugging of microservices. In T. M. Aamodt, M. M. Swift, and N. D. E. Jerger, editors, *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, pages 287–323. ACM, 2023.

[30] D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu. Microhecl: High-efficient root cause localization in large-scale microservice systems. In *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021*, pages 338–347. IEEE, 2021.

[31] J. Liu, J. H. Wang, and Y. Jiang. Janus: A unified distributed training framework for sparse mixture-of-experts models. In *Proceedings of the ACM SIGCOMM 2023 Conference, ACM SIGCOMM 2023, New York, NY, USA, 10-14 September 2023*, pages 486–498. ACM, 2023.

[32] K. Liu, Z. Jiang, J. Zhang, H. Wei, X. Zhong, L. Tan, T. Pan, and T. Huang. Hostping: Diagnosing intra-host network bottlenecks in RDMA servers. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, pages 15–29. USENIX Association, 2023.

[33] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue, and D. Pei. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*, pages 48–58. IEEE, 2020.

[34] J. Ma, D. Dong, C. Li, K. Wu, and L. Xiao. PAARD: proximity-aware all-reduce communication for dragonfly networks. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), New York City, NY, USA, September 30 - Oct. 3, 2021*, pages 255–262. IEEE, 2021.

[35] K. Maeng, S. Bharuka, I. Gao, M. C. Jeffrey, V. Saraph, B. Su, C. Trippel, J. Yang, M. Rabbat, B. Lucia, and C. Wu. Understanding and improving failure tolerant training for deep learning recommendation with partial recovery. In *Proceedings of Machine Learning and Systems 2021, MLSys 2021, virtual, April 5-9, 2021*. mlsys.org, 2021.

[36] L. Mariani, C. Monni, M. Pezzè, O. Riganelli, and R. Xin. Localizing faults in cloud systems. In *11th IEEE International Conference on Software Testing, Verification and Validation, ICST 2018, Västerås, Sweden, April 9-13, 2018*, pages 262–273. IEEE Computer Society, 2018.

[37] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.1*, Nov. 2023.

[38] H. Mi, H. Wang, Y. Zhou, M. R. Lyu, and H. Cai. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Trans. Parallel Distributed Syst.*, 24(6):1245–1255, 2013.

[39] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhattacharya. Anomaly detection using program control flow graph mining from execution logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 215–224. ACM, 2016.

[40] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia. Pipedream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, pages 1–15. ACM, 2019.

[41] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia. Efficient large-scale language model training on GPU clusters using megatron-lm. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, page 58. ACM, 2021.

[42] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[43] G. Ostrouchov, D. Maxwell, R. A. Ashraf, C. Engelmann, M. Shankar, and J. H. Rogers. GPU lifetimes on titan supercomputer: survival analysis and reliability. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 41. IEEE/ACM, 2020.

[44] P. Patarasuk and X. Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distributed Comput.*, 69(2):117–124, 2009.

[45] S. Rajasekaran, M. Ghobadi, and A. Akella. CASSINI: network-aware job scheduling in machine learning clusters. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*. USENIX Association, 2024.

[46] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM, 2020.

[47] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 3505–3506. ACM, 2020.

[48] S. Roy, A. C. König, I. Dvorkin, and M. Kumar. Perfaugur: Robust diagnostics for performance anomalies in cloud services. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1167–1178. IEEE Computer Society, 2015.

[49] P. Sack and W. Gropp. Collective algorithms for multiported torus networks. *ACM Trans. Parallel Comput.*, 1(2):12:1–12:33, 2015.

[50] D. D. Sensi, T. Bonato, D. Saam, and T. Hoefler. Swing: Short-cutting rings for higher bandwidth allreduce. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*. USENIX Association, 2024.

[51] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. A. Hechtman. Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10435–10444, 2018.

[52] D. Skinner and W. Kramer. Understanding the causes of performance variability in hpc workloads. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.*, pages 137–149, 2005.

[53] H. Su, M. Billingsley, and A. D. George. Parallel performance wizard: A performance analysis tool for partitioned global-address-space programming. In *22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida USA, April 14-18, 2008*, pages 1–8. IEEE, 2008.

