A Survey on Failure Analysis and Fault Injection in AI Systems

GUANGBA YU, GOU TAN, and HAOJIA HUANG, Sun Yat-sen University, China
ZHENYU ZHANG and PENGFEI CHEN, Sun Yat-sen University, China
ROBERTO NATELLA, Federico II University of Naples, Italy
ZIBIN ZHENG, Sun Yat-sen University, China

The rapid advancement of Artificial Intelligence (AI) has led to its integration into various areas, especially with Large Language Models (LLMs) significantly enhancing capabilities in Artificial Intelligence Generated Content (AIGC). However, the complexity of AI systems has also exposed their vulnerabilities, necessitating robust methods for failure analysis (FA) and fault injection (FI) to ensure resilience and reliability. Despite the importance of these techniques, there lacks a comprehensive review of FA and FI methodologies in AI systems. This study fills this gap by presenting a detailed survey of existing FA and FI approaches across six layers of AI systems. We systematically analyze 160 papers and repositories to answer three research questions including (1) what are the prevalent failures in AI systems, (2) what types of faults can current FI tools simulate, (3) what gaps exist between the simulated faults and real-world failures. Our findings reveal a taxonomy of AI system failures, assess the capabilities of existing FI tools, and highlight discrepancies between real-world and simulated failures. Moreover, this survey contributes to the field by providing a framework for fault diagnosis, evaluating the state-of-the-art in FI, and identifying areas for improvement in FI techniques to enhance the resilience of AI systems.

CCS Concepts: • **General and reference** → **Surveys and overviews**; **Reliability**; **Performance**; • **Software and its engineering** → **Software organization and properties**; • **Computing methodologies** → **Artificial intelligence**.

Additional Key Words and Phrases: Failure Analysis, Fault Injection, Chaos Engineering, AI system, MLOps

1 INTRODUCTION

Artificial Intelligence (AI) has made significant strides over the past decade, permeating both academic and industrial areas. Large Language Models (LLMs), in particular, have proven to be a game changer, propelling AI to unprecedented heights and facilitating a myriad of applications in fields such as software engineering [141, 192, 196] and human language translation [8, 59, 85]. This evolution has led to the integration of AI models into a growing array of products, transforming them into sophisticated AI systems. Notable examples of such integration include Gemini [51], Bing [117], and ChatGPT [137], which underscore the pivotal role of AI in enhancing and expanding the capabilities of modern technology solutions.

The escalating complexity and ubiquity of AI systems necessitate addressing their inherent vulnerabilities and failure-related challenges. A Meta AI report [208] points out over 100 failures during the training of OPT-175B. Similarly, ChatGPT encountered 173 outages in 2023, causing a maximum user impact over 427 minutes [138]. Such failures can degrade user experience, and even incur financial losses. Hence, mitigating AI system failures is of paramount importance.

Failure analysis (FA) and fault injection (FI) techniques are instrumental in identifying limitations and bolstering the reliability of AI systems. Researchers and practitioners alike have embarked on extensive investigations into AI system failures. Studies [22, 67, 69, 94, 100, 111, 178, 209] have analyzed AI system failures from platforms like Stack Overflow or Github, while others [45, 53, 73, 139, 176, 182, 207] have focused on failures in large-scale production AI systems. Such failure analyses enable the identification of patterns, root causes, and locations, thereby informing

Authors' addresses: Guangba Yu, yugb5@mail3.sysu.edu.cn; Gou Tan, tang29@mail2.sysu.edu.cn; Haojia Huang, huanghj78@mail2.sysu.edu.cn, Sun Yat-sen University, China; Zhenyu Zhang, zhangzhy239@mail2.sysu.edu.cn; Pengfei Chen, chenpf7@mail.sysu.edu.cn, Sun Yat-sen University, China; Roberto Natella, roberto.natella@unina.it, Federico II University of Naples, Italy; Zibin Zheng, zhzibin@mail.sysu.edu.cn, Sun Yat-sen University, China.

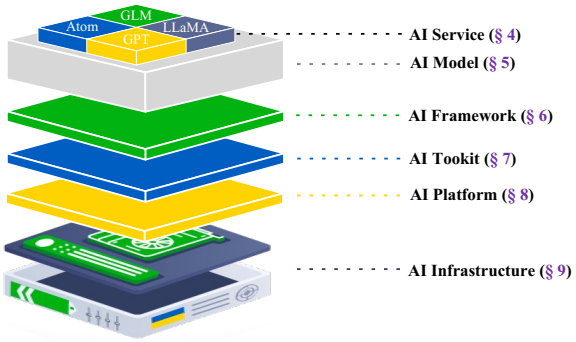

Fig. 1. An overall framework of AI system.

FI techniques. FI, a proactive approach, uncovers system weaknesses on resiliency before they become catastrophic failures. By deliberately injecting faults or abnormal conditions into systems, teams can evaluate and enhance their resilience to unexpected disruptions. Some existing FI approaches [16, 17, 58, 68, 74, 106, 164, 170] mimic faults in AI systems engineered by humans, while others [34, 91, 136, 162, 206, 210] simulate hardware errors.

Despite the progress, a comprehensive survey on FA and FI in AI systems is conspicuously absent. Furthermore, a gap exists between FI and FA, resulting in insufficient consideration of FA outcomes when crafting FI tools. Therefore, this study presents a comprehensive survey aimed at exploring and evaluating existing research for FA and FI in AI systems. We have meticulously reviewed and analyzed 160 corresponding papers and code repositories. As shown in Fig. 1, an AI system typically comprises six layers - AI Service, AI Model, AI Framework, AI Toolkit, AI Platform, and AI Infrastructure [187]. We attempt to address three research questions at each layer as follows.

- **RQ1:** What are the prevalent failures in current AI systems?
- **RQ2:** What types of faults can current FI tools simulate?
- **RQ3:** What gaps do exist between the simulated faults and the real-world failures?

RQ1 aims to catalog and analyze the failures that have occurred in current AI systems. Understanding these failures is crucial for several reasons. Since it helps in identifying common vulnerabilities within AI systems, informs developers about potential areas of improvement, and contributes to the development of more reliable AI applications. **RQ2** explores the capabilities of existing FI tools designed for AI systems. The ability to simulate a wide range of faults is essential for evaluating and enhancing the robustness and fault tolerance of AI systems. **RQ3** investigates the gap between simulated faults and real-world AI system failures, aiming to understand the limitations of current FI tools in producing the full spectrum of potential failures. Moreover, understanding these gaps helps in improving FI tools, and ultimately contributes to develop more resilient AI systems.

By examining the current landscape and identifying critical research gaps, this survey provides valuable insights for researchers and practitioners working towards building reliable and resilient AI systems. This study makes the following contributions:

- We present a comprehensive analysis and taxonomy of failures occurring at different layers of AI systems. By systematically characterizing these failures, we provide a valuable framework that can serve as a reference for failure diagnosis in AI systems.
- We conduct an in-depth examination of the capabilities of existing FI tools across various layers of AI systems. We offer insights into the state-of-the-art in simulating and reproducing potential failures. This work provides a foundation for assessing the reliability of AI systems.

- We explore the discrepancies between FI tools and real-world AI system failures. We identify the limitations of current FI approaches in simulating potential failure scenarios. By shedding light on these gaps, we emphasize the need for more comprehensive FI techniques in AI systems.

The rest of this paper is organized as follows. Section 2 provides background information on FA and FI in AI systems, followed by Section 3, which outlines our systematic literature review methodology. The subsequent sections analyze FA and FI in different layers of AI systems, including the AI service layer (Section 4), AI model layer (Section 5), AI framework layer (Section 6), AI toolkit layer (Section 7), AI platform layer (Section 8), and AI infrastructure layer (Section 9). Section 10 highlights research opportunities about FI in AI systems. The article concludes in Section 11.

2 BACKGROUND AND DEFINITIONS

2.1 Failures and Faults

We adopt the definitions of failures and faults proposed by previous work [7, 163]. Furthermore, we provide additional extensions and interpretations specific to AI systems.

- **Failure** is defined as "an incident that occurs when the delivered service deviates from the correct service" [7]. In the context of AI systems, failures can manifest in various ways. For example, a failure can occur when AI services become unreachable, and when the behavior of AI services does not meet the expected outcome (e.g., generating semantically incorrect text). These failures indicate a deviation from the desired or expected behavior of the AI system.
- **Fault** is the root cause of a failure. In AI systems, faults can be attributed to various sources, including algorithmic flaws, model design issues, or problems with the quality of the data used for training or inference. It is important to note that faults in AI systems may remain uncovered for some time, due to fault-tolerant approaches implemented in the system.

2.2 Failure Comparisons between AI and Cloud System

Failure analysis (FA) and fault injection (FI) are longstanding topics within the field of computer science, traditionally focusing on the robustness and reliability of systems. Historically, much of the literature has focused on cloud systems, reflecting their critical role in modern computing infrastructure [47, 52, 79, 93, 98, 172]. These systems adhere to a logic-based programming paradigm, where developers encode decision logic directly into the source code, facilitating a structured approach to FA and FI. In contrast, AI systems represent a paradigm shift towards a data-driven programming model. Here, developers design neural networks that derive decision-making logic from extensive datasets [22, 67, 69, 94, 100, 111, 178, 209]. This shift introduces both similarities and differences in the approach to FA between AI systems and traditional cloud systems.

As illustrated in Figure 2, while both AI and cloud systems are susceptible to common failures such as power disruptions and network outages, certain faults are unique or more critically impactful to AI systems. For instance, GPU failures, which might be relatively inconsequential in traditional cloud environments, can severely affect the performance and availability of AI systems. This distinction underscores the importance of conducting a comprehensive survey on FA and FI specifically for AI systems, especially in the current era dominated by "Large Models". This need forms one of the primary motivations behind the research presented in this article.

2.3 Failure Analysis and Fault Injection

Figure 3 demonstrates the relationship between faults, FA, and FI. In large-scale AI systems, faults commonly occur in AI systems due to their inherent complexity and the numerous interconnected components involved, leading to fault origination at various stages of system operation [22, 67, 69, 100, 111, 178, 209]. Once a fault is activated and a failure is detected (stage ①), engineers responsible

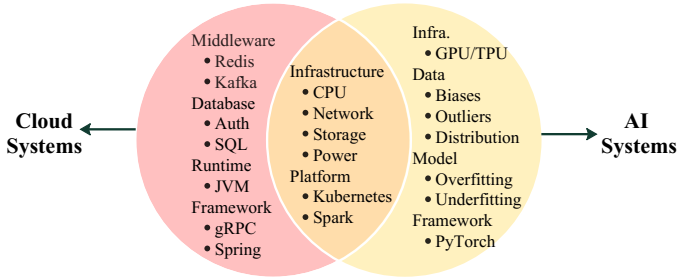

Fig. 2. Failure comparisons between AI and cloud systems.

for AI system maintenance engage in mitigating the failure based on observed behaviors (stage ②). During the failure mitigation process, engineers generate comprehensive incident reports encompassing failure details such as occurrence time, impact, failure manifestation, and mitigation strategies. The objective of FA is to utilize these incident reports as inputs to summarize the fault pattern (e.g., recurring type and location) (stage ③).

FI is a widely adopted technique for assessing and improving the reliability and security of systems, including AI systems. It involves the deliberate introduction of faults into a system to observe its behavior and validate its fault tolerance mechanisms. FI can be applied in various forms, such as software fault injection (e.g., mutation test [76] and API interception [189, 202]) or hardware fault injection (e.g., simulating hardware failures [130] and environmental disturbances [83]).

Leveraging the knowledge acquired from historical FA, engineers employ FI techniques to validate the reliability of AI systems. Following the injection of faults, engineers closely monitor AI system performance and behavior, ensuring the accurate identification and appropriate handling of the injected faults (stage ④). Based on the analysis of FI experiments, engineers can identify system vulnerabilities, weaknesses, and areas for improvement. By iteratively conducting FI experiments and refining the system based on the obtained results, the AI system can be continually improved, enhancing its reliability and effectiveness in real-world scenarios.

Consequently, FA and FI form closely intertwined processes that significantly contribute to the assessment and enhancement of AI system reliability. The insights derived from FA guide the selection and design of FI scenarios. The iterative feedback loop established by fault analysis and FI facilitates the continuous improvement of AI systems, thereby serving as the other motivation driving the research presented in this article.

3 SURVEY METHODOLOGY

To systematically collect the publications for conducting this survey, we specially constructed and maintained a repository [1] about FA and FI in AI systems. We first searched relevant papers in online digital libraries and extended the repository by manually inspecting all references of these papers.

To begin with, we searched several popular online digital libraries (e.g., IEEE Xplore, ACM Digital Library, Springer, Elsevier, and Wiley) with the following keywords: "failure and machine learning", "failure and deep learning", "failure and AI", "fault injection and machine learning", "fault injection and deep learning", "fault injection and AI", etc. We mainly focused on regular papers published relevant conferences (e.g., ICSE, FSE, ASE, SOSP, OSDI, ATC, NSDI, SC, DSN, ISSRE, AAAI, ICML, NeurIPS, etc) and journals (e.g., TSE, TOSEM, EMSE, TDSC, TPDS, TSC, etc). Upon identifying relevant papers, we conducted a recursive examination of the references, which allowed us to

[1] https://github.com/IntelligentDDS/awesome-papers/tree/main/Fault_tolerance#ai-system

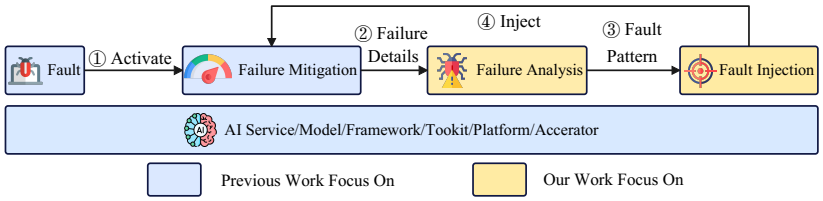

Fig. 3. Overall life cycle of an AI system failure.

