An Empirical Study on Quality Issues of Deep Learning Platform

Yanjie Gao[1], Xiaoxiang Shi[1], Haoxiang Lin[1†], Hongyu Zhang[2], Hao Wu[3], Rui Li[3], Mao Yang[1]
[1]Microsoft Research, Beijing, China [2]Chongqing University, Chongqing, China [3]Microsoft, Beijing, China
Email: {yanjga, v-xiaoxshi, haoxlin, wuh, ruli1, maoyang}@microsoft.com, hyzhang@cqu.edu.cn

***Abstract*—In recent years, deep learning (DL) has been increasingly adopted in many application areas. To help deep learning developers better train and test their models, enterprises have built dedicated, multi-tenant platforms equipped with a mass of computing devices like GPUs. The service quality of these platforms plays a critical role in system efficiency and user experience. Nevertheless, there indeed exist diverse types of quality issues that not only waste computing resources significantly but also slow down development productivity severely. In this paper, we present a comprehensive empirical study on quality issues of Platform-X in Microsoft. Platform-X is an internal production deep learning platform that serves hundreds of developers and researchers. We have manually examined 360 real issues and investigated their common symptoms, root causes, and mitigation actions. Our major findings include: (1) 28.33% of the quality issues are caused by hardware (the GPU, network, and compute node) faults; (2) 28.33% of them result from system-side faults (e.g., system defects and service outages); (3) User-side faults (e.g., user bugs and policy violation) account for more than two-fifths (43.34%) of all the common causes; (4) More than three-fifths of all the quality issues can be mitigated by simply resubmitting jobs (34.72%) and improving user code (24.72%). Our study results provide valuable guidance on promoting the service quality of deep learning platforms from both the development and maintenance aspects. The results further motivate possible research directions and tooling support.**

***Index Terms*—deep learning, deep learning platform, quality issue, empirical study**

I. Introduction

In recent years, deep learning (DL) has been increasingly adopted in many application areas, such as natural language processing [1], [2], gaming with reinforcement learning [3], computer programming [4], and satellite imagery [5]. IT enterprises have built dedicated, multi-tenant platforms (e.g., Microsoft Azure Machine Learning [6], Amazon SageMaker [7], and Google Cloud AI [8]) for providing convenient deep learning training and inference services to developers. These DL platforms are equipped with a large number of computing devices (e.g., CPUs, GPUs, or TPUs) and are internally interconnected with a high-speed network (e.g., via InfiniBand [9]), supporting a variety of DL frameworks and libraries such as PyTorch [10], TensorFlow [11], and Hugging Face [12].

Inside Microsoft, hundreds of developers and researchers use Platform-X, an internal production DL platform, to train and test their models every day for various tasks like advertisement and machine translation. Platform-X is built with widely used open-source software (e.g., Kubernetes [13]) and commodity computing hardware (e.g., GPUs) and is similar in architecture to the above-mentioned public DL platforms. For Platform-X, one of its core tasks is to guarantee the service quality of each user according to her service level agreement (SLA) when facing unexpected hardware and software faults. Although many quality assurance measures are intensively taken, in reality, a number of DL jobs still suffer from severe problems named *quality issues*, such that they cannot be submitted, run very slowly, hang, or even fail accidentally. These quality issues adversely affect not only user experience but also business productivity. When encountering a quality issue, the user reports it via an issue management system and expects site reliability engineers (SREs) to diagnose and resolve it as soon as possible, which also brings a heavy operational and maintenance burden to the platform support team. Therefore, understanding the quality issues raised in Platform-X, including their symptoms, root causes, and mitigation actions, becomes particularly important to help the platform support team maximize efficiency and help the platform engineering team improve system design and implementation.

There has been much previous work on the quality issues of conventional software systems [14]–[22]. For example, Zhou et al. [19] studied 210 randomly selected quality issues from an internal big data analytics platform of Microsoft. Recently, we have seen some deep-learning-related empirical studies [23]–[31] focusing on the failures and bugs of DL frameworks, compilers, programs, and jobs. For instance, Zhang et al. [27] researched 4,960 internal DL job failures collected from Microsoft within a three-week period. However, there is still a lack of specialized studies on the characteristics of DL platforms' quality issues.

In this paper, we present a comprehensive empirical study on quality issues of deep learning platform. We selected 360 quality issues randomly from the issue management system of Platform-X, which were reported by developers and researchers from November 2021 to February 2022. For each quality issue, we collected its related information, including, for example, the issue description, discussions, root cause, mitigation action, final fix solution, and so on. If possible, we also collected relevant information about the impacted job. More details are presented in Section IV-A. The purpose of the study is to provide a systematic and generalized understanding of DL platforms' quality issues. Platform-X has a similar system architecture, software stack, deep learning toolchain,

†Corresponding author.

job management, and quality issue handling to those of other companies in the industry. As a result, the findings of this study are not specific to Microsoft and could be applied to the deep learning platforms of other companies. Specifically, our study aims to address the following three research questions (RQs):

RQ1: What are the common symptoms of the quality issues in a deep learning platform?
RQ2: What are the common root causes of these quality issues?
RQ3: What are the common mitigation actions adopted by the platform support team?

We obtain many findings, and some of the major ones include:

1) Hardware faults account for nearly one-third (28.33%) of all common causes and result from GPUs, networks, and compute nodes.
2) Platform-side faults also account for nearly one-third (28.33%) of all common causes, among which System Defect, Resource Overload, and Platform Maintenance are the top three out of six categories.
3) User-side faults are more than two-fifths (43.34%) of all common causes, among which Buggy Code, Policy Violation, and Improper Permission are the top three out of five categories.
4) There are ten categories of mitigation actions. More than three-fifths of the total quality issues can be mitigated by Job Resubmission (34.72%) and User Code Improvement (24.72%).

Our findings would better guide the design and management of deep learning platforms to reduce quality issues and enhance reliability. They would also assist both platform development engineers and site reliability engineers in improving their daily development and operational activities. Based on the study results, we point out possible new directions for future research.

To summarize, this paper makes the following contributions:

1) We perform the first comprehensive study on quality issues of a production deep learning platform. We examine 360 quality issues and manually analyze their symptoms, root causes, and mitigation actions.
2) We point out the implications of our findings and suggest possible improvements for the development and operations of deep learning platforms.

The rest of the paper is organized as follows. In Section II, we give an overview of the deep learning platform Platform-X and the job life cycle. Section III introduces how quality issues are handled in Platform-X. Section IV presents the study methodology. In Section V, Section VI, and Section VII, we describe the common symptoms, root causes, and mitigation actions of the quality issues, respectively. Section VIII discusses the generality of our study and future research directions. We survey related work in Section IX and conclude this paper in Section X.

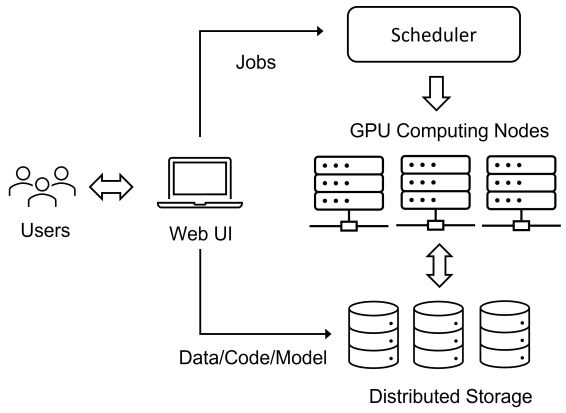

Fig. 1. Overview of Platform-X.