[54] A. Taherin, T. Patel, G. Georgakoudis, I. Laguna, and D. Tiwari. Examining failures and repairs on supercomputers with multi-gpu compute nodes. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2021, Taipei, Taiwan, June 21-24, 2021*, pages 305–313. IEEE, 2021.

[55] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in MPICH. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, 2005.

[56] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[57] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton-Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023.

[58] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. M. Brandt, V. J. Leung, M. Egele, and A. K. Coskun. Online diagnosis of performance variation in HPC systems using machine learning. *IEEE Trans. Parallel Distributed Syst.*, 30(4):883–896, 2019.

[59] J. Weng, J. H. Wang, J. Yang, and Y. Yang. Root cause analysis of anomalies of multitier services in public clouds. *IEEE/ACM Trans. Netw.*, 26(4):1646–1659, 2018.

[60] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding. Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 945–960. USENIX Association, 2022.

[61] L. Wu, J. Tordsson, E. Elmroth, and O. Kao. Microrca: Root cause localization of performance issues in microservices. In *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*, pages 1–9. IEEE, 2020.

[62] R. Zhang, W. Xiao, H. Zhang, Y. Liu, H. Lin, and M. Yang. An empirical study on program failures of deep learning jobs. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 1159–1170. ACM, 2020.

[63] Y. Zhou, C. Liang, N. Li, C. Yang, S. Zhu, and R. Jin. Robust online matching with user arrival distribution drift. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 459–466. AAAI Press, 2019.

Appendix

A Irregularity statistics

During the three months of tracing LLM training, we manually diagnosed a total of 126 anomalous devices which led to irregularities. As shown in Table 1, the majority were primarily caused by anomalous GPUs, accounting for 58.7%. Anomalous NICs were responsible for 23 occurrences, making up to 18.3%. Additionally, there were other anomalous devices involved, such as links, switches and CPUs. For Figure 3(b), additionally, this increase of irregularity frequency is sub-linear with the increasing number of GPUs. This is attributed to the fact that although the expectation of the number of anomalous devices increases sharply, one irregularity in overall training can be induced by multiple anomalous devices when a large number of GPUs are used.

Table 1: Anomalous devices of irregularity during a three month of LLM training.

Anomalous Device	Count	Percentage
GPUs	74	58.7%
NICs	23	18.3%
Links	12	9.5%
Switches	6	4.8%
CPUs	4	3.2%
Others	7	5.5%

B Periodic feature of fluctuation

The performance fluctuations exhibit an interesting periodic feature. Recall in Figure 6(a), we zoom in the per-iteration times and scrutinize 60 iterations between 50400 and 50460. An observation is that performance fluctuation shows a periodicity, approximately occurs once every 6-9 iterations. At other times, the duration of iterations fluctuates within a small range between 8.18 seconds to 9.45 seconds. To investigate the fluctuation, we map the iteration time into the frequency domain by applying the Discrete Fourier Transform (DFT) and Figure 18 shows the corresponding amplitude spectrum. Significant frequency components can be observed at 0.144, 0.288, and 0.432, which corresponds to periods of 6.94, 3.47, and 2.31 iterations respectively. Although the periodicity may

not be strict and the impact of each performance fluctuation varies, this can indicate that performance fluctuation and abnormal spike are different irregularities.

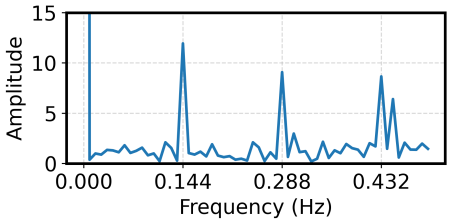


Figure 18: The frequency spectrum.