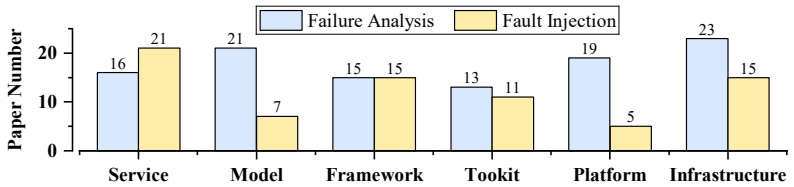

Fig. 4. Distribution of failure analysis and fault injection paper and repositories across layers.

manually inspect each reference and code repository of these papers. This process enabled us to collect additional publications and repositories related to our survey topics.

In total, our recursive search methodology enabled us to collect 160 publications and code repositories about FA and FI in AI systems, spanning from the year 2001 to 2024. These publications and repositories are classified into six layers by research topics including AI service, AI model, AI framework, AI toolkit, AI platform, and AI infrastructure. Moreover, the distribution of different classes is presented in Fig.4. It is important to note that some papers may address more than one research topic. Consequently, the total number of papers and repositories in Fig.4 is larger than 160. Next, we will show the details on FA and FI in different layers of AI systems.

4 FAILURE ANALYSIS AND FAULT INJECTION IN AI SERVICE

In the context of AI systems, the AI service layer can be considered as the topmost layer that directly interacts with users or other systems. This layer acts as an interface or entry point for users to access and utilize the AI capabilities provided by the underlying layers of the AI system. Potential failures at this layer can include unavailability of the service or incorrect outputs, such as incorrect inference from a classifier or hallucinations from an LLM-based service. This section delves into the FA and FI in AI services

4.1 Failure Analysis in AI Service

In our detailed exploration of potential failures in AI services, we have identified a broad spectrum of faults. These can be classified into several major categories including data fault, code and software fault, network transmission fault, and external attack. We summarize the types of failures in Table 1 that can occur in AI services. The *paper* column in Table 1 shows some representative papers in this study, as do the following tables below.

Data Fault. These faults related to the format, type, and noise of data can lead to the failure of AI services [165, 179, 211]. For example, incorrect data encoding (e.g., requesting UTF-8 but receiving ASCII) or inappropriate data types (e.g., expecting a string but receiving an integer) can prevent AI services from working properly. The JENGA framework explores the impact of data faults on predictions of machine learning models [165]. Furthermore, data drift and concept drift are common problems [179]. Data drift refers to a model trained on a specific distribution of data but then encountering a different distribution in practice. While concept drift occurs as the relationship between features and labels becomes invalid over time. Zhao *et al.* [211] investigate the impact of concept drift on model performance.

Development Fault. The primary reason for service failures is code quality such as bugs, logical faults in code, and poor software design [46, 63, 90]. Code faults typically originate from coding mistakes. Logical faults in code often involve incorrect algorithm implementation, impacting the accuracy and efficiency of inference. Thus, poor software design affects system performance. Additionally, failures in AI service API calls [189], originating from API incompatibility, API change, and API misuse, can lead to AI service failures.

Deployment Fault. During the deployment of AI services, various faults can arise, including outdated models [122], path configuration errors [60], and inappropriate resource allocation [112]. These faults can impact system performance and stability. As data changes over time, model performance may degrade, necessitating regular updates and retraining to maintain their effectiveness. Path configuration faults can prevent the proper loading of models and data. Inadequate resource allocation [112], especially inefficient use of CPU and GPU resources, can lead to decreased system performance and unnecessary waste.

Network Transmission Fault. It may arise from network congestion, bandwidth limitations, or failures in network hardware, leading to packet delays or losses [87]. Network congestion occurs when data traffic exceeds the network's bandwidth capacity, preventing data transmission on time. Bandwidth limitations are imposed by the maximum transmission rate of a network connection, often determined by service provider or the capabilities of network hardware. Furthermore, physical damage or configuration faults in network devices can also lead to packet delays or losses. failures in network hardware, leading to packet delays or losses [87]. Network congestion occurs when data traffic exceeds the network's bandwidth capacity, preventing data transmission on time. Bandwidth limitations are imposed by the maximum transmission rate of a network connection, often determined by service provider or the capabilities of network hardware. Furthermore, physical damage or configuration faults in network devices can also lead to packet delays or losses.

External Attack. In addition to internal faults within AI services, external attacks can also lead to service failures. These include network attacks such as Distributed Denial of Service (DDoS) attacks [113] and Man-in-the-Middle (MITM) attacks [149], which can not only cause temporary service interruptions but also lead to data leakage or corruption. Moreover, adversarial attacks target the AI model by designing malicious inputs (e.g., meticulously modified images or texts) to deceive the AI into making incorrect actions [2, 4, 145]. Adversarial attacks further divide into white-box attacks and black-box attacks. In white-box attacks, the deployed model is fully understood, including inputs and architecture, allowing for targeted attacks. In black-box attacks, only the model's inputs and output labels/confidences are accessible.

4.2 Fault Injection in AI Service

Fault injection in AI service layer encompasses four dimensions including data, service API, network transmission, and external attack. By simulating diverse fault scenarios in these dimensions, it is possible to assess the system's robustness and reliability, thereby preventing inaccurate predictions or even system failures. Table 2 illustrates the current FI tools in the AI service layer, followed by a description of each of them.

Data Fault. Data perturbation is the most intuitive method of FI at the data dimension. Introducing noise or modifying data manually can simulate uncertainties in real-world data. It can be achieved using tools such as NumPy [132], Scikit-learn [166], and so on. JENGA [165] is a framework that studies the impact of data faults (e.g., missing values, outliers, typing errors, and noisy) on the predictions of machine learning models. Additionally, data and concept drift can be simulated through carefully designed data disturbances. Moreover, tools such as scikit-multiflow [120] and MOA [12] enable the simulation of sudden, gradual, and incremental drifts in data streams.

Table 1. Failure Analysis in AI Service

Group	Failure	Description	Paper
Data	Data Quality	Issues related to the format, type, and noise of data.	[165]
	Data Drift	A model trained on a specific distribution of data but then encountering a different distribution in practice.	[179]
	Concept Drift	The relationship between features and labels becomes invalid over time.	[179, 211]
Development	Defective Code	Logical faults in code, and poor software design.	[46, 63, 90]
	Service API Fault	API Incompatibility, API Change, and API Misuse.	[15, 189]
Deployment Fault	Configuration Fault	Outdated models, path configuration faults, and inappropriate resource allocation.	[60, 112, 122]
Network Fault	Network Transmis. Fault	Network congestion, bandwidth limitations, or failures in network hardware.	[87]
External Attack	Network Attack	Lead to temporary service interruptions and data leakage or corruption.	[113, 149]
	Adversarial Attack	Deceive the AI into making incorrect actions through malicious inputs.	[2, 4, 145]

Table 2. Fault Injection in AI Service

Group	Fault	Description	Tools or Methods
Data	Data Perturbation	Introduce noise or modifying data artificially can simulate uncertainties in real-world data.	NumPy [132], Scikit-learn [166], JENGA [165]
	Data and Concept Drift	Simulate sudden, gradual, and incremental drifts in data streams.	Scikit-multiflow [120], MOA [12]
Service API	Service API Fault	Leverage Envoy to introduce API return errors into service communication.	Istio [71], MicroFI [19]
Network	Low-Quality Network	Simulate network delays, jitter, packet loss, and reordering.	Toxiproxy [184], ChaosBlade [18]
External Attack	Adversarial Attack	Use known information and relevant patterns to attack the model.	FGSM [50], PGD [108], DI-AA [191], Grey-box attack [88]
	Prompt Attack	Guide models to generate special outputs by adding prompts to the input text.	IPI [2], PromptAid [119], HouYi [101], Goal-guided attack [204]

Service API Fault. Istio, an open-source service mesh, addresses provides robust traffic management features. Istio's fault injection capabilities are primarily exposed through its Service API, which allows users to define fault injection rules declaratively [19, 71]. These rules are specified within Istio's VirtualService resources, which are then propagated to the Envoy proxies deployed as sidecars alongside each service instance. However, deploying and managing Istio can add complexity to the AI infrastructure. The learning curve for Istio is steep, and managing its components alongside AI services can be challenging.

Network Fault. Common network fault injections include network delays, network jitter, packet loss, and reordering. The injection of these network faults may result in the delayed or non-response of user requests for AI services. Toxiproxy [184] is a TCP proxy used to simulate network and system conditions for chaos and resilience testing. ChaosBlade [18] is an open source experimental injection tool that adheres to the principles of chaos engineering and chaos experimental models, supporting a rich array of experimental scenarios. Istio can also leverage the Envoy's advanced traffic management capabilities to simulate network conditions such as delay, packet loss, and service unavailability [72].

External Attack. Adversarial attacks are deliberate attacks on the AI service, including both white-box and black-box attacks. White-box attacks leverage model structure and parameter information to strategically generate adversarial samples, such as FGSM [50], PGD [108], and DI-AA [191]. Black-box attacks lack insight into the model and rely on observing model inputs

Table 3. Gap between Failure Analysis and Fault Injection in AI Service

Failure	JENGA	MOA	FGSM	PromptAid	Istio	ChaosBlade	Covered
Data Quality	✔						True
Data Drift		✔					True
Concept Drift		✔					True
Defective Code							False
Service API Fault					✔		True
Configuration Fault							False
Network Transmission Fault					✔	✔	True
Network Attack						✔	True
Adversarial Attack			✔	✔			True

and outputs to create adversarial samples, such as HopSkipJump [21], PopSkipJump [173], and GeoDA [148]. Grey-box attacks utilize partial information to generate adversarial samples. Raz *et al.* [88] attack image-to-text models based on adversarial perturbations. In addition to traditional methods, prompt fault injection has become popular for large models. It guides models to generate special outputs by adding prompts to the input text. Greshake *et al.* [2] use indirect prompt injection to exploit LLM-integrated applications and systematically investigate impacts and vulnerabilities, including data theft, worming, information ecosystem contamination, and other novel security risks. PromptAid [119] can introduce keyword and paraphrasing perturbations into prompts. HouYi [101] conducts prompt fault injections in a black-box manner. Zhang *et al.* [204] propose goal-guided generative prompt injection attack on LLMs.

4.3 Gap between Failure Analysis and Fault Injection in AI Service

Based on the comparative analysis presented in Table 3, which outlines the capabilities of various FI tools in addressing diverse failure modalities within AI services, several critical insights regarding the discrepancies between FA and FI can be elucidated.

Incomplete Coverage. The scrutinized FI tools, in aggregate, encompass a substantial proportion of the delineated fault types. Nonetheless, certain fault types (e.g., "Defective Code") remain unaddressed by any of the existing tools. This revelation underscores the exigency for subsequent research to ameliorate these deficiencies in fault simulation.

Diversity of Tools. Each FI tool exhibits a predilection for addressing specific fault types. For instance, JENGA is attuned to data quality faults, whereas MOA is adept at handling data and concept drift faults. This diversity implies that practitioners may be compelled to deploy a suite of tools to achieve a comprehensive simulation and analysis of failures within AI services, contingent upon the specific fault types of interest.

New Emerging Fault Types. The incorporation of fault types such as "Prompt Injection Attacks" accentuates the burgeoning significance of accounting for external factors and security facets in the FA of AI services. Nevertheless, traditional FI tools (e.g., ChaosBlade) are designed for cloud computing services and do not consider the specific scenarios of AI systems. As AI services become increasingly interconnected and susceptible to a myriad of threats, it is imperative to cultivate FI tools capable of efficaciously simulating failures emanating from these nascent domains.

5 FAILURE ANALYSIS AND FAULT INJECTION IN AI MODEL

AI model layer is a crucial component in AI systems, residing beneath the AI service layer. This layer is responsible for managing the various AI models and algorithms that power the AI capabilities of the system. Similar to the AI service layer, potential failures at this layer can include unavailability of the model or incorrect outputs. In this section, we delve into the FA and FI in AI models.