II. Background

A. Deep Learning

Deep learning (DL) has rapidly emerged as one of the most successful machine learning techniques. A DL model (aka deep neural network) is a layered data representation [32], being learned from massive input data. To write programs, developers make use of DL frameworks such as ONNX [33], PyTorch [10], and TensorFlow [11], as well as the Python programming language. These frameworks formalize a DL model as a tensor-oriented computation graph (i.e., a directed acyclic graph). A tensor is a multi-dimensional array. Each node on such a graph denotes a mathematical operation called an operator, which manipulates one or more tensors. Example operators include matrix product, rectified linear unit, 2D convolution, and linear transformation. A graph edge delivers an output tensor of the source node A to the destination node B as input and specifies that B can begin execution only when A has finished.

B. Microsoft Platform-X

Platform-X is the internal production DL platform of Microsoft, serving hundreds of developers and researchers. Every day, users submit and execute thousands of DL jobs on Platform-X for their routine work, such as machine translation, gaming, object detection, and advertisement.

Microsoft builds Platform-X with comparable computing hardware and widely used open-source software. For example, Platform-X is deployed on multiple physical GPU clusters and employs Kubernetes [13] for task planning, container orchestration, and heterogeneous hardware management. Platform-X also adopts a standard DL programming paradigm: it prepares diverse standard DL Docker [34] images composed of Python, the NVIDIA CUDA/cuDNN/cuBLAS runtime, DL frameworks such as PyTorch [10] and TensorFlow [11], and other popular libraries like Fairseq [35] to establish a hermetic job execution environment.

Figure 1 illustrates the workflow of Platform-X briefly. The system architecture and job management of Platform-X are very similar to those in public DL platforms such as Microsoft Azure Machine Learning [6], Amazon SageMaker [7], and Google Cloud AI [8]. Users first operate the web portal or

TABLE I
TELEMETRY DATA FOR QUALITY ISSUE DIAGNOSIS.

Dimension	Category	Examples
Job Metadata	Time	job submission time, job/process start and end times, job queuing period, job/process execution period
	Resource	specification/number of GPU, CPU, and input/output storage
	Software	GPU driver, OS, NVIDIA runtime, Python, framework, libraries, and their versions
Performance Metrics	Computing	average and peak utilization of GPU/CPU
	Memory	total/available sizes and average/peak utilization of the main and GPU memory
	Disk	total size, available size, read/write bytes
	Network	sent/received bytes of InfiniBand and Ethernet
	Node	healthy or faulty state
Runtime Logs	User	user-specified logs in the source code, containing execution progress, input data size, epoch number, batch size, accuracy, loss, model type, etc.
	System	logs of runtimes, system components, and hardware drivers, such as "NCCL INFO NET/IB: Using [0]mlx_x:xx"
	Failure	exception and error messages, such as "uncorrectable NVLink error detected"

command line tool to upload all the materials, including the input data, Python programs, shell scripts, and possibly model checkpoints, to the distributed storage (e.g., Azure Blobs[1]). Next, users specify the resource quota (e.g., the GPU model and number), Docker image, startup shell script, main Python file, input/output paths, and other configurations for their jobs. Users can also specify custom Docker images preinstalled with all the dependent libraries. Once a submitted job is chosen to run, the scheduler of Platform-X allocates all the requested resources at once using the gang scheduling [36] algorithm and instantiates containers on one or more GPU compute nodes. A compute node in Platform-X is a physical server or a virtual machine equipped with GPUs, CPUs, main memory, disks, and network interface cards. Afterward, the DL training code of such a job iteratively updates the learnable parameters (i.e., weights and biases) until the model learning performance (e.g., predictive accuracy) achieves our expectation. Finally, when the model training finishes, the job saves the final model files and evaluation results to the distributed storage.

III. QUALITY ISSUE HANDLING IN PLATFORM-X

Platform-X adopts a standard and well-defined process for handling quality issues. Firstly, users log in to the issue management system of Platform-X from a web portal and create an issue item. They follow one of the provided templates to describe the issue in detail, including, for example, the title, issue description, proposed severity level, job URL, failure messages, and custom Docker image tag. More supplementary information, such as runtime logs and performance metrics, is encouraged to be submitted as attachments.

The issue management system then delivers the new quality issue to an appropriate *site reliability engineer* (SRE)[2] for investigation based on the issue type and historical statistics. The SRE invokes an integrated tool to automatically discover previous duplicate, similar, or related quality issues. If none is found, the SRE will follow a formal troubleshooting guide to investigate the issue step by step in a top-down process. Essentially, the SRE starts from a failure or abnormal site of the impacted job process(es) by reviewing the error messages or exceptional performance metrics. After analyzing the programs and runtime logs, she tries to reason a critical path of the problem. Guided by the troubleshooting decision rules, the SRE dives into the bottleneck or failed stages in the critical path to identify the root cause.

Platform-X has recorded various kinds of telemetry data for the above issue diagnosis. These data contain job metadata, performance metrics, and runtime logs, most of which are listed in Table I. Job metadata include the time statistics (e.g., job/process start and end times), allocated resources (e.g., the GPU specification and number), and dependent software. Performance metrics include the usage of various resources (e.g., the average utilization of GPU/CPU) and the node state (i.e., healthy or faulty). Runtime logs include regular and failure messages printed by user code, runtimes, system components, and hardware drivers.

Next, the SRE adjusts the severity level and proposes a mitigation action by the service level agreement (SLA) to keep the user's work going. The mitigation comes from the referred actions for previous duplicate/similar/related quality issues, instructions in the troubleshooting guide, and her domain knowledge. The SRE regularly contacts the user via Microsoft Teams[3] or telephone to discuss the details, seek clarification, and report progress. Each interaction with the user is noted down carefully in the issue record.

In the end, the SRE investigates the final fix solution if the quality issue is caused by hardware or platform-side faults (Section VI-A and Section VI-B). Sometimes, the issue is beyond the responsibility of Platform-X (e.g., caused by a completely damaged compute node or an outage of the distributed storage), so she transfers its ownership to the corresponding team. After closing the quality issue, the SRE also updates the troubleshooting guide to facilitate the future handling of similar ones.

[1]https://azure.microsoft.com/en-us/products/storage/blobs

[2]https://en.wikipedia.org/wiki/Site_reliability_engineering

[3]https://www.microsoft.com/en-us/microsoft-teams/group-chat-software

IV. Empirical Study Methodology

A. Study Subjects

We consider the real quality issues of Platform-X as our study subjects. They were submitted by developers and researchers during a four-month period from November 2021 to February 2022. There existed duplicate quality issues because multiple users were simultaneously affected by a single incident (e.g., cluster unavailability); thus, we performed issue deduplication. In the end, we selected 360 quality issues randomly out of the complete set.

For each quality issue, we collected all its related information for later investigation, including, for example, the username, group, cluster, job URL, issue title, description, attachments, timestamps of issue status update, severity level, discussion details, mitigation action, root cause, related issues, and final fix solution. If the job URL existed, we further tried to pull the job metadata (e.g., the GPU number and final job status), execution logs, and various runtime metrics (e.g., the GPU utilization and network sent/received bytes).