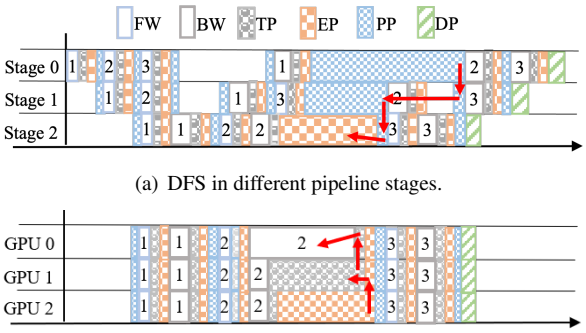
C Algorithm insights

We use two examples to explain our insights on the algorithm.

DFS for point-to-point communication. We illustrate an example in Figure 19(a), where an irregularity occurs in stage 2, leading to the overall training being prolonged. Suppose that the GPU at stage 0 is being monitored, and its "PP" operator has been identified as irregular. It is unclear whether this prolonged "PP" operator is due to the communication link anomaly at stage 0, or if it is influenced by the malfunctions on other GPUs. We then track the same "PP" operator on the downstream stage 1 and find that its execution time is normal, which excludes the inter-stage communication issue between stages 0 and 1. We further backtrack the communication operators on stage 1 and detect an anomalous "PP" operator. The reason for backtracking is intuitive, as the prolonged execution time of one operator is caused by anomalies in preceding operators or in the operator itself. By repeating the above procedure, we move to stage 2 and observe an abnormal "EP" operator, which is different from the previous "PP" operators. The DFS and the corresponding backtracking trajectories are illustrated as the arrow lines in Figure 19(a).

BFS for collective communication. As shown in Figure 19(b), where GPU 0 and GPU 1 are in the same TP group, GPU 1 and GPU 2 are in the same EP group. If the monitored anomalous operator in Holmes is the "EP" operator in GPU 2, we traverse the GPUs in the same EP group and locate the normal "EP" operator in GPU 1. To further diagnose the anomaly, we backtrack to the preceding communication operators and identify an anomalous "TP" operator. By traversing the other GPUs within the same TP group, we locate a normal "TP" operator in GPU 0. Since no other abnormal operators are detected in GPU 0, we can infer that the irregularity is likely caused by GPU 0 computing slowly. The BFS and the corresponding backtracking trajectories are illustrated as the arrow lines in Figure 19(b).

Correlation between operators and nodes Holmes localizes the irregularities with the correlations among operators and correlations between operators and nodes. To be specific, the correlations among operators help us to identify which operator belonging to a collective communication is the root



(a) DFS in different pipeline stages.

(b) BFS in collective communication groups.

Figure 19: Cause localization examples.

cause of the irregularity. The correlations between operators and nodes help us to recursively find the GPU where the irregularity takes place. However, deeper integration of these correlations and Holmes can be further taken into consideration. For example, investigating the local correlation of nodes and their operators. As this is not the main contribution of our paper, we leave this as our further research.

D Device-wise irregularity rate decay.

MAP-based $I(d)$ accumulation. The accumulation formula is shown as the following:

$$I(d) = \Big(\sum_{x \text{ uses } d} \mathcal{R}(x) + a(d)\Big) \Big/ \sum_{d} \sum_{x \text{ uses } d} \mathcal{R}(x) + a(d) + b(d)$$

where Beta distribution $I(d) \sim \mathrm{Be}(a(d), b(d))$ is used as the prior distribution. $a = \alpha \cdot p(d)$, $b = \alpha \cdot (1 - p(d))$, where $p(d)$ is the device irregularity ratio from our production experience and α is a scaling factor.

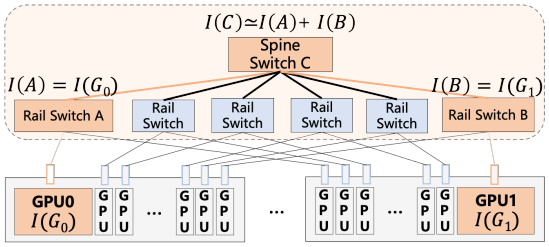
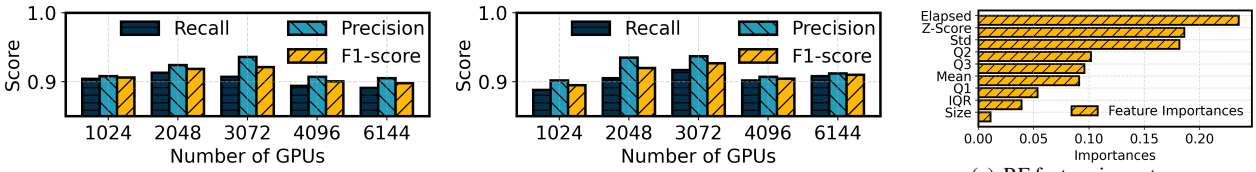


Figure 20: Topology-aware Irregularity rate decay.