Table 4. Failure Analysis in AI Model

Group	Failure	Description	Paper
Data	Data Quality	Low-quality data leads to poor model performance.	[32, 95, 193]
	Data Preprocessing Fault	The inadequate handling of data noise, damage, loss, and inconsistency.	[15, 33, 109]
Model Hyperparameter	Inappropriate Layer and Neuron Quantity	Incorrectly setting the number of layers and neurons can affect the model's parameter count and performance.	[9, 15, 54, 116, 188]
	Inappropriate Learning Rate, Epochs, and Batch Size	Influence the training speed and model performance (overfitting and underfitting).	[57, 167]
Model Structure and Algorithm	Misuse Neural Networks	Inappropriate types of neural networks.	[171]
	Misuse Activation Function	Introduce non-linearity to enhance model fitting ability.	[40]
	Misuse Regularization	Inappropriate regularization lead to overfitting.	[181]
	Misuse Optimizer	Influence the training speed and model performance.	[55]
	Misuse Loss Function	Affect the speed and degree of convergence in training.	[190]
	Dataset Partitioning Fault	Insufficient data for training and validation.	[124]

5.1 Failure Analysis in AI Model

Recent research has explored multiple reasons for AI model failures. Researchers analyze various sources, including GitHub commits, Stack Overflow posts, and expert interviews. These analyses have provided crucial insights into enhancing the reliability and robustness of AI systems. For example, Humbatova *et al.* [67] categorize faults within deep learning systems by examining 1,059 GitHub commits and issues of AI systems, while Islam *et al.* [69] analyze error-prone stages, bug types, and root causes in deep learning pipelines from 2,716 Stack Overflow posts and 500 GitHub bug fix commits. Additionally, Nikanjam *et al.* [131] classify faults in deep reinforcement learning programs from 761 documents. We focus on the training and testing phases of AI models, where faults are categorized as data faults, model hyperparameter faults, and model structure and algorithm faults, as shown in Table 4.

Data Fault. The quality of training data is pivotal for successful model training. Alice *et al.* [32] conduct a systematic analysis of data quality attributes (accuracy, uniqueness, consistency, completeness, and timeliness) across five software bug datasets and find that 20-71% of data labels are inaccurate, which could severely hinder model training in extreme cases. Some studies [95, 193] have examined challenges faced by data quality. Furthermore, data preprocessing and augmentation are crucial. Raw data are susceptible to noise, damage, loss, and inconsistency, thus necessitating preprocessing steps (e.g., data cleaning, integration, transformation, and reduction) to facilitate easier knowledge extraction from datasets. Data augmentation aims to expand the training dataset through specific transformations to enhance the model's generalizability. Das [33] lists ten common faults in data preprocessing, while Maharana *et al.* [109] discuss various data preprocessing and data augmentation techniques to enhance model performance.

Model Hyperparameter Fault. The parameters of an AI model include both pre-training hyperparameters (e.g., the number of hidden layers, batch size, and learning rate) and post-training model parameters (e.g., network weights and biases). Basha *et al.* [9] examine the effects of different numbers of fully connected layers on convolutional neural networks (CNNs). Uzair *et al.* [188] investigate how the number of hidden layers affects the efficiency of neural networks. ShortGPT [116] points out that many layers in LLMs exhibit high redundancy, with some layers playing minimal roles. Gurnee *et al.* [54] utilize seven different models (ranging from 70 million to 6.9 billion parameters) to study the sparsity of activations in LLM neurons. Additionally, the learning rate (LR), epochs, and batch size (BS) influence the training speed and the performance of the trained model. He *et al.* [57] indicate that the ratio of batch size to learning rate should not be too large to ensure good generalization. Shafi *et al.* [167] explore the optimization of hyperparameters, including the learning rate, batch size, and epochs, as well as their interrelationships.

Model Structure and Algorithm Fault. Shiri *et al.* [171] investigate various aspects of different models and evaluate their performance on three public datasets. Activation functions are crucial for introducing non-linearity. Dubey *et al.* [40] compare the performance of 18 distinct activation functions (e.g., Sigmoid, Tanh, and ReLU) on various datasets. Model training requires regularization to avoid overfitting. Tian *et al.* [181] compare different regularization techniques, including sparse regularization, low-rank regularization, dropout, batch normalization, and others. They discusse the selection of regularization techniques for specific tasks. The selection of optimizers significantly affects model performance and training speed. Haji *et al.* [55] compare various optimizers like SGD, Adam, AdaGrad, etc. They highlight the advantages and disadvantages of these optimizers in terms of training speed, convergence rate, and performance. The loss function is also essential for minimizing the discrepancy between predicted results and target values. Wang *et al.* [190] introduce 31 loss functions from five aspects: classification, regression, unsupervised learning of traditional machine learning, object detection, and face recognition of deep learning. Additionally, the ratio of training to validation data must be considered to ensure the model has enough data for learning while the validation data is adequate for model adjustments [124].

5.2 Fault Injection in AI Model

Fault injection in AI models typically involves interfering with the training process to create models with inferior performance. This technique was called mutation testing. The concept of mutation testing that we will discuss next is analogous to fault injection. A common method for conducting mutation testing on AI models involves designing mutation operators that introduce faults into the training data or the model training program, and then analyzing the behavioral differences between the original and mutated models.

Recent studies in mutation testing have made significant contributions. We have summarized these works as shown in Table 5. DeepMutation [106] designs 13 mutation operators that inject faults into both the training data and code of deep learning models. DeepMutation++ [62] combines DeepMutation (eight model-level operators for FNN models) and proposes nine new operators specifically for RNN models, enabling both static mutations in FNNs and RNNs and dynamic mutations in RNNs. MuNN [170] develops five mutation operators, focusing on model-level fault injection. DeepCrime [68] implements 24 deep learning mutation operators to test the robustness of deep learning systems, including training data operators, hyperparameters operators, activation function operators, regularization operators, weights operators, loss function operators, optimization operators, and validation operators. Additionally, some studies have focused on mutation testing in reinforcement learning [104, 177] and unsupervised learning [103].

5.3 Gap between Failure Analysis and Fault Injection in AI Model

Based on the comparative analysis presented in Table 6, which delineates the capabilities of various FI tools in addressing diverse failure modalities within AI models, two critical insights regarding the discrepancies between FA and FI can be elucidated:

Differentiated Focus. The distinct FI tools appear to concentrate on disparate facets of AI model failures. For instance, DeepMutation and DeepCrime are adept at handling data quality and preprocessing faults, whereas MuNN is tailored towards layer and neuron quantity faults. This specialization implies that the selection of an FI tool should be contingent upon the specific fault types targeted for analysis.

Coverage Inconsistency. Table 6 reveals a disparity in coverage, with certain fault types, such as "Layer and Neuron Quantity Fault" and "Misuse Activation Function", being addressed by multiple tools, while others, like "Inappropriate LR, Epochs, and BS", are solely within the purview

Table 5. Fault Injection in AI Model

Group	Fault	DeepMutation [106]	DeepMutation++ [62]	MuNN [170]	DeepCrime [68]
Data	Duplicates training data	✓			
	Shuffle training data	✓			
	Change labels of training data	✓			✓
	Remove part of training data	✓			✓
	Unbalance training data				✓
	Add noise to training data	✓			✓
	Make output classes overlap				✓
Hyperparameters	Change batch size				✓
	Decrease learning rate				✓
	Change number of epochs				✓
	Disable data batching				✓
Activation Function	Change activation function			✓	✓
	Remove activation function	✓			✓
	Add activation function				✓
Regularisation	Add weights regularisation				✓
	Change weights regularisation				✓
	Remove weights regularisation				✓
	Change dropout rate				✓
	Change patience parameter				✓
Weights	Change weights initialisation				✓
	Add bias to a layer			✓	✓
	Remove bias from a layer				✓
	Change weights	✓	✓	✓	
	Shuffle weights	✓	✓		
Loss function	Change loss function				✓
Optimisation Function	Change optimisation function				✓
	Change gradient clipping				✓
Validation	Remove validation set				✓
Layers	Remove layer	✓	✓		
	Add layer	✓	✓		
	Duplicate one layer		✓		
Neuron	Delete Input Neuron			✓	
	Delete Hidden Neuron			✓	
	Block a neuron effect 0	✓	✓		
	Invert the activation status	✓	✓		
	Switch two neurons	✓	✓		
RNN Specific	Fuzz weights		✓		
	Reduce weight's precision		✓		
	Clear the state to 0		✓		
	Reset state to previous state		✓		
	Fuzz state value		✓		
	Reduce state value's precision		✓		
	Clear the gate value to 0		✓		
	Fuzz gate value		✓		
	Reduce gate value's precision		✓		
Number		13	17	5	24

of DeepCrime. This inconsistency may reflect the inherent challenges associated with simulating specific fault types or indicate a relative lack of focus within the FI.

6 FAILURE ANALYSIS AND FAULT INJECTION FOR AI FRAMEWORK

AI framework layer, including TensorFlow [1], PyTorch [140], and Keras [82], acts as a bridge between the AI model layer and the underlying hardware and system infrastructure. Akin to other software systems, these AI frameworks are susceptible to a variety of faults [94, 178]. Failures at the

Table 6. Gap between Failure Analysis and Fault Injection in AI Model

Failure	DeepMutation	DeepMutation++	MuNN	DeepCrime	Covered
Data Quality	✔			✔	True
Data Preprocessing Fault	✔			✔	True
Layer and Neuron Quantity Fault	✔	✔	✔		True
Inappropriate LR, Epochs, and BS				✔	True
Misuse Neural Networks	✔	✔	✔		True
Misuse Activation Function	✔		✔	✔	True
Misuse Regularization				✔	True
Misuse Optimizer				✔	True
Misuse Loss Function				✔	True
Dataset Partitioning Fault				✔	True

Table 7. Failure analysis in AI Framework

Group	Failure	Description	Paper
Data	Tensor Alignment Fault	Tensors do not align as expected, leading to shape mismatches.	[67, 209]
	Input Format Fault	The shape or type of input data mismatches the expected format.	[67, 69]
	Dataloader Crash Fault	Dataloader crashes due to memory leak in multipleworker.	[64]
API	API Usage Fault	API is used in a way that does not conform to the correct logic.	[67]
	API Compatibility Fault	Different APIs are not compatible with each other.	[67, 69, 209]
	API Version Fault	API version is incompatible with the code or dependencies.	[67, 69, 209]
Configuration	Framework Config. Fault	Incorrect configuration when using a framework.	[146, 198, 209]
	Device Config. Fault	Inability to leverage computing devices for optimal performance.	[67, 198]
	Environment Config. Fault	Environmental configuration faults when developing and deploying an AI framework.	[22, 94, 207]
Performance	Memory Management Fault	Faults occur when managing memory between heterogeneous devices.	[111, 146, 198]
	Parallelism Fault	Includes insufficient parallelism and excessive parallelism.	[111]
	Operator Inefficiency Fault	Trade-off of using different linear algebra operators.	[143]
Code	Syntax Fault	Faults occur when implementing and using AI framework.	[22, 178]
	Cross-Language Fault	Faults occur when utilizing multi-programming-language.	[94]
Algorithm	Implementation Logic Fault	Faults present in the algorithm's implementation.	[22, 146, 178]
	Algorithm Inefficiency Fault	Algorithms implemented using outdated or inefficient methods.	[75, 111]

AI framework layer can lead to unavailability, incorrect outputs, and poor performance perceived by users. Thus, ensuring the robustness of AI frameworks is crucial for the reliability of AI systems. This section is dedicated to analyzing faults at the AI framework layer.

6.1 Failure Analysis in AI Framework

Over recent years, a significant volume of research [1, 22, 67, 69, 94, 178, 198, 209] has been dedicated to the analysis of failures in AI frameworks. Studies on faults in AI failure can be primarily bifurcated into two categories including failures arising from the usage of AI frameworks, and failures stemming from the frameworks' implementation. We summarize the types of failures in Table 7 that can occur both in the utilization and the implementation of AI frameworks.

Data Fault may occur during the data input stage of an AI model. This type of fault is typically caused by unaligned tensors or incorrect formatting of input data. For example, it could occur when inputting a tensor array instead of an individual element from the array, or when mistakenly using a transposed tensor instead of the original tensor [67, 209]. Even more critically, the shape or type of the data inputted to the model may completely mismatch the expected input by the model which leads to the model unable to run correctly [67, 69]. One additional fault that occurs during the data

input stage is the dataloader crash fault [64]. This fault primarily occurs in training tasks of LLM in multiple workers. This issue arises from a gradual memory leak due to PyTorch's implementation of dataloader, which is caused by the copy-on-write mechanism used in the multiprocessing fork start method, combined with a suboptimal design of the Python list.

API Fault typically occurs during the call to APIs provided by AI frameworks. Such faults may due to the using of an API in a way that does not conform to the logic set out by developers of the framework [67]. Indeed, lack of inter-API compatibility and versioning issues could be one of the main culprits [67, 69, 209]. When different APIs are not compatible with each other or when the version of the API being used is not compatible with the requirements of the code or dependencies, it can result in API faults.

Configuration Fault typically occurs due to incorrect configuration of the framework. One example of this type of fault in TensorFlow is the confusion with computation model. Users may incorrectly constructed TensorFlow computation graphs using control-flow instead of data-flow semantics [198, 209]. Quan *et al.* [146] also analyse the failures in building and initialing JavaScript-based DL systems, such as npm package installation and multi-backend initialization. Another situation of this fault is the misconfiguration of the computing device (e.g., GPU). This type of misconfiguration can include selecting the wrong GPU device, mistakenly using CPU tensors instead of GPU tensors, or improper allocation of resources between CPU and GPU [67, 198].

Environment configuration faults mainly encompass the problems during the development and deployment processes of AI framework. Given that AI frameworks typically function in heterogeneous environments, ensuring compatibility with various devices and systems becomes crucial during the development process [22, 198]. This can result in failures during the build and compilation process, which hinders the development of AI frameworks. Apart from encountering environment configuration faults during the development process, deploying an AI framework also entails addressing environment faults, such as "path not found", "library not found" and "permission denied" [207]. Moreover, deploying the AI framework on various operating systems (e.g., Linux, Windows, Mac, and Docker environments) or utilizing different types of acceleration devices within the framework can also give rise to environment-related faults [94].