B. Data Labeling

For each of the 360 quality issues, we carefully examined its related information (in particular, the issue description and discussions) and manually labeled the symptom, root cause, and mitigation action to answer the three research questions. To reduce the inevitable subjectivity, two authors read all the issue-related information independently for the data labeling. We first extracted key sentences and phrases for the above three dimensions, and then we refined them into the classification schema by the existing ones of related studies (such as those by Zhou et al. [19] and Zhang et al. [27]) and our domain knowledge. When unsure of certain details, we contacted the issue submitter and corresponding SRE directly for help.

C. Threats to Validity

Threats to Internal Validity. Although there are some reference materials (such as the issue description and user-SRE discussions) for facilitating the investigation of the symptoms, root causes, and mitigation actions, subjectivity is inevitable due to a mass of manual effort and the inherent complexity of deep learning quality issues. To reduce such a threat, two of the authors independently analyzed and labeled each of the 360 quality issues. We also strived to reach a group consensus before making decisions in case of disagreement. When meeting a complicated issue, we contacted the submitter and corresponding SRE directly to seek help.

Threats to External Validity. We perform our empirical study on quality issues collected from Platform-X. Therefore, certain findings may be restricted to Microsoft and may no longer hold in the deep learning platforms of other companies, although Platform-X is similar to them in the system architecture, software stack, deep learning toolchain, job management, and quality issue handling. To reduce this threat, we try not to draw particular conclusions that only apply to Platform-X and Microsoft in this paper. We will discuss the generality of our findings in Section VIII-A.

TABLE II
Classification of common symptoms.

Category	No.	Ratio
Job Crash	198	55.00%
Job Submission Failure	51	14.17%
Abnormal Job Behavior	46	12.78%
Job Hang	27	7.50%
Cluster Unavailability	21	5.83%
Job Slowdown	11	3.05%
Data Loss	6	1.67%
Total	360	100.00%

V. What are the common symptoms?

In this section, we study the common symptoms of the quality issues. A symptom is the subjective evidence of a quality issue observed by users, which can be discovered in the issue title and description. Table II presents the symptom classification, including seven categories in total.

Since users care more about their DL jobs, the overwhelming majority of the symptoms (five categories; 333; 92.50%) are related to specific jobs. The largest category is *Job Crash* (198; 55.00%), more than half the total. For example, a 32-GPU distributed training job ran for a long time, suddenly printed an error message complaining that the NVIDIA Collective Communications Library (NCCL) [37] timed out, and then crashed. The remaining four job-related categories, ordered in number, are *Job Submission Failure* (51; 14.17%), *Abnormal Job Behavior* (46; 12.78%), *Job Hang* (27; 7.50%), and *Job Slowdown* (11; 3.05%). Abnormal Job Behavior means that a DL job seems to be executing fine, but it exhibits some unusual behaviors. For instance, a user noticed that only one of the four allocated GPUs worked as expected, yet the others had remained idle since the job started.

Two categories of the symptoms are not directly related to a specific DL job. One is *Cluster Unavailability* (21; 5.83%), which means that the GPU cluster cannot provide continuous service for job submission and execution anymore (e.g., due to an outage or cluster maintenance). For example, a user found that some GPUs of a cluster were unavailable from Platform-X's web portal, and any job submission by him and other colleagues to such a cluster continued to fail. The other is *Data Loss* (6; 1.67%), indicating that users lost their data suddenly and inexplicably. For instance, an input data folder on the distributed storage disappeared without warning. Another example is that a user could not query the telemetry data of his historical DL jobs.

Finding 1: Almost all the common symptoms (92.50%) are directly related to specific deep learning jobs, among which the largest category is Job Crash (55.00%), more than half the total.
Implication: Failed DL jobs waste lots of computing resources and time. Therefore, users and the platform should work together to reduce and tolerate job failures, such as improving the user code and implementing a more effective model checkpointing technique.

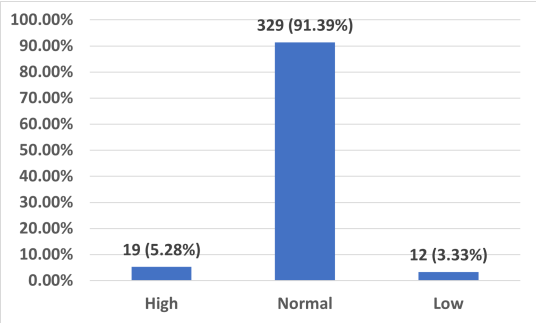

Fig. 2. Severity of quality issues.

TABLE III
OVERALL CLASSIFICATION OF COMMON ROOT CAUSES.

Dimension	No.	Ratio
Hardware Fault	102	28.33%
Platform-side Fault	102	28.33%
User-side Fault	156	43.34%
Total	360	100.00%

Furthermore, we classify the severity of the quality issues. As mentioned in Section III, the user needs to claim a severity level based on the business impact when submitting a quality issue to the Platform-X support team. The issue management system and SREs refer to the user-proposed severity to prioritize the issue handling (e.g., determining the required time of a mitigation action). Note that the SRE may adjust the severity level according to the previous/similar quality issues and her domain knowledge to reflect a more precise situation. Platform-X adopts three severity levels: High, Normal, and Low. Figure 2 shows the severity distribution among all the 360 quality issues. We observe that the vast majority (329; 91.39%) of the quality issues have the Normal level. Only 19 (5.27%) quality issues are High-level, which require that the support team starts dealing with them in a very short period, updates the progress with the users timely and regularly, and applies successful mitigation actions within an allotted time. The remaining 12 (3.33%) issues are at the Low level.

VI. WHAT ARE THE COMMON ROOT CAUSES?

In this section, we present the classification of the common root causes of the 360 quality issues. Table IV, Table V, and Table VI summarize twenty-two categories in total. We also group them into three major dimensions: Hardware, Platform-side, and User-side. Table III shows an overall distribution across these dimensions.

A. Hardware Faults

Platform-X is built with heterogeneous commodity hardware such as NVIDIA GPUs and InfiniBand networking, which may encounter a relatively high probability of hardware malfunction. Table III demonstrates that the hardware fault is a major type of root cause, resulting in nearly one-third (102; 28.33%) of all the quality issues. There are eleven categories of hardware faults in total, and we further divide them into three groups: GPU, Network, and Node. The detailed classification and distribution of the hardware faults are illustrated in Table IV.

TABLE IV
CLASSIFICATION OF HARDWARE FAULTS.