In the estimation in § 4.4, we treat each device set equally. However, this could lead to unfairness since the connection of the network fabric is uneven. For example, assuming that irregularities are occurring in both rail switches A and B which slows down the upside transmission. If we traverse the network devices, we could detect irregularities that involve A, B and the spine switch C, and $I(C) \simeq I(A) + I(B)$, as shown in Figure 20. In this case, C will be identified as the most possible device in error. Holmes also designs a device-wise irregularity rate decay mechanism to address this issue. For a



(a) Accuracy with Gini Impurity (b) Accuracy with Entropy (c) RF feature importance

Figure 21: Evaluation on Holmes's detection.

network device d with connections $C(d)$ to downside NICs, we define its decay factor $H(d)$ as follows:

$$H(d) = \frac{\ln(C(d) + \varepsilon - 1)}{C(d)}$$

wherein $\varepsilon = 1e-6$. By substituting the irregularity rate as decayed irregularity rate, the final estimation of device anomalous probability is:

$$I(d) = \frac{\sum_{x \text{ uses } d} \mathcal{R}(x) \cdot H(d) + a(d)}{\sum_d \sum_{x \text{ uses } d} \mathcal{R}(x) \cdot H(d) + a(d) + b(d)}.$$

E Evaluation results

We hereby present the supplemental evaluation results of Holmes. The Network topology used in the testbed is shown in Figure 22.

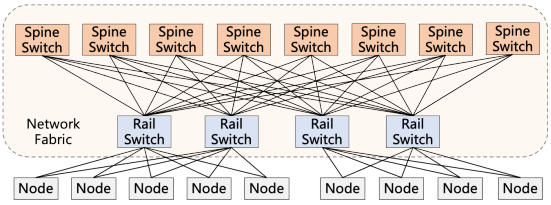


Figure 22: Network topology used in the testbed.

CommOps logger overhead. The communication logger's performance impact is evaluated across multiple training scales (Figure 23). Results show training speed degradation of only 0.064% and 0.083% for 1,024 and 3,072 GPU training when deploying CommOps logger, respectively. Even in the extreme scenario with 6144 GPUs, training time increases by 0.19%, with per-operator logging averaging under $2\mu s$. These metrics confirm the logger's ultra-lightweight design and negligible overhead across cluster scales.

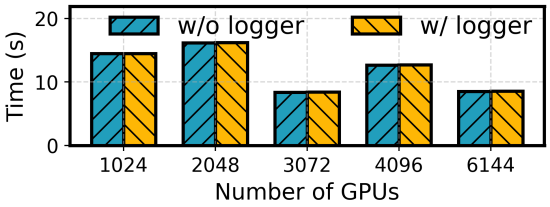


Figure 23: Impact on training with CommOps logger enabled.

Diagnosis of different types of irregularities. We present the diagnosis accuracy of the three types of irregularities across different training scales, as shown in Figure 24. The average diagnosis accuracy of Performance spike, Performance fluctuation, and Performance degradation is 90.7%, 93.0%, 93.2% respectively. Detecting Performance spikes is more challenging because they occur randomly over time, making it difficult to determine when a spike happening. Performance fluctuation and Performance degradation are easy to detect as they typically follow periodic or persistent patterns.

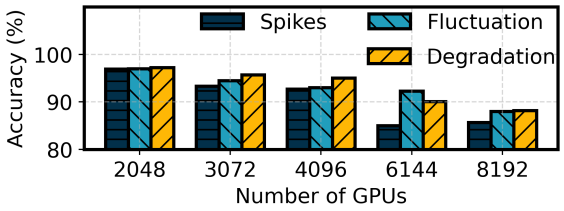


Figure 24: Accuracy for three types of irregularities.