Performance Fault typically does not result in system downtime but can significantly impact the runtime of the system. In the aspect of AI framework, there is a wide range of causes for performance faults which can be quite diverse. One of the causes is memory inefficiencies. Existing AI frameworks such as PyTorch [140] and TensorFlow [1] are typically implemented using C/C++ and CUDA, and their memory management is often done manually [111, 198]. These frameworks need to handle memory exchanges between heterogeneous devices, which can potentially introduce memory inefficiency faults [146]. Apart from memory management faults, another cause for performance faults is threading inefficiency [111]. Such fault is commonly found in GPU related code. Insufficient parallelism can result in underutilization of device resources, while excessive parallelism can introduce additional overhead(e.g., context switches). Another cause is the trade-off of using different linear algebra libraries/operators. For example, when performing a small matrix operation on a GPU, the computation time may be longer compared to performing the same operation on a CPU [143].

Code Fault primarily refers to logic faults that occur during the implementation of AI framework. One example of code fault in AI framework is a syntax fault, which may occur both in the utilization and the implementation of AI framework. Expect traditional syntax fault occurring in command software system, AI frameworks also face faults related to tensor syntax faults during the implementation [22, 178]. Such faults may occur on account of tensor shape misalignment and operation on tensors across different devices. Apart from common syntax faults, another noteworthy code

Table 8. Fault Inject Tools to AI Framework

Tool	Framework	Description	Advantage	Instrumented	Link
TensorFI	TensorFlow	An interface-level fault injection approach focusing on the data-flow graph of TensorFlow.	Preserves portability and performance of the original system.	True	[36]
InjectTF	TensorFlow	Fault injection frameworks for both TensorFlow 1 and TensorFlow 2.	Compatible with different versions of TensorFlow.	True	[114]
TorchFI	PyTorch	A fault inject tool designed for PyTorch.	Simulate bit-flip errors that occur in registers or memory.	True	[11]
PyTorchFI	PyTorch	A tool that introduces perturbations in convolutional operations within neural networks.	Ensure compatibility with future versions and allows fault code to run at the native speed.	True	[144]
TensorFI2	TensorFlow	A tool utilizes the Keras API to intercept the state of tensors and injects fault to TensorFlow 2.	Avoid the overhead of graph duplication and inject faults into the model parameters.	True	[37]
SNIFF	Keras	A fault injection tool designed for reverse engineering of neural networks.	Specialize in neural network classifiers using the softmax activation function in the output layer.	True	[14]
MindFI	MindSpore	Perform fault injection on MindSpore.	Offer ease of use, stability, and efficiency.	False	[212]
enpheeph	Framework-agnostic	A fault injection tool independent of the underlying AI framework.	Adapt to different DNN frameworks with minimal modifications.	True	[5]

fault in AI frameworks is the problem in cross-programming-language communication. This kind of fault is particularly common in AI frameworks that utilize multi-programming-language [94].

Algorithm Fault is related to the defects in algorithm design [22, 178, 198]. This algorithm fault can be primarily categorized into two aspects including the incorrect implementation logic of an algorithm and the inefficient algorithm implementation. The former aspect mainly pertains to bugs present in the algorithm implementation within the AI framework [22, 146, 178]. The latter aspect arises from the challenge faced by AI framework developers in keeping up with the latest research and incorporating the most efficient methods for algorithm implementation [75, 111].

6.2 Fault Injection in AI Framework

In recent years, there has been a significant increase in research focusing on FI techniques specifically targeted at AI frameworks. As shown in Table 8, we elaborate on these works that are categorized according to different AI frameworks. These techniques often rely on a process known as "instrumentation". This is a method used in fault injection where the system, such as source code or logic gates, is modified to inject faults more accurately or efficiently.

Tensorflow. There are a series of works focus on designing an FI system for TensorFlow as it is one of the most popular frameworks in AI application. TensorFI [23] introduces an interface-level FI approach that focuses on the data-flow graph of TensorFlow. During the inference phase, TensorFI injects both hardware or software faults into TensorFlow operators, corrupting the output of the affected operators. As AI applications developed using TensorFlow 2 do not necessarily depend on data flow graphs, TensorFI2 [125] utilizes the Keras API to intercept the state of tensors and injects fault to TensorFlow 2. TensorFI2 employs the Keras Model API to modify the layer state or weight matrices that holds the learned model parameters, and utilizes the Keras backend API to intercept the layer computation or activation matrices that holds the output states of the layers. These make TensorFI2 avoid the overhead of graph duplication and inject faults into the model parameters. InjectTF [10] is another FI framework designed for TensorFlow. InjectTF implements

Table 9. Gap between Failure Analysis and Fault Injection in AI Framework

Failure	TensorFI	InjectTF	TorchFI	PyTorchFI	TensorFI2	SNIFF	MindFI	enpheeph	Covered
Data Fault	✔	✔	✔	✔	✔	✔	✔	✔	True
API Misuse									False
Configuration Fault									False
Performance Fault									False
Code Fault									False
Algorithm Fault	✔	✔	✔	✔	✔	✔	✔	✔	True

dedicated FI frameworks for both TensorFlow 1 and TensorFlow 2, namely InjectTF1 and InjectTF2. Similar to TensorFI, InjectTF involves the creation of a new data-flow graph.

PyTorch. TorchFI [48] is a fault inject tool designed for PyTorch, which simulates bit-flip faults that occur in registers or memory by performing single-bit flips on variables or activations within the framework. TorchFI focuses on convolution and fully connected layers and achieves fault injection by modifying the selected nodes within the neural network. Another related work, PyTorchFI [110] allows users to introduce neural network perturbations during the execution phase specifically in convolution operations targeting on weights and neurons in DNN. PyTorchFI does not make any modifications to the neural network topology or the source code of PyTorch itself. Instead, it utilizes PyTorch's hook functions to perturb the values of neurons during the forward propagation process of the computational model. By utilizing hooks to insert faults, PyTorchFI ensures compatibility with future versions of PyTorch and enables the fault code to run at the native speed of PyTorch, resulting in minimal overhead.

Other Frameworks. As a high-level API of TensorFlow, there are studies that explore fault injection based on Keras [82]. SNIFF [14] utilizes fault injection to achieve reverse engineering of neural networks. In the experimental process, it investigates and the fault injection to Keras. MindSpore [118] is a newly developed open source deep learning computing framework by Huawei. MindFI [212] is capable of performing fault injection on MindSpore at the data, software, and hardware levels, following three concepts including ease of use, stable and efficient.

Framework-agnostic. In contrast to prior approaches, enpheeph [27] does not rely on the underlying DNN frameworks. Consequently, it can seamlessly adapt to different DNN frameworks with minimal, or even zero, modifications to the internal code. enpheeph allows for fault injection at various levels, ranging from bit-level and tensor-level to layer-level. It also provides the flexibility to customize the precise number of bit-accurate injections during the execution process. In terms of network compression, enpheeph is the only framework that comprehensive supports for sparse tensors and quantization. This surpasses the functionality offered by most DNN frameworks, as they generally lack direct support for these features. Furthermore, enpheeph has the ability to scale and expand its capabilities across heterogeneous devices.

6.3 Gap between Failure Analysis and Fault Injection in AI Framework

Upon comparison of the fault types derived from FA, it becomes evident that there is room for enhancement in existing FI tools for AI frameworks. By identifying the gaps between FI tools and the outcomes of FA, engineers can gain a deeper understanding of the limitations of FI tools.

Fault Type Accommodation. The first gap is related to the fault types that the FI tool needs to accommodate and implement. The fault types identified in FA may not always have corresponding injection implementations. Table 9 presents the fault types supported by existing FI tools in AI framework, along with the fault types revealed during FA. Existing FI tools primarily inject faults by modifying specific tensors and constants or by using bit flipping techniques. Therefore, there are relatively fewer implementations of FI specifically targeting faults that are unrelated to

Table 10. Failure Analysis of AI Toolkit

Group	Failure	Description	Paper
Synchronization	Data Race	Inability to determine the order of "read&write" and "write&write" actions among multiple threads.	[70, 195]
	Barrier Divergence	Threads within the same block fail to reach a barrier due to variations in their execution flow.	[26, 195]
	Redundant Barrier Func.	Unnecessary synchronization operations.	[26, 195]
Memory Safety	Out-of-bounds Access	Access buffers beyond boundaries in global memory or shared memory.	[174, 213]
	Temporal Safety Fault	Access GPU memory that has already been freed or has not been properly allocated or initialized.	[174, 213]
	Failed Free Operation	Double free and invalid free operations.	[174, 213]
Dependency	Intra-dependency Fault	Incorrect versioning or unsuccessful installation of a toolkit.	[15, 65]
	Inter-dependency Fault	Mismatch of software and hardware.	[65]
Communication	NCCL Fault	Possibly due to a network error or a remote process exiting prematurely.	[45, 64, 78]
	NVLink Fault	Caused by the hardware failures like GPU overheating.	[45, 64]
	MPI Fault	A failure of network connection to peer MPI process or an internal failure of the MPI daemon itself.	[73]

model variables, such as "Performance Fault" and "Configuration Fault" which are more commonly encountered in chaos engineering practices targeting microservices. So extensive research and collection of prevalent faults in AI frameworks are required for the development of a FI tool so that the tool can better address the needs and requirements of users [125].

Framework Divergence. The second gap emerges from the divergence between the AI framework targeted by FA and the framework targeted by FI. This divergence often stems from version differences among AI frameworks. For instance, TensorFlow 1 and TensorFlow 2 demonstrate substantial differences in API usage and runtime logic. Failure analysis conducted on TensorFlow 1 may not be directly applicable to FI tools designed for TensorFlow 2. This implies that the TensorFI in Table 9 is only capable of injecting faults into TensorFlow 1, and not TensorFlow 2. This necessitates taking into account the variations among AI frameworks when designing FI tools and selecting a widely applicable method for FI [125], which might require a reanalysis of failures.

7 FAILURE ANALYSIS AND FAULT INJECTION FOR AI TOOLKIT

AI toolkit layer acts as an intermediate interface between the AI framework and underlying devices (e.g., GPU and NIC), facilitating the AI framework to utilize external functionalities in its implementation. The most commonly used AI toolkit is CUDA [133], a parallel computing runtime and API developed by NVIDIA. Besides CUDA, there are other GPU-specific toolkits available for AI and high-performance computing (HPC) solution development. Potential failures at this layer can include unavailability or incorrect outputs. This section delves into FA and FI in AI toolkits

7.1 Failure Analysis in AI Toolkit

In the following section, we will delve into and analyze the failures that typically occur within the AI toolkit. As shown in Table 10, these failures can be categorized into several types, including Synchronization Fault, Memory Safety Fault, Dependency Fault etc.

Synchronization Fault is frequently encountered due to the concurrent nature as GPU programs commonly operate using multiple threads. In contrast to CPUs that commonly utilize lock mechanisms for data synchronization, GPUs predominantly rely on barriers as their synchronization mechanism. In particular, a barrier is represented as a barrier function *__syncthreads()* in CUDA kernel functions [133]. There are primarily three main causes for synchronization faults: data race,

barrier divergence, and redundant barrier function [26, 195]. Data race primarily occurs due to the inability to determine the order of "read&write" and "write&write" actions among multiple threads [70, 195]. Barrier Divergence occurs when certain threads within the same block fail to reach a barrier due to variations in their execution flow. One common scenario is when a barrier (e.g., *__syncthreads()*) is positioned inside an if code block, leaving only a subset of threads can reach the barrier. Redundant Barrier Function is typically caused by unnecessary synchronization operations, which can result in inefficiencies in both speed and memory utilization of GPU programs.

Memory Safety Fault is frequently observed in low-level programming languages which provide direct memory access and management, particularly those specifically tailored for GPU programming(e.g., CUDA). Memory safety is typically ensured by the restriction that memory allocations can only be accessed between their intended bounds and during their lifetime [174, 213]. The primary causes of memory safety faults can be categorized into three factors: out-of-bounds access, temporal safety, and failed free operation [213]. In GPU programming, out-of-bounds access encompasses accessing buffers beyond their boundaries in global memory or shared memory. Temporal safety faults primarily arise when attempting to access GPU memory that has already been freed or accessing GPU memory that has not been properly allocated or initialized. Additionally, use-after-scope fault [213] can occur in local memory of GPU. Failed free operation includes double free and invalid free. Double free occurs when attempting to free memory that has already been freed, while invalid free refers to freeing memory that was not dynamically allocated.

Dependency Fault primarily occurs when there is a mismatch between the AI toolkit and the higher-level AI framework or AI application. Based on the number of components involved, dependency fault can be categorized into two types [65]: intra- and inter-dependency fault. Intra-dependency faults can occur due to incorrect versioning or unsuccessful installation of a toolkit. Unsuccessful installation can also be categorized into two types: missing installation of required libraries and incorrect path configuration. Inter-dependency faults in AI toolkit can occur can arise from a mismatch of software or hardware. An example of software mismatch is the mismatch between CUDA version and PyTorch version, which may occur a "driver too old" fault when running PyTorch [142]. Hardware mismatch can occur when the hardware lacks specific features required by the AI framework. For example, TensorFlow 1.6 utilizes the AVX feature of CPUs. However, if the CPU does not support AVX, it can lead to a dependency fault.