Group	Category	No.	Ratio
GPU	GPU Memory Fault	9	2.50%
	GPU Unavailability	5	1.39%
	NVLink Error	4	1.11%
	Broken GPU Driver	2	0.55%
	Subtotal	20	5.55%
Network	InfiniBand Port Down	6	1.67%
	InfiniBand Slowdown	5	1.39%
	InfiniBand Other Faults	6	1.67%
	Ethernet Fault	8	2.22%
	Subtotal	25	6.95%
Node	Node Outage	50	13.89%
	Node Damage	5	1.39%
	Node Preemption	2	0.55%
	Subtotal	57	15.83%
Subtotal		102	28.33%

GPUs are the primary computing devices for deep learning jobs. The long and heavy workload could lead to various *GPU faults* [38]–[41]. The first category is *GPU Memory Fault*, meaning that 9 (2.50%) quality issues are caused by faulty GPU memory. Typical cases include uncorrectable ECC (error correction code) errors, GPU MMU (memory management unit) errors, illegal memory accesses, and memory leaks. For example, a long-running distributed training job with 56 GPUs suddenly threw a CUDA runtime error and failed. After using the NVIDIA Data Center GPU Manager (DCGM) [42] for further diagnosis, the SRE finally identified that the root cause of such a job failure was one GPU raising an ECC error. Sometimes, a DL job cannot find or register the allocated GPUs because a *GPU Unavailability* fault is raised. This category has 5 (1.39%) quality issues. The third category is *NVLink Error*, which consists of 4 (1.11%) quality issues. The NVIDIA NVLink is "a direct GPU-to-GPU interconnect that scales multi-GPU input/output (IO)" [43] within a compute node for distributed DL training. Figure 3 shows the failure logs of an example NVLink error. In the fourth and last category, two (0.55%) quality issues root in *Broken GPU Driver*, which usually needs node restart or driver reinstallation because the NVIDIA GPU driver is not working appropriately.

Distributed deep learning training across multiple compute nodes is pretty common in Platform-X. These nodes are internally interconnected with a high-speed network (e.g., via InfiniBand [9]). We observe that 25 (6.95%) quality issues are caused by four categories of *network faults*, among which three categories are InfiniBand-related. InfiniBand is "a computer networking communications standard used in high-performance computing that features very high throughput and very low latency" [9] and is widely used in a variety of

```
1 ...
2 terminate called after throwing an instance of
  ↪  'c10::Error'
3   what(): CUDA error: uncorrectable NVLink error
    ↪  detected during the execution
4 Exception raised from create_event_internal at
  ↪  ../c10/cuda/CUDACachingAllocator.cpp:687 (most
  ↪  recent call first)
5 ...
6 c10:cuda:CUDACachingAllocator::raw_delete(void *)
  ↪  ... in /opt/.../torch/lib/libc10_cuda.so)
```

Fig. 3. Failure logs of an NVLink Error example. The job crashed when calling the CUDACachineAllocator::raw_delete() function.

cloud-based platforms. *InfiniBand Port Down* is the largest category, containing 6 (1.67%) cases. It means that there is no physical connection between two InfiniBand Host Channel Adapters of a cable, and thus runtime errors of the NVIDIA Collective Communications Library (NCCL) [37] are triggered. The second category is *InfiniBand Slowdown* (5; 1.39%), which means that the InfiniBand adapter is extremely slow, sometimes an order of magnitude slower than normal. As a result, the DL jobs also slow down or even get stuck. There are 6 (1.67%) *other InfiniBand faults*, including adapter initialization failures, broken driver, write transaction faults, and memory registration failures. The fourth root cause category, *Ethernet Fault*, is tied for the largest one. Ethernet is also intensively used in Platform-X for connecting the distributed storage and internal services and for facilitating users to debug their failed jobs via the secure shell protocol (SSH). This category has 8 (2.22%) cases, including, for example, transient Ethernet failures, name resolution errors, and connection reset by peer.

A compute node (or node for short) in Platform-X is a distinct schedulable unit for computation with GPUs, CPUs, main memory, disks, and network interface cards. It can be a physical server or a virtual machine (VM). Since Platform-X uses commodity hardware, node faults are unavoidable and lead to 57 (15.83%) quality issues, more than the sum of the other two groups. 50 (13.89%) of them are *Node Outages*, such as OS kernel panic and ephemeral disk errors, causing the DL jobs to fail or hang. To be noted, ephemeral disks are created and attached to a compute node as temporary storage (e.g., a job pulling and storing its remote input data for later fast data access). This category is the largest in the node group, far more than the other two. The *Node Damage* category consists of 5 (1.39%) cases. The Platform-X support team will immediately de-commit (i.e., recall) a completely damaged node from its belonging cluster and hand it over to a dedicated hardware support team for further repairs. The third and last category is *Node Preemption*, which has only 2 (0.55%) cases. The users applied for *spot nodes* [44] (i.e., currently unused nodes at significant cost savings); however, they were unconscious that Platform-X terminated their jobs and reclaimed the spot nodes for others who paid regular fees.

TABLE V
CLASSIFICATION OF PLATFORM-SIDE FAULTS.

Category	No.	Ratio
System Defect	37	10.28%
Resource Overload	21	5.83%
Platform Maintenance	14	3.89%
Transient Service Outage	11	3.05%
Resource Contention	10	2.78%
Regression	9	2.50%
Subtotal	102	28.33%

Finding 2: Hardware faults account for nearly one-third (28.33%) of all the common causes and result from the GPU, network, and compute node.
Implication: Hardware faults are inevitable in deep learning platforms. Proactive hardware testing (e.g., fault injection [45] and stress testing [46]) and fault prediction (e.g., using machine learning prediction models [40], [41]) techniques need to be developed and applied.

B. Platform-side Faults

In this section, we describe the study on platform-side faults, which consist of 102 (28.33%) cases and are further classified into six categories. Table V shows the detailed classification and distribution.

System Defect is the largest category and results in 37 (10.28%) quality issues. Typical defects include:

1) Code bugs. For example, Platform-X forgot to package a dependent library into an official Docker image. As another example, some DL jobs could not access the distributed storage due to a bug in the storage service code.
2) Service misconfiguration. For instance, a DL job failed because of a misconfigured domain name system (DNS) service.
3) Wrong access control. As mentioned, an input data folder on the distributed storage disappeared without warning. The root cause is that the access control of such a folder was mistakenly set by Platform-X, and thus another colleague in the same team deleted it accidentally.

For certain system resources, Platform-X does not have explicit quota control on users and DL jobs. Therefore, if users are unaware of such a system design limitation and employ a resource excessively, *Resource Overload* faults are triggered. There are 21 (5.83%) cases in this category. For instance, a user tried to load all the input data into the main memory for optimal performance. Nevertheless, the data was overlarge, and the main memory soon ran out. As another example, it is a common practice for users to store their temporary data (such as model checkpoints, evaluation results, software cache, or even input data) on the local disks of compute nodes. Sometimes, users forget to clean out old files, and the disk space is quickly used up.

Since multiple DL jobs may compete for the same resources on a compute node, they could interfere with each other and trigger *Resource Contention* faults. This category includes 10 (2.78%) cases. For instance, a DL job experienced an unexpected slowdown. The SRE discovered that such a job received far less CPU time than usual because another job created too many CPU tasks at the same time. Resource Contention is similar to the above Resource Overload, and both can be alleviated by adopting a more proper system design (e.g., applying resource isolation and quota control).

Platform-X performs regular *Platform Maintenance* on GPU clusters for hardware replacement, node reimaging, software upgrade, and other tasks, which leads to 14 (3.89%) quality issues. If users do not carry on proactive job migration, existing running DL jobs will be terminated automatically by Platform-X. Occasionally, some users are unaware of the maintenance notification, and thus they fail to submit jobs persistently.