Auto-tuning using different features. We compare Holmes's auto-tuning using different features including P75 (75th-percentile latency), P99 (99th-percentile latency), and mean operator execution time in Figure 25. Experimental results show that tail latency-based tuning (P75 and P99) performs worse than the mean-based strategy, due to their greater sensitivity to irregularities. For example, when training with 8192 GPUs, tuning the detection model using mean operator execution time achieves the highest F1-Score of 0.91, while all the others are less than 0.90. Notably, P99 performs slightly worse than P75, as it is more sensitive to extreme values.

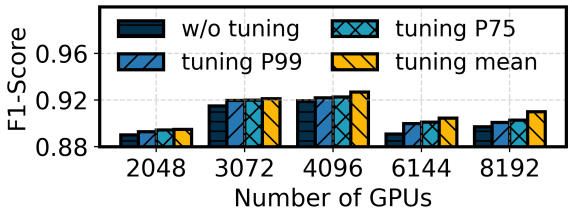


Figure 25: F1-Score of tail latency tuning.

MAP-based irregularity rate accumulation. We conducted comprehensive experiments to compare the accumulation approaches of $R(x)$ in cross-iteration device localization as shown in Figure26. To use the prior device irregularity ratio from our production experience, we employ a MAP-based irregularity rate accumulation approach, while we compare it with the naive linear accumulation without prior. In experiments of different scales, using the prior knowledge is proved

more effective than calculating the sum of the irregularity rate. The greatest improvement was observed in the setup with 8192 GPUs, where the accuracy was 3.3% higher.

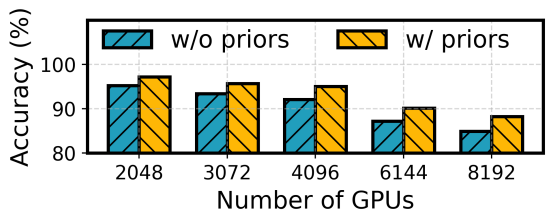


Figure 26: Impact of prior knowledge on localization.

Training throughput. In § 5.3 we illustrate an example of Holmes's localization and find the root cause of irregularity is that GPU 2574 computes slow. As shown in Figure 27. After we isolate the node containing GPU 2574, there are no spikes observed and the average training throughput increases from 85.43 samples/second to 90.62 samples/second.

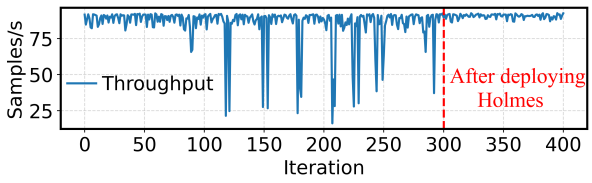


Figure 27: The example training throughput.

F Detailed evaluation of RF

Configuration of baselines of RF models. We compare our RF detection model with other machine learning models like 3-Sigma, KNN, KNN+PCA. The 3-sigma rule we used is performed over each individual operator's own time series. According to the 3-sigma rule, the operator elapsed time within the $(\mu-\sigma,\mu+\sigma)$ range accounts for 68.27% of all its own elapsed time series, and the others can be regarded as abnormal, where μ is the mean and σ is the standard deviation.

Model assessment. We also study the selection of RF models. Figure 21(a) and 21(b) show the Precision, Recall and F1-Score when we use Gini impurity and Entropy as the criterion of RF respectively. In all the evaluated training, the precision is higher than 0.9, while the Recall is higher than 0.85. In particular, we can observe that our RF detector obtains the best F1-Score in training with 3072 GPUs, which is 92.1% and 92.7% for Gini and Entropy. This means that our RF detector could detect the anomalous operators correctly even in large-scale training. Figure 21(c) further illustrates the feature importance distribution within the dataset, highlighting the key factors that contribute to the detection performance of the RF model.