Communication Fault primarily occurs within the communication mechanism of distributed AI training. As two mostly common communication mechanisms in distributed AI, the faults occurring in NCCL [135] and NVLink can significantly impact the workload of distributed AI [45, 64], e.g., the training of LLM. The nccl fault is possibly due to a network fault or a remote process exiting prematurely [45]. And the NVLink fault is mainly caused by the hardware failures in GPU. Hu *et al.* [64] observes that training 7B models in Kalos tend to result in GPU overheating, which can cause NVLink fault. This phenomenon increases with the optimization of communication costs because it leads to more exceptionally low GPU idle rates. Another commonly used communication tool is MPI [123]. AI systems that utilize MPI also faces the faults associated with MPI itself. This kind of faults is usually due to either a fault of network connection to peer MPI process, or possibly an internal fault of the MPI daemon itself [73].

7.2 Fault Injection in AI Toolkits

In recent years, several FI techniques specifically designed for AI toolkits have emerged. In this section, we will elaborate on these works, categorizing them according to the different AI toolkits they target (details shown in Table 11).

Table 11. Fault Inject Tools to AI Toolkit

Tool	Description	Advantage	Instr.	Link
Simulee	A fuzzing framework for CUDA programs.	Automatically generate associated error-inducing test inputs.	False	[89]
CUDAsmith	A fuzzing framework for CUDA compiler.	Test several versions of NVCC and Clang compilers for CUDA with different optimization levels.	True	[49]
CLsmith	Investigate many-core compiler fuzzing in OpenCL.	Utilize many-core random differential testing and many-core EMI testing to detect bugs in OpenCL compilers.	True	[24]
FastFIT	A fault injection tool designed for MPI.	Inject faults randomly into input parameters of collective interface.endtabular	True	[43]

CUDA. As a parallel computing platform and application programming interface (API) developed by NVIDIA, CUDA offers AI systems high-performance and highly parallel capabilities. Recently, there are two main areas of work in fault injection at the CUDA level.

- **FI in CUDA Programs**. Simulee [194] utilizes LLVM bytecode to trace the execution of CUDA programs, enabling the detection of synchronization faults in CUDA. During the test input generation phase, Simulee incorporates the principles of fuzzing and introduces Evolutionary Programming [199] as a method to generate CUDA programs with built-in synchronization faults. Simulee introduces synchronization faults into CUDA programs by altering the dimensions and arguments that control the organization of threads within CUDA kernel functions.
- **FI in CUDA Compilers**. Faults in the CUDA compiler can either lead to compile-time faults or emit executable codes that may lead to runtime faults. CUDAsmith [77] proposes a FI framework for CUDA compilers, which can be used to test several versions of NVCC and Clang compilers for CUDA with different optimization levels. CUDAsmith continuously generates CUDA programs to detect potential bugs in the CUDA compiler and utilizes equivalence modulo inputs (EMI) testing techniques to solve the test oracle problem.

OpenCL. Open Computing Language (OpenCL) is a framework that provides programming capabilities on heterogeneous devices(e.g., GPUs, CPUs, and FPGAs) [86, 105]. Christopher *et al.* [96] investigate many-core compiler fuzzing in the context of OpenCL and introduce a tool, CLsmith. They utilize many-core random differential testing and many-core EMI testing to detect bugs in OpenCL compilers by injecting EMI blocks into existing OpenCL kernels.

Collective Communication. Collective communication is defined as communication that involves a group of processes, which plays a significant role in distributed AI scenarios. The intricate communication among different nodes poses significant challenges to the reliability of collective communication. FastFIT [43] is a fault injection tool specifically designed for MPI. It injects faults randomly into the input parameters of collective interface. In particular, FastFIT manifests the fault by one bit flip in one of the input parameters, which typically include the send/receive buffer address, data elements, data type, communication destination, and communicator.

7.3 Gap between Failure Analysis and Fault Injection in AI Tookit

Injecting faults into an AI toolkit is a complex process that comes with various challenges. As shown in Table 12, it is evident that there exists a significant gap between the capabilities of various fault injection tools in simulating specific types of faults within AI tookits.

Incomplete coverage of fault types. Table 12 highlights that certain types of faults, such as "Temporal Safety Fault", "Failed Free Operation", "Intra-dependency Fault", "NCCL Fault", and

Table 12. Gap between Failure Analysis and Fault Injection in AI Toolkit

Failure	Simulee	CUDAsmith	CLsmith	FastFIT	Covered
Data Race	✔				True
Barrier Divergence	✔			✔	True
Redundant Barrier Func.	✔		✔		True
Out-of-Bounds Access				✔	True
Temporal Safety Fault					False
Failed Free Operation					False
Intra-dependency Fault					False
Inter-dependency Fault		✔	✔	✔	True
NCCL Fault					False
NVLink Fault					False
MPI Fault				✔	True

"NVLink Fault", are not covered by the existing FI tools listed. This gap indicates that the current FI techniques may not comprehensively address the diverse range of faults that can occur in AI toolkits, potentially leaving blind spots in testing and resilience evaluation.

Lack of FI capabilities for distributed and parallel computing. While tools like FastFIT can inject faults related to MPI and inter-dependency faults, there is a lack of FI capabilities specifically targeting faults that can arise in distributed and parallel computing environments. AI toolkits increasingly leverage distributed and parallel computing for efficient model training and inference, making it crucial to have FI techniques that can simulate and analyze faults in these scenarios, such as "NCCL Fault" and "NVLink Fault".

8 FAILURE ANALYSIS AND FAULT INJECTION FOR AI PLATFORM

AI platform layer plays a crucial role in the overall architecture of an AI system. This layer serves as the foundation of the above layers. It abstracts the complexities of underlying hardware, offering a unified interface for functionalities like data management and sharing, workflow scheduling, and resource allocation. Failures in the AI platform layer can hinder data collaboration between different AI applications, cause scheduling failures for AI training or inference tasks, and so on. This section delves into the FA and FI in AI platforms.

8.1 Failure Analysis in AI Platform

In this section, we primarily introduce FA about Spark [203], Ray [121] and Platform-X in Microsoft [45], which are three representative AI platforms. Due to limited FA work in platform layer, we supplement several fault types based on the merged pull requests (PRs) that are responsible to fix bugs on GitHub [160]. Notably, not all pull requests in this category are exclusively for bug fixes. Some may focus on introducing new features or updating documentation. To specifically identify bug-fixing pull requests, we employ keyword searches in the tags and titles, leveraging established bug-related terms such as fix, defect, fault, bug, issue, mistake, correct, fault, and flaw, aligning with prior research [69, 169]. Table 13 shows the detailed failures of AI platforms.

Code Faults are prevalent in AI platforms due to their inherent complexity (e.g., intricate software stacks and distributed environment). Concurrency faults, often caused by race conditions or deadlocks, have been reported in several issues [150, 157–159]. API incompatibility issues, where the platform encounters compatibility problems with external APIs, have also been observed [155, 156, 207]. Misconfigurations, where incorrect system configurations lead to malfunctions, and inadequate access control mechanisms, allowing unauthorized access, have also been documented

Table 13. Failure Analysis of AI Platform

Group	Failure	Description	Paper or Issue
Code	Concurrency Fault	Concurrent faults caused by race condition or deadlock.	[150, 157–159]
	API Incompatibility	Incompatibility faults with external APIs.	[155, 156, 207]
	Misconfiguration	Incorrect system configurations leading to malfunctions.	[45, 154]
	Wrong Access Control	Inadequate permissions leading to unauthorized access.	[45, 64, 73]
	Exception Fault	Faults in exception handling mechanisms.	[151, 153]
	Memory Leak	Unreleased memory causing system slowdown.	[152, 175]
Platform Maintenance	Tool/Library Fault	Faults with outdated or incompatible tools/libraries.	[73, 154]
	Misoperation	Faults due to incorrect user operations.	[45, 64]
Platform Resource	Resource Contention	Resource sharing faults causing performance bottlenecks.	[45, 102, 168]
	Resource Overload	Excessive resource usage leading to system overload.	[45, 207]

as prevalent failures in AI platforms [45, 73]. Exception handling defects and memory leaks, which can cause system slowdowns or crashes, are other defects identified in the literature [151–153, 175].

Platform Maintenance Faults are common when performing regular platform maintenance, such as node additions and deletions, software upgrade, and other task. These include problems related to outdated or incompatible tools and libraries used within the platform [73, 154], as well as misoperations resulting from incorrect user actions or procedures [45].

Platform Resource Faults include resource contention and resource overload problem. Resource contention occurs when multiple components or workloads compete for shared resources, leading to performance bottlenecks [45, 102, 168]. Sarah *et al.* [168] and Lu *et al.* [102] analyze the performance impact caused by interference between Spark applications from the perspective of mutual interference and propose a technology that can quickly diagnoses the root cause of interference. Resource overload, on the other hand, refers to situations where excessive resource usage causes system overload and performance degradation [45, 207].

8.2 Fault Injection in AI Platform

In recent years, several FI techniques specifically designed for AI Platform have emerged. We have summarized these works as shown in Table 14.

As a distributed computing architecture, the communication between nodes and the influence between node states are commom faults in AI platforms, such as node partition caused by network fault and node recovery bugs caused by node crashes. Therefore, there are currently some related works that inject faults into these issues to discover corresponding bugs. ChaosBlade [18] can introduce resource hog faults into target system to test its resilience. Chen *et al.* propose a consistency-guided fault injection technique called CoFI to systematically injects network partitions to effectively expose partition bugs in distributed systems [20]. Gao *et al.* propose CrashFuzz, a fault injection testing approach that can effectively test crash recovery behaviors and reveal crash recovery bugs in distributed systems [44].

Data-intensive scalable computing (DISC) has become popular due to the increasing demands of analyzing big data. For example, Apache Spark and Hadoop allow developers to write dataflow-based applications with user-defined functions to process data with custom logic. Testing such applications is difficult. Many programming details in data processing code within Spark programs are prone to false statements that need to be correctly and automatically tested. Hence, João *et al.* propose TRANSMUT-SPARK, a tool that automates the mutation testing process of Big Data processing code within Spark programs [127]. Ahmad *et al.* propose DepFuzz [66] to increase the effectiveness and efficiency of fuzz testing dataflow-based big data applications such as Apache Spark-based DISC applications written in Scala.

Table 14. Fault Injection Tools to AI Platform

Tool	Description	Instrumented	Link
ChaosBlade	Inject resource contention fault at OS level.	False	[18]
CoFI	Inject network partition fault into cloud systems.	False	[20]
CrashFuzz	Inject crash or reboot faults into nodes.	False	[44]
TRANSMUT-SPARK	Automate mutation testing of data processing within Spark.	False	[127]
DepFuzz	A fuzzing framework for dataflow-based applications.	True	[66]

Table 15. Gap between Failure Analysis and Fault Injection in AI Platform

Failure	ChaosBlade	CoFI	CrashFuzz	TRANSMUT-SPARK	DepFuzz	Coverd
Concurrency Fault		✔				True
API Incompatibility				✔	✔	True
Misconfiguration						False
Wrong Access Control						False
Exception Fault			✔	✔	✔	True
Memory Leak						False
Tool/Library Fault						False
Misoperation			✔	✔	✔	True
Resource Contention (excl. GPU)	✔					True
Resource Contention (GPU)						False
Resource Overload						False

8.3 Gap between Failure Analysis and Fault Injection in AI Platform

From Table 15, two insights regarding the gap between FA and FI in AI platforms can be gleaned.

Limited Coverage. Not all types of failures are covered by the listed FI tools. For instance, failures due to misconfiguration, wrong access control, memory leaks, tool/library faults, and GPU resource contention are not being simulated by any of the tools. This indicates a significant gap in the current FI capabilities and highlights areas where further research are needed.

Lack of Specific Design. The current fault injection tools appear to be designed primarily for traditional cloud platforms, such as Kubernetes, with less consideration given to the unique characteristics and requirements of AI platforms. For instance, none of the listed tools can simulate failures related to GPU resource contention, which is a critical aspect of AI platforms due to their heavy reliance on GPU resources for computation-intensive tasks. This lack of specific design for AI platforms introduces a substantial gap in the ability to accurately simulate and study the full range of potential failures in these systems.

9 FAILURE ANALYSIS AND FAULT INJECTION FOR AI INFRASTRUCTURE

AI infrastructure layer serves as the foundational layer in an AI system architecture, providing the underlying physical and virtualized resources necessary for the deployment and operation of AI applications and services. This layer is responsible for managing and orchestrating the computing, storage, and networking resources required by the AI platform layer and other higher-level layers. Potential failures in this layer can lead to service unavailability, system deployment failures, model training failures, etc. This section delves into the FA and FI in AI infrastructure layer.

9.1 Failure Analysis in AI Infrastructure

In this section, we primarily introduce FA in the AI Infrastructure layer. Table 16 shows the detailed failures of AI infrastructure.

9.1.1 Hardware Accelerators. We analyze the failure of GPU, FPGA and TPU, which are three representative hardware accelerators.

GPU. GPU has become the most commonly used underlying hardware in AI and HPC systems. However, according to existing research, the frequency of failures caused by GPU is still high [176]. Research conducted on the Titan supercomputer explores different aspects of GPU failures. This includes the examination of GPU faults in a broader context [182], the analysis of specific GPU software faults [128], the characterization of GPU failures concerning temperature and power [129], and the investigation of spatial characteristics associated with failures [183]. Ostrouchov *et al.* [139] find that GPU reliability is dependent on heat dissipation to an extent that strongly correlates with detailed nuances of the cooling architecture and job scheduling. Nie *et al.* [129] analyze the relationship between single bit faults occurrence and temperature on the Titan supercomputer, and propose a machine learning based technique for GPU soft-fault prediction. A study about another supercomputer, Blue Water, analyzes GPU failures among other hardware failures [38]. The study reveals that GPUs rank among the top three most prone to failures, and notably, GPU memory exhibits greater sensitivity to uncorrectable faults compared to main memory.