The fifth category is *Transient Service Outage*, which accounts for 3.05% (11) of all the root causes. For example, a user was not able to connect to the allocated compute nodes because the SSH service was temporarily unavailable. These service outages were transient and could be automatically recovered after a while. However, we do not have further details to reason more fundamental causes.

Regression in the tools, runtimes, and services of Platform-X causes 9 (2.50%) quality issues. For example, an updated job submission tool changed certain environment variables and thus broke backward compatibility. The regression comes from the roll-out policy adopted by Platform-X. At present, new features and updates are incrementally rolled out, and Platform-X enlarges the deployment scope gradually. Therefore, both old and new versions have to be maintained, which requires users to pay attention. Once regression happens, the user could roll the software or service back to an older version for issue mitigation.

Finding 3: Platform-side faults also account for nearly one-third (28.33%) of all the common causes, among which System Defect, Resource Overload, and Platform Maintenance are the top three out of six categories.
Implication: Platforms should improve the system design and implementation to deliver a better deep learning service. Various testing techniques (such as formal/stress/regression testing) would help expose the platform-side faults as early as possible.

C. User-side Faults

Although it is usually thought that users should not report any problems of their own making, we are surprised to notice that 156 (43.34%) quality issues are actually caused by user-side faults. We further classify them into five categories and show the details in Table VI.

The first and largest category is *Buggy Code*, which involves 54 (15.00%) cases and is nearly half of the total user-side faults. These code bugs exist in deep learning programs, Shell scripts, configuration files, and custom Dockerfiles, many of which have been mentioned in the empirical study by Zhang et al. [27]. For example, a number of bugs came from "the discrepancies between local and platform execution environments." [27] Figure 4 shows that a DL job accessed a nonexistent local folder for TensorBoard [47] visualization. It seems that the user forgot to adapt the code and use a path on Platform-X. Due to the complexity of Platform-X's execution environment, users should adopt more defensive programming to handle various potential faults that may not be exposed on their local development machines. For instance, we noticed that a job crashed because of exceptional data in an unexpected format. Better practices include cleansing the dataset proactively before job submission and sampling data for local testing. Some jobs rely on external services but neglect to deal with service unavailability appropriately; for example, they failed to pull external Docker images, models, or datasets. Hugging Face [12] provides tools for users to build state-of-the-art AI applications from "the reference open source in machine learning," whose pre-trained models and datasets are stored in its own repository. Figure 5 demonstrates a bad-gateway failure in which a DL job could not download a required Hugging Face model. In order to reduce the probability of external services being unavailable, users are encouraged to switch to equivalent internal services instead (e.g., uploading the Hugging Face model to the internal distributed storage in advance).

We further found some performance bugs not mentioned by Zhang et al. [27] since they studied only deep learning program

TABLE VI
CLASSIFICATION OF USER-SIDE FAULTS.

Category	No.	Ratio
Buggy Code	54	15.00%
Policy Violation	42	11.66%
Improper Permission	35	9.72%
Software Incompatibility	14	3.89%
Misoperation	11	3.05%
Subtotal	156	43.34%

```
1 ...
2 files = [f for f in listdir(logdir) if
  ↪  isfile(join(logdir, f))]
3 FileNotFoundError: [Errno 2] No such file or
  ↪  directory: '~/tensorboard/xxx'
4 ...
```

Fig. 4. Failure logs of a Buggy Code example. The job forgot to adapt the code and still used a nonexistent local folder for TensorBoard visualization.

```
1 ...
2 raise HTTPError(http_error_msg, response=self)
3 requests.exceptions.HTTPError: 502 Server Error:
  ↪  Bad Gateway for url:
  ↪  https://huggingface.co/xxx.model
4 ...
```

Fig. 5. Failure logs of another Buggy Code example. The job was unable to download a required model from Hugging Face and forgot to handle such a service-unavailability failure.

failures. For instance, a job read a lot of small data files from the distributed storage, slowing down the training process seriously. As another example, a multi-GPU training job ran very slowly because of the potential contention between threads. We also noticed some software hang bugs [48], [49] that kept the jobs going nowhere; for example, one job misconfigured InfiniBand in the source code and hung on the NVIDIA NCCL communication.

Platform-X enforces many policy rules to guarantee that users employ precious platform resources more properly and efficiently. Although Platform-X has provided documentation and training courses for policy explanation, some users are still unaware of those rules, which triggers *Policy Violation* faults and leads to 42 (11.66%) quality issues. This category is the second largest and accounts for about a quarter of all the user-side faults. As an example, Platform-X automatically killed user jobs whose GPUs had been idle for a prescribed period of time to reduce resource waste and maximize platform utilization. We also noticed that some users submitted *interactive* deep learning programs (e.g., Jupyter [50] notebooks) to Platform-X for development and testing purposes. Because of the non-deterministic interaction, it was impossible to know when and how long the GPUs would be used in advance. Finally, Platform-X killed these interactive jobs for being idle too long. Other violated policy rules include, for example, the data on Platform-X's temporary storage expiring after a few days (so users need to move them to persistent storage as soon as possible) and a DL job not initiating too many connections to certain internal services. Violation of the latter triggers service throttling and leads to job slowdown or even service termination.

Users and their DL jobs need appropriate permissions to access the resources of Platform-X; otherwise, *Improper Permission* faults are raised, resulting in 35 (9.72%) quality issues. For instance, a user could not submit any job to a certain GPU cluster. In fact, his request for cluster access was still being processed, and thus the user had no access permission at that moment. Another example is that a DL job failed to pull a Docker image from the hub due to missing a correct credential.

The fourth category is *Software Incompatibility*, consisting of 14 (3.89%) cases. With the increasing adoption of deep learning in many application areas, DL-related software, such as NVIDIA runtimes, frameworks, libraries, and optimization toolkits, has been rapidly evolving recently. However, since they are independently developed by different communities, incompatibility may happen between mismatched components. For example, a job got stuck because the standard DL Docker image used a lower version of the NVIDIA CUDA runtime than that for local development. Commonly, DL jobs install dependent libraries during the initialization stage. If users forget to specify library versions explicitly, they may get the latest software not yet tested by themselves, which could trigger software incompatibility. For instance, a job installed a newer incompatible version of Microsoft DeepSpeed [51] (an optimization toolkit) and then experienced an apparent slowdown due to the degraded network throughput of the compute nodes. To reduce software incompatibility, users need a deeper understanding of various DL software components and use custom Docker images with all dependent libraries pre-installed.

Misoperations of users cause 11 (3.05%) quality issues because they may not be familiar with the operation process. For instance, a user applied a wrong filter when querying information on the web portal and thus received unexpected results. Another example is that a user deleted his workspace inadvertently by mistake. A workspace provides "a centralized place to work with all the artifacts you create." [52] Therefore, the user lost the history of all his training jobs without even knowing that.

TABLE VII
CLASSIFICATION OF COMMON MITIGATION ACTIONS.

Category	No.	Ratio
Job Resubmission	125	34.72%
User Code Improvement	89	24.72%
Operation Correction	26	7.22%
System Reconfiguration	23	6.39%
Software Rollback	22	6.11%
Automatic Healing	22	6.11%
System Hotfix	19	5.28%
Node De-commission	5	1.39%
Job Killing	4	1.11%
Quota Increase	3	0.84%
Others*	22	6.11%
Total	360	100.00%

* "Others" means that the quality issues lack explicit details on how they were mitigated.

Finding 4: User-side faults account for more than two-fifths (43.34%) of all the common causes, among which Buggy Code, Policy Violation, and Improper Permission are the top three out of five categories.
Implication: Users should improve deep learning code, refer to documentation, and participate in training to reduce the faults made by themselves. Static analysis tools could be developed to detect user-side faults automatically as early as possible.

VII. FROM QUALITY ISSUES TO MITIGATION ACTIONS

Site reliability engineers (SREs) have a responsibility to apply mitigation actions quickly when new quality issues are reported. The purpose of a mitigation action is to renew the impacted job back to work or recover the services of Platform-X in the shortest possible time; otherwise, business is adversely affected, and precious resources are significantly wasted. The mitigation is usually a workaround instead of a final fix solution because the latter requires a long time of thorough investigation, error-free implementation, and extensive testing and thus may not be affordable.

In this section, we study the common mitigation actions adopted by SREs and classify them into ten categories. Table VII shows the classification details. Note that 22 (6.11%)

quality issues lack explicit details on how they were mitigated; therefore, we mark their mitigation actions as *Others*.

Job Resubmission is the largest category, containing 125 (34.72%) cases. As we have seen in Section VI, many job failures and slowdowns are caused by various hardware and platform-side faults, most of which actually affect only one or a few compute nodes. Once SREs identify a faulty node, or Platform-X detects one automatically, it is de-committed from the belonging cluster for offline repair. Therefore, resubmitting the impacted job without modifying any configurations and parameters will most likely avoid the faulty nodes and can finish the job successfully.

User Code Improvement is the second largest category and applies to 89 (24.72%) quality issues. Many are caused by user-side faults, which should not be reported to the platform support team indeed. However, in order not to hinder our business, SREs are active in helping users refine their code, access permission, and submission parameters. For example, an SRE instructed the user to increase the NCCL timeout value in the source code for an InfiniBand-related issue. For part of the issues caused by Resource Overload and Resource Contention, code improvement is also an effective mitigation method.

26 (7.22%) quality issues due to Misoperation, Policy Violation, and Improper Permission can be mitigated by *Operation Correction*. For example, as we mentioned, a user applied the wrong query filter. The SRE instructed the user on how to query information properly on the web portal and suggested a correct filter.

The *System Reconfiguration* category includes 23 (6.39%) quality issues, which are mainly caused by Resource Overload, Policy Violation, and Improper Permission. SREs need to reconfigure the access permission, policy rules, or service parameters. For instance, an issue reported that the storage ran out. To mitigate it, the SRE and cluster administrator increased the total storage volume and performed data cleaning.

22 (6.11%) quality issues caused by Software Incompatibility, Regression, and System Defect (one case) are mitigated by *Software Rollback* to the last version that worked. For the system tools, components, and services, Platform-X preserves several latest working versions.

Some service outages and hardware faults are transient or can be automatically recovered by Platform-X. Therefore, 22 (6.11%) quality issues are mitigated by *Automatic Healing*, meaning that the users do not need to perform specific actions.

Many system defects may show a potential broad influence and do not have simple mitigation actions. Therefore, the support teams of Platform-X and other system services work together to deliver *System Hotfixes* as quickly as possible. For instance, an OS upgrade caused a storage service not to work properly because of a minor incompatibility issue. The support team fixed the incompatibility soon and then applied a system hotfix. This category has 19 (5.28%) cases in total. Note that hotfixes need further verification and stress testing before they become official system patches.

The remaining three smaller categories are:

1) *Node De-commission* (5; 1.39%), which eliminates a faulty compute node immediately from the cluster and applies to Node Damage faults only.
2) *Job Killing* (4; 1.11%), which mitigates zombie jobs that use up all the resources and stop further job submission.
3) *Quota Increase* (3; 0.84%), which requires that users increase the quota of certain resources (e.g., the main memory) to resolve Resource Overload and Resource Contention faults.

Finding 5: There are ten categories of mitigation actions. More than three-fifths of the total quality issues are mitigated by Job Resubmission (34.72%) and User Code Improvement (24.72%).
Implication: Auto-recommendation tools based on historical statistics could be developed to automate and accelerate the mitigation process.

VIII. Discussion

A. Generality of Our Study

Our study is exclusively conducted in Microsoft; however, we believe that the selected deep learning quality issues are common, and the study results can be generalized to other DL platforms, such as Microsoft Azure Machine Learning [6], Amazon SageMaker [7], and Google Cloud AI [8]. The key reason is the twofold similarity between Platform-X and others:

1) Platform-X is built with the widely adopted hardware, system architecture, and software stack [26], [53], [54], analogical to those of other platforms. In addition, Platform-X employs a similar mechanism of job management (e.g., submission and execution).
2) The DL jobs running on Platform-X adopt a standard programming paradigm. For example, they are programmed with the Python language, popular frameworks (e.g., ONNX [33], PyTorch [10], and TensorFlow [11]) and libraries (e.g., Fairseq [35] and Microsoft DeepSpeed [51]), and stochastic algorithms. These jobs also target common tasks such as image recognition, natural language processing, and gaming.

Researchers have also observed similar quality issues and root causes. Prior work [38], [39], [55] found that "GPUs are among the top-3 most failed hardware and GPU memory is more sensitive to uncorrectable errors than main memory," [41] which is consistent with our finding. The chronicles [56] of the OPT (Open Pre-trained Transformers) development described how Meta (formerly Facebook) researchers fought against various quality issues. Many came from hardware faults (e.g., InfiniBand issues and GPU ECC errors), resulting in "$\sim$2 machines going down every day." [56] There was once a severe platform-side fault: "the cloud provider's support team accidentally deleted our entire cluster on December 21, 2021." [56] For user-side faults, existing DL-related empirical studies [23], [27], [31] have mentioned some code bugs that lead to job crashes and slowdowns.

B. Future Research Directions

Based on our study, we propose the following future research directions:

Tool Support.

Hardware Fault Prediction. Nearly one-third (102; 28.33%) of the quality issues are caused by the faults of GPUs, networking, and compute nodes. We could train a machine learning model to predict which hardware device would malfunction and when it happens. For example, Liu et al. [41] used machine learning techniques (e.g., LSTM [57], 1D-CNN [58], and ensemble learning [59]) to predict GPU errors based on various GPU and machine parameters, such as temperature, machine uptime, and GPU utilization/type/position.

Hang Analyzer. Although job hangs only account for 7.50% (27) of the quality issues, it is rather challenging for SREs to diagnose and resolve them. Unlike prior software hangs (i.e., unresponsiveness or freeze) [48], [49], job hangs deeply involve the underlying NVIDIA runtimes (i.e., NCCL) and hardware (e.g., the GPU and InfiniBand). For instance, there exists a potential deadlock in NCCL when the number of communication nodes increases. We could develop static and dynamic hang analyzers to reduce this type of issue proactively before job submission.

Crash Tolerance. 198 (55.00%) quality issues are job crashes; thus, the latest training progress (i.e., model weights and biases) is permanently lost. A common yet simple method against a crash is to save model checkpoints regularly. However, the built-in model checkpointing of DL frameworks is preliminary and inefficient. We could employ frequency adaptivity [60], [61], incremental checkpointing [62], asynchrony [61], [63], and high-performance serialization [64], [65] to implement more effective model checkpointing. Moreover, elastic training [66]–[68] is also a useful technique that enables dynamic job scaling to prevent various crashes.