Given the distinctions in workload between HPC and AI systems, the following discussion delves into GPU failure analysis specifically tailored to AI systems. Zhang *et al.* [207] present the first comprehensive empirical study on program failures of deep learning jobs and found that the GPU "Out of Memory" fault accounts for 65.0% of the failures in the deep learning specific dimension. Since in a large-scale deep learning cluster, GPU failures are inevitable and they cause severe consequences, Liu *et al.* [97] propose prediction models of GPU failures under large-scale production deep learning workloads. The prediction model takes into account static parameters such as GPU type, as well as dynamic parameters such as GPU temperature and power consumption, and integrates parallel and cascading architectures to make good predictions of GPU failures.

FPGA. FPGA is a digital technology designed to be configured by a customer or a designer after manufacturing, hence the term "field-programmable". As a neural network accelerator, FPGA is the subject of various studies related to reliability. Radu *et al.* [147] propose a new probabilistic method, the Component Failure Analysis (CFA), that uses FPGA specific techniques and algorithms for analyzing SEUs in implemented FPGA designs. McNelles *et al.* [115] use Dynamic Flowgraph Methodology (DFM) to model FPGA, showing the potential advantage of DFM for modeling FPGA-based systems compared with static methods and simulation.

Examining an FPGA from diverse perspectives leads to varied insights and advantages. Conmy *et al.* [29] employ a semi-automated FPTC analysis technique, customized for specific fault types identified on an FPGA, to thoroughly examine individual faults within electronic components. These components support a modularized design embedded on the FPGA. The study demonstrates how the analysis of these individual faults can be seamlessly integrated with crosscutting safety analysis, thereby reinforcing and validating the necessary safety properties. Xu *et al.* [197] take the entire FPGA-based neural network accelerator, including the control unit and DMA modules into consideration. The experiments on four typical neural networks showed that hardware faults can incur both system exceptions, such as system stall and prediction accuracy loss.

TPU. TPU is initially developed by Google to accelerate machine learning workloads, specifically targeting the training and inference of deep neural networks within the TensorFlow framework. Pablo *et al.* [13] measure TPU's atmospheric neutrons reliability at different temperatures, that goes from -40°C to +90°C. They show a decrease in the FIT rate of almost 4× as temperature increases. Rubens *et al.* [81] investigat the reliability of TPU executing 2D-3D convolutions and eight CNNs to high-energy, mono-energetic, and thermal neutron. They find that despite the high fault rate, most neutron-induced faults do not change the CNNs detection/classification. Rubens *et al.* [80] investigate the reliability of TPUs to atmospheric neutrons, reporting experimental data equivalent to more than 30 million years of natural irradiation.

Table 16. Failure Analysis of AI Infrastructure

Group	Failure	Description	Paper
Hardware Accelerator	Bit-flip Fault	Radiation or temperature changes cause data bits to flip.	[129, 130, 147]
	Stuck-at Fault	A circuit element stuck in a state.	[29, 38, 197]
	Out of Memory	Out of memory due to excessive workload.	[64, 207]
	Off the Bus	GPU loses the connection to host	[183]
Network	Network Jam	Heavy traffic slows data flow.	[83]
	Network Loss	Data packets fail to reach destination, disrupting communication.	[83]
	InfiniBand Fault	InfiniBand port down and other InfiniBand-related failures.	[45, 64]
	Ethernet Fault	Ethernet port down and other Ethernet-related failures.	[45, 64]
Node	Node Crash	OS kernel panic or ephemeral disk errors, causing nodes to fail.	[45, 64, 107]
	Node Partition	Abnormal communication leads to inconsistency between nodes.	[20]

Table 17. Fault Injection Tools to AI Infrastructure

Tool	Description	Instr.	Link
SASSIFI	Instrument low-level GPU assembly language (SASS) to inject faults.	True	[56]
LLFI-GPU	Operate on the LLVM intermediate representation (IR) to inject faults.	True	[92]
SCFIT	A FPGA-based fault injection technique for SEU fault model.	False	[41]
GPU-Qin	Inject faults based on the CUDA GPU debugging tool namely cuda-gdb.	False	[42]
NVBitFI	Instrument code dynamically to inject faults into GPU programs.	True	[186]
ThunderVolt	A framework allowing adaptive aggressive voltage underscaling.	False	[205]
ChaosBlade	Inject OS-level faults to simulate network faults and node faults.	False	[18]
ThorFI	Provide non-intrusive fault injection capabilities for a cloud tenant.	False	[31]
NetLoiter	Automate the simulation of network faults.	False	[161]
FCatch	Inject node crash fault to detect Time-of-fault bugs in cloud systems.	True	[99]
CoFI	Inject network partition fault into cloud system to expose partition bugs.	False	[20]

9.1.2 Network. Network failures have long been a significant area of research, particularly concerning traditional faults such as congestion, packet loss, and latency, which have been extensively discussed and studied. Hassan *et al.* [83] study how network faults occurring in the links between the nodes of the cloud management platforms can propagate and affect the applications that are hosted on the virtual machines.

However, with the advancement of AI systems, the networking infrastructure of AI systems has become increasingly complex, leading to the emergence of unique fault types. Distributed deep learning training across multiple compute nodes is pretty common and these nodes are internally interconnected with a high-speed network (e.g., via InfiniBand). Gao *et al.* [45] and C4 [39] classify network faults on AI platform into InfiniBand-related and Ethernet-related.

9.1.3 Node. A node in AI platform is a distinct schedulable unit for computation with GPUs, CPUs, main memory, disks, and network. Gao *et al.* [45] classify node faults on AI platform into node outage, node damage and node preemption. These faults summarize the impact caused by faults occurring inside the node. In addition to these faults, communication faults between nodes are also of concern [20]. Thus, we classify node faults into two types, namely node crash, node partition.

9.2 Fault Injection in AI Infrastructure

9.2.1 Hardware Accelerator. **GPU.** Research on FI in GPUs is rich and can be broadly categorized into three types including Software, Hardware/Simulation and Hybrid.

- **Software.** At present, various FI techniques exist at different levels of programming languages. Commonly, faults are injected at the GPU assembly code (SASS) level, which is the instruction-level code running directly on the GPU. For instance, SASSIFI [56] employs the SASSI (SASS Instrumentation) framework for compile-time instrumentation of SASS code to insert fault injection code. GPU-Qin [42] utilizes CUDA-GDB to control faults during runtime without

modifying the code. NVBitFI [186] dynamically loads relevant code as a library during runtime for fault injection. Besides SASS level, there are also works at the PTX and LLVM IR levels. For instance, LLFI-GPU [92] improves FI in LLVM IR, the intermediate representation language.

- **Hardware/Simulation.** Hardware or simulation-based approaches provide a more realistic reflection of fault impacts. Direct methods include radiation experiments to evaluate hardware reliability. For example, Oliveira *et al.* [136] use beam tests to study the radiation effects on NVIDIA and Intel accelerators, quantifying and limiting radiation effects by observing amplitude and fault propagation in final outputs. Simulation approaches replace actual hardware faults (e.g., electromagnetic interferences at the physical level) by injecting their expected effects on memory and registers (e.g., flipped and stuck bits), thus approximating the hardware fault process [35, 180, 206]. They simulate faults at different levels, such as modifying simulator variables, introducing faults at the RTL level, and injecting faults at the gate level. The NVIDIA Data Center GPU Manager (DCGM) includes an fault injection framework allows users to simulate the fault handling behavior of the DCGM APIs when GPU faults are encountered [134].
- **Hybrid.** Some research endeavors have sought to combine software and hardware-level approaches. Josie *et al.* [28] combine the accuracy of microarchitecture simulation with the speed of software-level FI. It performs detailed microarchitecture FI on a GPU model (FlexGripPlus), describing the impact of faults on convolutional calculations.

FPGA. Compared to GPUs, the programmability of FPGAs makes it easier to implement hardware-level fault injection. There are two major classes for FPGA-based fault injection methods.

- **Reconfiguration-based techniques.** In reconfiguration-based techniques, faults are injected by changing the bit stream needed for configuring FPGA. Antoni *et al.* introduce a novel methodology for injecting single event upsets (SEUs) in memory elements. This approach involves conducting the injection directly within the reconfigurable FPGA, leveraging the runtime reconfiguration capabilities of the device [6]. Gabriel *et al.* propose a fault injection tool to evaluate the impact of faults in an FPGA's configuration memory [126].
- **Instrumentation-based techniques.** In instrumentation-based techniques, supplementary circuits are incorporated into the original circuits, and both are integrated within the FPGA after synthesis. Mojtaba *et al.* propose an FPGA-based fault injection technique [41], which utilizes debugging facilities of Altera FPGAs in order to inject single event upset (SEU) and multiple bit upset (MBU) fault models in both flip-flops and memory units. Pierluigi *et al.* propose a method that utilizes FPGA devices to emulate systems and employs an innovative system instrumentation approach for fault injection. This approach significantly reduces experimental time without requiring FPGA reconfiguration, achieving notable performance improvements in both compute-intensive and input/output-intensive applications [25].

TPU. TPU, as a variant of systolic arrays, represents a parallel computing architecture. The intrinsic parallelism and matrix multiplication efficiency inherent in systolic arrays empower them to achieve superior performance in both the training and inference phases of deep neural networks. Numerous studies have been conducted to investigate faults associated with systolic arrays. Udit *et al.* [3] propose an RTL-level fault injection framework for systolic arrays. Using this framework, they characterized the software effect of faults induced by stuck-at faults within the multiply and accumulation units of the systolic array. Zhang *et al.* [205] and Holst *et al.* [61] study the effects of timing faults in systolic arrays, thus, degrading DNN's accuracy.

9.2.2 Network. In addition to general purpose fault injection tools such as ChaosBlade [18] that can introduce common network faults such as network delay and network packet loss, there are now some network fault injection tools for large infrastructure. Domenico *et al.* propose ThorFI [31], a

Table 18. Gap between Failure Analysis and Fault Injection in AI Infrastructure

Failure	SASSIFI	LLFI-GPU	SCFIT	NVBitFI	ThunderVolt	ThorFI	NetLoiter	FCatch	CoFI	Covered
Bit-flip Fault	✔	✔	✔	✔	✔					True
Stuck-at Fault				✔	✔					True
Out of Memory										False
Off the Bus										False
Network Jam						✔	✔			True
Network Loss							✔	✔		True
InfiniBand Fault										False
Ethernet Fault										False
Node Crash						✔		✔		True
Node Partition									✔	True

novel approach for virtual networks in infrastructures. ThorFI is designed to provide non-intrusive fault injection capabilities for a cloud tenant, and to isolate injections from interfering with other tenants on the infrastructure. Michal *et al.* propose NetLoiter [161], which can introduce real world fault, including lossy channels, network jitter, data corruption, or disconnections.

9.2.3 Node. Currently, the work of fault injection for nodes mainly focuses on two aspects. On the one hand, it is aimed at the failure of the node itself, which means injecting OS-level faults into the virtual machine or host machine to simulate faults such as node outage. This kind of fault can be implemented by general fault injection tools such as ChaosBlade [18]. On the other hand, node crash is simulated in the process of communication between nodes to test the reliability of the whole system [20, 44, 99].

9.3 Gap between Failure Analysis and Fault Injection in AI Infrastructure

From the Table 18, several key observations regarding the gap between FA and FI at the infrastructure level in AI systems can be made:

Limited coverage of fault types. The table shows that several types of faults, such as "Out of Memory", "Off the Bus", "InfiniBand Fault", and "Ethernet Fault", are not currently being simulated by any of the listed FI tools. This suggests a significant gap in the ability of current tools to simulate a comprehensive set of failure scenarios at the infrastructure level in AI systems.

Hardware and network-specific FI The existing tools seem to focus on specific areas of the infrastructure. For example, "Bit-flip Fault" and "Stuck-at Fault" are well-covered by tools designed for hardware faults like SASSIFI, LLFI-GPU, and ThunderVolt. On the other hand, network-related faults such as "Network Jam", "Network Loss" are covered by NetLoiter, FCatch. This suggests that the current fault injection tools are specialized for either hardware or network faults but not both.

Lack of realism in hardware accelerator faults. Most of the existing fault injection tools for hardware accelerators, such as GPUs, operate at the software level or are based on simulations. The faults generated by these methods can be classified as emulated faults. While emulation provides high efficiency, a significant drawback is that the faults may lack realism. This is because emulated faults may not accurately represent the complex physical processes that cause real hardware faults. As a result, the conclusions drawn from studies using these tools may not fully apply to real-world scenarios where hardware faults occur. This lack of realism in emulated faults represents another significant gap in the current state of fault injection for AI systems.

10 FUTURE OPPORTUNITIES OF FAULT INJECTION IN AI SYSTEMS

FI is a widely used technique for evaluating the reliability of AI systems. However, as discussed above, there is a huge gap between FA and FI. Bridging this gap is a research opportunity in future.

10.1 More Comprehensive Fault Injection

Support for more faults types. As shown in previous sections, there are a number of real faults that have occurred that cannot yet be simulated by existing FI tools. For example, no FI tool in Table 14 can simulate GPU contention faults in Table 13. Ignoring these faults leads to incomplete FI testing and may hide risks in AI systems. Therefore, based on the FA results in this paper, designing FI tools that cover as much as possible all the faults that have occurred historically can provide a more comprehensive assessment of the reliability and fault tolerance of AI systems.