Code Advisor. We have seen that many quality issues result from user code bugs, improper permission, and policy violations in our study. Therefore, we could develop static-analysis-based code advisors which detect various issues proactively in users' programs, Shell scripts, Dockerfiles, and configuration/credential files. Also, these code advisors may provide advanced fix suggestions or automated program repair features.

Platform Improvement.

Fault-aware Job Scheduling. After examining all the issue records and discussing with some SREs, we observe that some particular compute nodes have a higher probability of outages under heavy load. Currently, deep learning platforms allocate nodes to a job mainly based on the required resources. Such platforms could learn from the historical failure data of compute nodes and incorporate fault awareness into their job schedulers. A possible future work is to enable platforms to estimate job workloads and schedule long-running jobs to more stable nodes instead of the fault-prone ones. In this way, potential job issues could be reduced.

Platform Testing. More than half of the quality issues are due to hardware and platform-side faults. Since the computing devices (e.g., GPUs and TPUs), high-speed networks (e.g., InfiniBand), and software (e.g., Kubernetes and NVIDIA runtimes) for deep learning are evolving fast, platforms need more advanced and comprehensive testing methods. For example, Microsoft SuperBench [46] uses representative training workloads to validate AI infrastructure and has detected many new hardware/platform issues. We could also employ stress testing [69], fault injection [45], [70], and model checking [71] to test DL platforms more thoroughly and effectively.

IX. Related Work

Over the years, many researchers have conducted empirical studies [19]–[22] to understand the characteristics of the big data analytics platform and data-parallel programming paradigm. For example, Zhou et al. [19] studied 210 randomly selected quality issues from an internal production big data platform of Microsoft. Kavulya et al. [20] analyzed 10-month MapReduce logs from Yahoo's M45 supercomputing cluster and characterized "resource utilization patterns, job patterns, and sources of failures." Xiao et al. [21] studied the production data-parallel programs of Microsoft and presented "interesting findings on commutativity, non-determinism, and correctness." Li et al. [22] provided a comprehensive study on two hundred failed big data jobs, which investigated major failure types, root causes, fixes, and debugging practices.

There is also a line of research [38], [39], [55], [72] studying the various faults arising in high-performance computing (HPC) clusters. For example, Tiwari et al. [38] analyzed general GPU errors on the Titan supercomputer and presented the implications for future HPC design and operation. Nie et al. [39] extended the above work to study GPU soft-errors "that can be corrected by the ECC mechanism but do not result in execution loss." Di Martino et al. [72] presented an analysis of system failures from Blue Waters, another supercomputer. Gupta et al. [55] studied the spatial characteristics of system failures. These researches discovered that GPU errors were among the top hardware faults, which is consistent with our finding.

Recently, we have seen some empirical studies on deep learning [23]–[31]. However, most focus on the failures and bugs of DL frameworks, compilers, programs, and jobs. For example, Zhang et al. [27] studied 4,960 internal DL job failures collected from Microsoft within three weeks and classified them into twenty categories. In this paper, we have seen that the quality issues caused by user-side faults are more than two-fifths of the total; therefore, the work by Zhang et al. could help users improve their deep learning programs significantly. Cao et al. [31] researched 238 performance bugs in TensorFlow and Keras programs, which were collected from 225 Stack Overflow posts. Shen et al. [29] conducted a systematic study of 603 bugs in three popular DL compilers and developed a practical fuzzer to expose more bugs in Apache TVM [73]. Jeon et al. [26] analyzed low GPU utilization of large-scale, multi-tenant GPU clusters for DL training. Taking the perspective of platforms, the authors pointed out that the root causes came from gang scheduling [36], resource locality, and job failures. They also suggested how to improve the future generation

of cluster schedulers. This research considered more about the GPU utilization of clusters. Instead, our work focuses on quality issues and provides valuable guidelines for the future development of deep learning platforms.

X. Conclusion

This paper has presented a comprehensive empirical study on quality issues of Platform-X, an internal production deep learning platform of Microsoft. We randomly selected 360 real issues and manually analyzed their common symptoms, root causes, and mitigation actions. Based on our findings, we suggested possible research topics to reduce quality issues, enhance platform reliability, and improve operational activities. We believe that this work provides valuable guidelines for the future development of deep learning platforms.

References

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, A. D. Michaël Mathieu, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, 2019.

[4] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021.

[5] E. Rolf, J. Proctor, T. Carleton, I. Bolliger, V. Shankar, M. Ishihara, B. Recht, and S. Hsiang, "A generalizable and accessible approach to machine learning with global satellite imagery," *Nature Communications*, vol. 12, 2021.

[6] "Microsoft azure machine learning," https://azure.microsoft.com/en-us/services/machine-learning-service, 2022.

[7] "Amazon sagemaker," https://aws.amazon.com/sagemaker, 2022.

[8] "Google cloud ai," https://cloud.google.com/products/ai, 2022.

[9] Wikipedia, "InfiniBand — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=InfiniBand&oldid=1104722169, 2022.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, vol. 32. Curran Associates, Inc., 2019, pp. 8024–8035.

[11] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283.

[12] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *CoRR*, vol. abs/1910.03771, 2019.

[13] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Commun. ACM*, vol. 59, no. 5, p. 50–57, apr 2016.

[14] C. B. Seaman, F. Shull, M. Regardie, D. Elbert, R. L. Feldmann, Y. Guo, and S. Godfrey, "Defect categorization: Making use of a decade of widely varying historical data," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 149–157.

[15] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, and C. Zhai, "Have things changed now? an empirical study of bug characteristics in modern open source software," in *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability*, ser. ASID '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 25–33.

[16] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An empirical study of operating systems errors," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 73–88.

[17] N. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 797–814, 2000.

[18] C. Andersson and P. Runeson, "A replicated quantitative analysis of fault distributions in complex software systems," *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 273–286, 2007.

[19] H. Zhou, J.-G. Lou, H. Zhang, H. Lin, H. Lin, and T. Qin, "An empirical study on quality issues of production big data platform," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15. IEEE Press, 2015, p. 17–26.

[20] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 94–103.

[21] T. Xiao, J. Zhang, H. Zhou, Z. Guo, S. McDirmid, W. Lin, W. Chen, and L. Zhou, "Nondeterminism in mapreduce considered harmful? an empirical study on non-commutative aggregators in mapreduce programs," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 44–53.

[22] S. Li, H. Zhou, H. Lin, T. Xiao, H. Lin, W. Lin, and T. Xie, "A characteristic study on failures of production distributed data-parallel programs," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, p. 963–972.

[23] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang, "An empirical study on tensorflow program bugs," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 129–140.

[24] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 510–520.

[25] T. Zhang, C. Gao, L. Ma, M. Lyu, and M. Kim, "An empirical study of common challenges in developing deep learning applications," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, 2019, pp. 104–115.

[26] M. Jeon, S. Venkataraman, A. Phanishayee, u. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant gpu clusters for dnn training workloads," in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '19. USA: USENIX Association, 2019, pp. 947–960.

[27] R. Zhang, W. Xiao, H. Zhang, Y. Liu, H. Lin, and M. Yang, "An empirical study on program failures of deep learning jobs," in *Proceedings of the 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1159–1170.

[28] L. Jia, H. Zhong, X. Wang, L. Huang, and X. Lu, "An empirical study on bugs inside tensorflow," in *Database Systems for Advanced Applications: 25th International Conference, DASFAA 2020, Jeju, South Korea, September 24–27, 2020, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 604–620.

[29] Q. Shen, H. Ma, J. Chen, Y. Tian, S.-C. Cheung, and X. Chen, "A comprehensive study of deep learning compiler bugs," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*,

ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 968–980.

[30] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021.

[31] J. Cao, B. Chen, C. Sun, L. Hu, and X. Peng, "Characterizing performance bugs in deep learning systems," *CoRR*, vol. abs/2112.01771, 2021.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[33] ONNX, "Open neural network exchange," https://onnx.ai/, 2017.

[34] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, mar 2014.

[35] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A fast, extensible toolkit for sequence modeling," in *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[36] J. Ousterhout, "Scheduling techniques for concurrent systems," in *Proceedings of the 3rd International Conference on Distributed Computing Systems*, 1982, pp. 22–30.

[37] NVIDIA, "Nvidia collective communications library (nccl)," https://developer.nvidia.com/nccl, 2022.

[38] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland, "Understanding gpu errors on large-scale hpc systems and the implications for system design and operation," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 331–342.

[39] B. Nie, D. Tiwari, S. Gupta, E. Smirni, and J. H. Rogers, "A large-scale study of soft-errors on gpus in the field," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 519–530.

[40] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine learning models for gpu error prediction in a large scale hpc system," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 95–106.

[41] H. Liu, Z. Li, C. Tan, R. Yang, G. Cao, Z. Liu, and C. Guo, "Prediction of GPU failures under deep learning workloads," *CoRR*, vol. abs/2201.11853, 2022.

[42] NVIDIA, "Nvidia data center gpu manager (dcgm)," https://developer.nvidia.com/dcgm, 2022.

[43] ——, "Nvlink & nvswitch: Fastest hpc data center platform," https://www.nvidia.com/en-us/data-center/nvlink, 2022.

[44] M. Azure, "Azure spot virtual machines," https://azure.microsoft.com/en-us/products/virtual-machines/spot/, 2022.

[45] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "Nvbitfi: Dynamic fault injection for gpus," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 284–291.

[46] Microsoft, "Superbench: a validation and profiling tool for ai infrastructure," https://github.com/microsoft/superbenchmark, 2021.

[47] "Tensorboard: Tensorflow's visualization toolkit," https://www.tensorflow.org/tensorboard, 2022.

[48] X. Wang, Z. Guo, X. Liu, Z. Xu, H. Lin, X. Wang, and Z. Zhang, "Hang analysis: Fighting responsiveness bugs," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, ser. Eurosys '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 177–190.

[49] J. He, T. Dai, X. Gu, and G. Jin, "Hangfix: Automatically fixing software hang bugs for production cloud systems," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 344–357.

[50] Jupyter, "Project jupyter," https://jupyter.org, 2022.

[51] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 3505–3506.

[52] Microsoft, "What is an azure machine learning workspace?" https://learn.microsoft.com/en-us/azure/machine-learning/concept-workspace, 2022.

[53] S. Boag, P. Dube, B. Herta, W. Hummer, V. Ishakian, K. JAYARAM, M. Kalantar, V. Muthusamy, P. NAG-PURKAR, and F. Rosenberg, "Scalable multi-framework multi-tenant lifecycle management of deep learning training jobs," in *Workshop on ML Systems, NIPS*, 2017.

[54] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective cluster scheduling for deep learning," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI '18. USA: USENIX Association, 2018, p. 595–610.

[55] S. Gupta, D. Tiwari, C. Jantzi, J. Rogers, and D. Maxwell, "Understanding and exploiting spatial properties of system failures on extreme-scale hpc systems," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 37–44.

[56] Meta, "Metaseq: A codebase for working with open pre-trained transformers." https://github.com/facebookresearch/metaseq/blob/main/projects/OPT/chronicles/README.md, 2022.

[57] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[58] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical systems and signal processing*, vol. 151, p. 107398, 2021.

[59] C. Zhang and Y. Ma, *Ensemble Machine Learning: Methods and Applications*. Springer Publishing Company, Incorporated, 2012.

[60] Z. Lan and Y. Li, "Adaptive fault management of parallel applications for high-performance computing," *IEEE Trans. Comput.*, vol. 57, no. 12, p. 1647–1660, dec 2008.

[61] J. Mohan, A. Phanishayee, and V. Chidambaram, "CheckFreq: Frequent, Fine-Grained DNN checkpointing," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, Feb. 2021, pp. 203–216.

[62] A. Eisenman, K. K. Matam, S. Ingram, D. Mudigere, R. Krishnamoorthi, M. Annavaram, K. Nair, and M. Smelyanskiy, "Check-n-run: A checkpointing system for training recommendation models," *CoRR*, vol. abs/2010.08679, 2020.

[63] B. Nicolae, J. Li, J. M. Wozniak, G. Bosilca, M. Dorier, and F. Cappello, "Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 172–181.

[64] K. Varda *et al.*, "Cap'n proto serialization/rpc system - core tools and c++ library," https://github.com/capnproto/capnproto, 2013.

[65] D. Raghavan, P. Levis, M. Zaharia, and I. Zhang, "Breakfast of champions: Towards zero-copy serialization with nic scatter-gather," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 199–205.

[66] PyTorch, "Torch distributed elastic," https://pytorch.org/docs/1.12/distributed.elastic.html, 2022.

[67] Y. Wu, K. Ma, X. Yan, Z. Liu, and J. Cheng, "Elastic deep learning in multi-tenant GPU cluster," *CoRR*, vol. abs/1909.11985, 2019.

[68] D. Shukla, M. Sivathanu, S. Viswanatha, B. Gulavani, R. Nehme, A. Agrawal, C. Chen, N. Kwatra, R. Ramjee, P. Sharma, A. Katiyar, V. Modi, V. Sharma, A. Singh, S. Singhal, K. Welankar, L. Xun, R. Anupindi, K. Elangovan, H. Rahman, Z. Lin, R. Seetharaman, C. Xu, E. Ailijiang, S. Krishnappa, and M. Russinovich, "Singularity: Planet-scale, preemptive and elastic scheduling of ai workloads," 2022.

[69] I. Vanninen, "Limits of the cloud: Stress testing in azure – part 1," https://zure.com/blog/limits-of-the-cloud-stress-testing-in-azure-part-1/, 2021.

[70] M. Linkhorst, "chaoskube: periodically killing random pods in your kubernetes cluster," https://github.com/linki/chaoskube, 2022.

[71] J. Yang, T. Chen, M. Wu, Z. Xu, X. Liu, H. Lin, M. Yang, F. Long, L. Zhang, and L. Zhou, "Modist: Transparent model checking of unmodified distributed systems," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI '09. USA: USENIX Association, 2009, p. 213–228.

[72] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 610–621.

[73] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An automated End-to-End optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 578–594.