Cross-layer multiple fault injection. Current FI tools typically only inject a fault at a single layer. However, in distributed systems, there are situations where multiple faults occur simultaneously [200]. Designing tools that support simultaneous injection of multiple faults can facilitate the creation of more complex fault scenarios. At the same time, considering the dependencies between AI system layers can also enable the simulation of richer fault scenarios through cross-layer fault linkage injection. Consequently, the development of tools that facilitate cross-layer multiple fault injection will prove advantageous in evaluating the reliability and fault-tolerance of AI systems when multiple faults occur at different layers simultaneously.

LLM-specific FI tool. As LLMs continue to gain prominence in both academic and industrial sectors, the importance of assessing their reliability cannot be overstated. Given the unique characteristics and potential failure of LLMs, there is a pressing need for the development of FI tools specifically designed for these systems. Such tools should be capable of simulating LLM-specific failures, including those related to language understanding, reasoning, and generation.

10.2 More Generalizable Fault Injection

Compatible with more layers and frameworks. From the perspective of FI generality, current FI tools are in a state of fragmentation. For example, PytorchFI [144] can only inject faults related to PyTorch, while TensorFI [36] can only inject faults to TensorFlow. Even for the same AI framework, there may be conceptual differences between versions. For example, TensorFlow 1 and TensorFlow 2 exhibit significant differences in API usage and runtime logic, requiring separate fault injection tools (e.g., TensorFI [36] and InjectTF [114]) to be designed for them. This results in a considerable number of FI tools that engineers must maintain, as well as a significant amount of time required to learn to use them. Consequently, the design of a more unified tool that can inject faults across different layers and across different frameworks is of great importance in the future.

Non-instrumented injection. Numerous contemporary FI tools in Table 8 and Table 11 necessitate the instrumentation of the target for FI (e.g., modifying the framework source code). This increases the difficulty of utilising FI tools. Given that the majority of frameworks and algorithms associated with AI systems are implemented in Python, it is possible to implement python bytecode modifications that do not necessitate code instrumentation, as was previously the cases [84, 202]. Even for non-Python implementations, it is possible to achieve non-instrumented FI through eBPF [185, 201]. Future research into work injection without code instrumentation could facilitate the development of more user-friendly FI tools.

10.3 More Intelligent Fault Injection

A FI policy must specify the location of the FI, the type and intensity of the fault, and so forth. This combination of several attributes forms a vast search space for FI policies. The objective is to identify valuable FI policies from this vast search space and to discover as many faults as possible with as few fault injections as possible. Currently, this process relies mainly on expert experience, resulting in high and inefficient labour costs. In the future, it is anticipated that intelligent algorithms will be introduced to facilitate the selection of fault injection strategies in an intelligent manner.

LLM-based Fault Injection. LLM has already demonstrated its capabilities in several software engineering tasks [141, 192, 196]. In the future, the integration of LLM and Reinforcement Learning from Human Feedback (RLHF) will enable the translation of natural language descriptions of fault scenarios directly into FI policies, thereby reducing the manual effort required to design and implement fault scenarios in AI systems [30].

11 CONCLUSION

In this study, we have examined the current state of FA and FI in AI systems, providing a critical overview of prevalent failures, the capabilities of existing FI tools, and the gaps between simulated and actual failures. Our analysis, based on a thorough review of relevant paper and code repositories, has revealed significant gaps in the ability of current FI tools to simulate the wide range of failures that occur in real-world AI systems. Moreover, this survey contributes by discussing technical challenges of FI in AI systems and outlining future research avenues. The findings of this study serve as a foundation for further advancements in the field of FA and FI for AI systems.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI 2016*. USENIX Association, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[2] Sahar Abdelnabi, Kai Greshake, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *AISec 2023*. ACM, 79–90. https://doi.org/10.1145/3605764.3623985

[3] Udit Kumar Agarwal, Abraham Chan, Ali Asgari, and Karthik Pattabiraman. 2023. Towards Reliability Assessment of Systolic Arrays against Stuck-at Faults. In *DSN 2023)*. 230–236. https://doi.org/10.1109/DSN-S58398.2023.00063

[4] Naveed Akhtar, Ajmal Mian, Navid Kardan, and Mubarak Shah. 2021. Advances in Adversarial Attacks and Defenses in Computer Vision: A Survey. *IEEE Access* 9 (2021), 155161–155196. https://doi.org/10.1109/ACCESS.2021.3127960

[5] Alexei95. 2024. Enpheeph Github. https://github.com/Alexei95/enpheeph. Accessed 2024.

[6] Lörinc Antoni, Régis Leveugle, and Béla Fehér. 2002. Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes. In *DFT 2002*. IEEE, 245–253. https://doi.org/10.1109/DFTVS.2002.1173521

[7] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (2004), 11–33. https://doi.org/10.1109/TDSC.2004.2

[8] Fu Bang. 2023. GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings. In *NLP-OSS 2023*. Empirical Methods in Natural Language Processing, 212–218. https://doi.org/10.18653/v1/2023.nlposs-1.24

[9] S. H. Shabbeer Basha, Shiv Ram Dubey, Viswanath Pulabaigari, and Snehasis Mukherjee. 2020. Impact of fully connected layers on performance of convolutional neural networks for image classification. *Neurocomputing* 378 (2020), 112–119. https://doi.org/10.1016/J.NEUCOM.2019.10.008

[10] Michael Beyer, Andrey Morozov, Emil Valiev, Christoph Schorn, Lydia Gauerhof, Kai Ding, and Klaus Janschek. 2020. Fault Injectors for TensorFlow: Evaluation of the Impact of Random Hardware Faults on Deep CNNs. *CoRR* abs/2012.07037 (2020). https://arxiv.org/abs/2012.07037

[11] Bfgoldstein. 2024. Torchfi Github. https://github.com/bfgoldstein/torchfi. Accessed 2024.

[12] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive Online Analysis. *J. Mach. Learn. Res.* 11 (2010), 1601–1604. https://api.semanticscholar.org/CorpusID:12397401

[13] Pablo R. Bodmann and Paolo Rech. 2024. Tensor Processing Unit Reliability Dependence on Temperature and Radiation Source. *IEEE Transactions on Nuclear Science* (2024), 1–1. https://doi.org/10.1109/TNS.2024.3359524

[14] Jakub Breier, Dirmanto Jap, Xiaolu Hou, Shivam Bhasin, and Yang Liu. 2022. SNIFF: Reverse Engineering of Neural Networks With Fault Attacks. *IEEE Transactions on Reliability* 71, 4 (2022), 1527–1539. https://doi.org/10.1109/TR.2021.3105697

[15] Junming Cao, Bihuan Chen, Chao Sun, Longjie Hu, Shuaihong Wu, and Xin Peng. 2022. Understanding performance problems in deep learning systems. In *ESEC/FSE 2022*. ACM, 357–369. https://doi.org/10.1145/3540250.3549123

[16] Abraham Chan, Arpan Gujarati, Karthik Pattabiraman, and Sathish Gopalakrishnan. 2022. The Fault in Our Data Stars: Studying Mitigation Techniques against Faulty Training Data in Machine Learning Applications. In *DSN 2022*. IEEE/IFIP, 163–171. https://doi.org/10.1109/DSN53405.2022.00027

[17] Abraham Chan, Niranjhana Narayanan, Arpan Gujarati, Karthik Pattabiraman, and Sathish Gopalakrishnan. 2021. Understanding the Resilience of Neural Network Ensembles against Faulty Training Data. In *QRS 2021*. IEEE, 1100–1111. https://doi.org/10.1109/QRS54544.2021.00118

[18] Chaosblade. 2024. Chaosblade Github. https://github.com/chaosblade-io/chaosblade.

[19] Hongyang Chen, Pengfei Chen, Guangba Yu, Xiaoyun Li, Zilong He, and Huxing Zhang. 2024. MicroFI: Non-Intrusive and Prioritized Request-Level Fault Injection for Microservice Applications. *IEEE Transactions on Dependable and Secure Computing* (2024), 1–18. https://doi.org/10.1109/TDSC.2024.3363902

[20] Haicheng Chen, Wensheng Dou, Dong Wang, and Feng Qin. 2020. CoFI: Consistency-Guided Fault Injection for Cloud Systems. In *ASE 2020*. IEEE, 536–547. https://doi.org/10.1145/3324884.3416548

[21] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. 2020. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. In *SP 2020*. IEEE, 1277–1294. https://doi.org/10.1109/SP40000.2020.00045

[22] Junjie Chen, Yihua Liang, Qingchao Shen, Jiajun Jiang, and Shuochuan Li. 2023. Toward Understanding Deep Learning Framework Bugs. *ACM TOSEM* 32, 6, Article 135 (2023), 31 pages. https://doi.org/10.1145/3587155

[23] Zitao Chen, Niranjhana Narayanan, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2020. TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications. In *ISSRE 2020*. IEEE, 426–435. https://doi.org/10.1109/ISSRE5003.2020.00047

[24] ChrisLidbury. 2024. CLSmith Github. https://github.com/ChrisLidbury/CLSmith/. Accessed 2024.

[25] Pierluigi Civera, Luca Macchiarulo, Maurizio Rebaudengo, Matteo Sonza Reorda, and Massimo Violante. 2001. FPGA-Based Fault Injection for Microprocessor Systems. In *ATS 2001*. IEEE, 304. https://doi.org/10.1109/ATS.2001.990301

[26] Peter Collingbourne, Alastair F. Donaldson, Jeroen Ketema, and Shaz Qadeer. 2013. Interleaving and lock-step semantics for analysis and verification of GPU kernels. In *ESOP 2013*. Springer, 270–289. https://doi.org/10.1007/978-3-642-37036-6_16

[27] Alessio Colucci, Andreas Steininger, and Muhammad Shafique. 2022. enpheeph: A Fault Injection Framework for Spiking and Compressed Deep Neural Networks. In *IROS 2022*. 5155–5162. https://doi.org/10.1109/IROS47612.2022.9982181

[28] Josie E. Rodriguez Condia, Fernando Fernandes dos Santos, Matteo Sonza Reorda, and Paolo Rech. 2021. Combining Architectural Simulation and Software Fault Injection for a Fast and Accurate CNNs Reliability Evaluation on GPUs. In *VTS 2021*. IEEE, 1–7. https://doi.org/10.1109/VTS50974.2021.9441044

[29] Philippa Conmy and Iain Bate. 2010. Component-based safety analysis of FPGAs. *IEEE Transactions on Industrial Informatics* 6, 2 (2010), 195–205. https://doi.org/10.1109/TII.2009.2039938

[30] Domenico Cotroneo and Pietro Liguori. 2024. Neural Fault Injection: Generating Software Faults from Natural Language. *CoRR* abs/2404.07491 (2024). https://doi.org/10.48550/ARXIV.2404.07491

[31] Domenico Cotroneo, Luigi De Simone, and Roberto Natella. 2022. ThorFI: a Novel Approach for Network Fault Injection as a Service. *J. Netw. Comput. Appl.* 201 (2022), 103334. https://doi.org/10.1016/J.JNCA.2022.103334

[32] Roland Croft, Muhammad Ali Babar, and M. Mehdi Kholoosi. 2023. Data Quality for Software Vulnerability Datasets. In *ICSE 2023*. IEEE, 121–133. https://doi.org/10.1109/ICSE48619.2023.00022

[33] Saikat Das. 2022. 10 Frequently Encountered Issues in Data Preprocessing. https://www.analyticsvidhya.com/blog/2022/08/10-frequently-encountered-issues-in-data-preprocessing/.

[34] Jiachao Deng, Yuntan Fang, Zidong Du, Ying Wang, Huawei Li, Olivier Temam, Paolo Ienne, David Novo, Xiaowei Li, Yunji Chen, and Chengyong Wu. 2015. Retraining-based timing error mitigation for hardware neural networks. In *DATE 2015*. ACM, 593–596. http://dl.acm.org/citation.cfm?id=2755887

[35] Jiachao Deng, Yuntan Fang, Zidong Du, Ying Wang, Huawei Li, Olivier Temam, Paolo Ienne, David Novo, Xiaowei Li, Yunji Chen, and Chengyong Wu. 2015. Retraining-based timing error mitigation for hardware neural networks. In *DATE 2015*. ACM, 593–596. http://dl.acm.org/citation.cfm?id=2755887

[36] DependableSystemsLab. 2024. TensorFI Github. https://github.com/DependableSystemsLab/TensorFI. Accessed 2024.

[37] DependableSystemsLab. 2024. TensorFI2 Github. https://github.com/DependableSystemsLab/TensorFI2. Accessed 2024.

[38] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K. Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer. 2014. Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters. In *DSN 2014*. 610–621. https://doi.org/10.1109/DSN.2014.62

[39] Jianbo Dong, Bin Luo, Jun Zhang, Pengcheng Zhang, Fei Feng, Yikai Zhu, Ang Liu, Zian Chen, Yi Shi, Hairong Jiao, Gang Lu, Yu Guan, Ennan Zhai, Wencong Xiao, Hanyu Zhao, Man Yuan, Siran Yang, Xiang Li, Jiamang Wang, Rui Men, Jianwei Zhang, Huang Zhong, Dennis Cai, Yuan Xie, and Binzhang Fu. 2024. Boosting Large-scale Parallel Training Efficiency with C4: A Communication-Driven Approach. arXiv:2406.04594

[40] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. 2022. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing* 503 (2022), 92–108. https://doi.org/10.1016/J.NEUCOM.2022.06.111

[41] Mojtaba Ebrahimi, Abbas Mohammadi, Alireza Ejlali, and Seyed Ghassem Miremadi. 2014. A fast, flexible, and easy-to-develop FPGA-based fault injection technique. *Microelectron. Reliab.* 54, 5 (2014), 1000–1008. https://doi.org/10.1016/J.MICROREL.2014.01.002

[42] Bo Fang, Karthik Pattabiraman, Matei Ripeanu, and Sudhanva Gurumurthi. 2014. GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications. In *ISPASS 2014*. IEEE, 221–230. https://doi.org/10.1109/ISPASS.2014.6844486

[43] Kun Feng, Manjunath Gorentla Venkata, Dong Li, and Xian-He Sun. 2015. Fast Fault Injection and Sensitivity Analysis for Collective Communications. In *CLUSTER 2015*. IEEE, 148–157. https://doi.org/10.1109/CLUSTER.2015.31

[44] Yu Gao, Wensheng Dou, Dong Wang, Wenhan Feng, Jun Wei, Hua Zhong, and Tao Huang. 2023. Coverage Guided Fault Injection for Cloud Systems. In *ICSE 2023*. IEEE, 2211–2223. https://doi.org/10.1109/ICSE48619.2023.00186

[45] Yanjie Gao, Xiaoxiang Shi, Haoxiang Lin, Hongyu Zhang, Hao Wu, Rui Li, and Mao Yang. 2023. An Empirical Study on Quality Issues of Deep Learning Platform. In *ICSE-SEIP 2023*. IEEE, 455–466. https://doi.org/10.1109/ICSE-SEIP58684.2023.00052

[46] Bahar Gezici and Ayça Kolukisa Tarhan. 2022. Systematic literature review on software quality for AI-based software. *Empir. Softw. Eng.* 27, 3 (2022), 66. https://doi.org/10.1007/S10664-021-10105-2

[47] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. 2022. How to fight production incidents?: an empirical study on a large-scale cloud service. In *SoCC 2022*. ACM, 126–141. https://doi.org/10.1145/3542929.3563482

[48] Brunno F. Goldstein, Sudarshan Srinivasan, Dipankar Das, Kunal Banerjee, Leandro Santiago de Araújo, Victor da Cruz Ferreira, Alexandre Solon Nery, Sandip Kundu, and Felipe M. G. França. 2020. Reliability Evaluation of Compressed Deep Learning Models. In *LASCAS 2020*. IEEE, 1–5. https://doi.org/10.1109/LASCAS45839.2020.9069026

[49] Gongbell. 2024. CUDAsmith Github. https://github.com/gongbell/CUDAsmith. Accessed 2024.

[50] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR 2015*. http://arxiv.org/abs/1412.6572

[51] Google. 2024. Google Gemini. https://ai.google.dev/. Accessed 2024.

[52] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. 2016. Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages. In *SoCC 2016*. ACM, 1–16. https://doi.org/10.1145/2987550.2987583

[53] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. 2017. Failures in Large Scale Systems: Long-Term Measurement, Analysis, and Implications. In *SC 2017*. ACM, Article 44, 12 pages. https://doi.org/10.1145/3126908.3126937

[54] Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. 2023. Finding Neurons in a Haystack: Case Studies with Sparse Probing. *CoRR* abs/2305.01610 (2023). https://doi.org/10.48550/ARXIV.2305.01610

[55] Saad Hikmat Haji and Adnan Mohsin Abdulazeez. 2021. Comparison of optimization techniques based on gradient descent algorithm: A review. *Journal of Archaeology of Egypt/Egyptology* 18, 4 (2021), 2715–2743.

[56] Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W. Keckler, and Joel S. Emer. 2017. SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation. In *ISPASS 2017*. IEEE, 249–258. https://doi.org/10.1109/ISPASS.2017.7975296

[57] Fengxiang He, Tongliang Liu, and Dacheng Tao. 2019. Control Batch Size and Learning Rate to Generalize Well: Theoretical and Empirical Evidence. In *NeurIPS 2019*. 1141–1150. https://proceedings.neurips.cc/paper/2019/hash/dc6a70712a252123c40d2adba6a11d84-Abstract.html

[58] Dan Hendrycks and Thomas G. Dietterich. 2019. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *ICLR 2019*. OpenReview.net. https://openreview.net/forum?id=HJz6tiCqYm

[59] Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. 2023. How Good Are GPT Models at Machine Translation? A Comprehensive Evaluation. arXiv:2302.09210 [cs.CL]

[60] Zhe Zhang Hien Luu, Max Pumperla. 2024. *MLOps with Ray: Best Practices and Strategies for Adopting Machine Learning Operations*. Apress Media LLC.

[61] Stefan Holst, Lim Bumun, and Xiaoqing Wen. 2021. GPU-Accelerated Timing Simulation of Systolic-Array-Based AI Accelerators. In *ATS 2021*. IEEE, 127–132. https://doi.org/10.1109/ATS52891.2021.00034

[62] Qiang Hu, Lei Ma, Xiaofei Xie, Bing Yu, Yang Liu, and Jianjun Zhao. 2019. DeepMutation++: A Mutation Testing Framework for Deep Learning Systems. In *ASE 2019*. IEEE, 1158–1161. https://doi.org/10.1109/ASE.2019.00126

[63] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. Characterization and prediction of deep learning workloads in large-scale GPU datacenters. In *SC 2021*. ACM, 104.

[64] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, Yonggang Wen, and Tianwei Zhang. 2024. Characterization of Large Language Model Development in the Datacenter. In *NSDI 2024*. USENIX Association, 709–729. https://www.usenix.org/conference/nsdi24/presentation/hu

[65] Kaifeng Huang, Bihuan Chen, Susheng Wu, Junming Cao, Lei Ma, and Xin Peng. 2023. Demystifying Dependency Bugs in Deep Learning Stack. In *ESEC/FSE 2023*. ACM, 450–462. https://doi.org/10.1145/3611643.3616325

[66] Ahmad Humayun, Miryung Kim, and Muhammad Ali Gulzar. 2023. Co-dependence Aware Fuzzing for Dataflow-Based Big Data Analytics. In *ESEC/FSE 2023*. ACM, 1050–1061. https://doi.org/10.1145/3611643.3616298

[67] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. 2020. Taxonomy of Real Faults in Deep Learning Systems. In *ICSE 2020*. ACM, 1110–1121. https://doi.org/10.1145/3377811.3380395

[68] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. DeepCrime: Mutation Testing of Deep Learning Systems Based on Real Faults. In *ISSTA 2021*. ACM, 67–78. https://doi.org/10.1145/3460319.3464825

[69] Md Johirul Islam, Giang Nguyen, Rangeet Pan, and Hridesh Rajan. 2019. A Comprehensive Study on Deep Learning Bug Characteristics. In *ESEC/FSE 2019*. ACM, 510–520. https://doi.org/10.1145/3338906.3338955

[70] Mohammad Majharul Islam and Abdullah Muzahid. 2018. Bugaroo: Exposing Memory Model Bugs in Many-Core Systems. In *ISSRE 2018*. 178–188. https://doi.org/10.1109/ISSRE.2018.00028

[71] Istio. 2024. Istio HTTPDirectResponse Fault Injection. https://istio.io/latest/docs/reference/config/networking/virtual-service/#HTTPDirectResponse.

[72] Istio. 2024. Istio Network Fault Injection. https://istio.io/latest/docs/tasks/traffic-management/fault-injection/.

[73] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, unjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *USENIX ATC 2019*. USENIX, 947–960. https://www.usenix.org/conference/atc19/presentation/jeon

[74] Li Jia, Hao Zhong, Xiaoyin Wang, Linpeng Huang, and Zexuan Li. 2022. How Do Injected Bugs Affect Deep Learning?. In *SANER 2022*. IEEE, 793–804. https://doi.org/10.1109/SANER53432.2022.00097

[75] Li Jia, Hao Zhong, Xiaoyin Wang, Linpeng Huang, and Xuansheng Lu. 2020. An Empirical Study on Bugs Inside TensorFlow. In *DASFAA 2020*, Vol. 12112. Springer, 604–620. https://doi.org/10.1007/978-3-030-59410-7_40

[76] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE TSE* 37, 5 (2011), 649–678. https://doi.org/10.1109/TSE.2010.62

[77] Bo Jiang, Xiaoyan Wang, Wing Kwong Chan, T. H. Tse, Na Li, Yongfeng Yin, and Zhenyu Zhang. 2020. CUDAsmith: A Fuzzer for CUDA Compilers. In *COMPSAC 2020*. IEEE, 861–871. https://doi.org/10.1109/COMPSAC48688.2020.0-156

[78] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. 2024. MegaScale: Scaling Large Language Model Training to More Than 10, 000 GPUs. In *NSDI 2024*. USENIX Association, 745–760. https://www.usenix.org/conference/nsdi24/presentation/jiang-ziheng

[79] Michael Washburn Jr., Pavithra Sathiyanarayanan, Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2016. What went right and what went wrong: an analysis of 155 postmortems from game development. In *ICSE Companion Volume 2016*. ACM, 280–289. https://doi.org/10.1145/2889160.2889253

[80] Rubens Luiz Rech Junior, Carlo Cazzaniga, Maria Kastriotou, Christopher Frost, and Paolo Rech. 2022. Sensitivity of Google's Tensor Processing Units to High-Energy, Mono-Energetic, and Thermal Neutrons. In *RADECS 2022*. 1–6. https://doi.org/10.1109/RADECS55911.2022.10412572

[81] Rubens Luiz Rech Junior and Paolo Rech. 2022. Reliability of Google's Tensor Processing Units for Embedded Applications. In *DATE 2022*. IEEE, 376–381. https://doi.org/10.23919/DATE54114.2022.9774600

[82] Keras. 2024. Keras. https://keras.io/. Accessed 2024.

[83] Hassan Mahmood Khan, Frederico Cerveira, Tiago Cruz, and Henrique Madeira. 2023. Network Failures in Cloud Management Platforms: A Study on OpenStack. In *CLOSER 2023*. SCITEPRESS, 228–235. https://doi.org/10.5220/0011851400003488

[84] Kmaork. 2024. Inject python code into a running python process. https://github.com/kmaork/hypno. Accessed 2024.

[85] Tom Kocmi and Christian Federmann. 2023. Large Language Models Are State-of-the-Art Evaluators of Translation Quality. arXiv:2302.14520 [cs.CL]

[86] Yongbon Koo, Sunghoon Kim, and Young-Guk Ha. 2021. OpenCL-Darknet: implementation and optimization of OpenCL-based deep learning object detection framework. *World Wide Web* 24, 4 (2021), 1299–1319. https://doi.org/10.1007/S11280-020-00778-Y

[87] Alyssa Lamberti. 2023. 16 Most Common Network Problems: How to Find & Fix Them. https://obkio.com/blog/common-network-problems/. Accessed 2024.

[88] Raz Lapid and Moshe Sipper. 2023. I See Dead People: Gray-Box Adversarial Attack on Image-To-Text Models. *CoRR* abs/2306.07591 (2023). https://doi.org/10.48550/ARXIV.2306.07591
[89] Lebronmydx. 2024. Simulee Github. https://github.com/Lebronmydx/Simulee. Accessed 2024.
[90] Valentina Lenarduzzi, Francesco Lomio, Sergio Moreschini, Davide Taibi, and Damian Andrew Tamburri. 2021. Software Quality for AI: Where We Are Now?. In *SWQD 2021*, Vol. 404. Springer, 43–53. https://doi.org/10.1007/978-3-030-65854-0_4
[91] Guanpeng Li, Siva Kumar Sastry Hari, Michael B. Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel S. Emer, and Stephen W. Keckler. 2017. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *SC 2017*. ACM, 8. https://doi.org/10.1145/3126908.3126964
[92] Guanpeng Li, Karthik Pattabiraman, Chen-Yong Cher, and Pradip Bose. 2016. Understanding error propagation in GPGPU applications. In *SC 2016*. IEEE, 240–251. https://doi.org/10.1109/SC.2016.20
[93] Xiaoyun Li, Guangba Yu, Pengfei Chen, Hongyang Chen, and Zhekang Chen. 2022. Going through the Life Cycle of Faults in Clouds: Guidelines on Fault Handling. In *ISSRE 2022*. IEEE, 121–132. https://doi.org/10.1109/ISSRE55969.2022.00022
[94] Zengyang Li, Sicheng Wang, Wenshuo Wang, Peng Liang, Ran Mo, and Bing Li. 2023. Understanding Bugs in Multi-Language Deep Learning Frameworks. In *ICPC 2023*. IEEE, 328–338. https://doi.org/10.1109/ICPC58990.2023.00047
[95] Weixin Liang, Girmaw Abebe Tadesse, Daniel E. Ho, Li Fei-Fei, Matei Zaharia, Ce Zhang, and James Zou. 2022. Author Correction: Advances, challenges and opportunities in creating data for trustworthy AI. *Nat. Mac. Intell.* 4, 10 (2022), 904. https://doi.org/10.1038/S42256-022-00548-7
[96] Christopher Lidbury, Andrei Lascu, Nathan Chong, and Alastair F. Donaldson. 2015. Many-core compiler fuzzing. In *PLDI 2015*. ACM, 65–76. https://doi.org/10.1145/2737924.2737986
[97] Heting Liu, Zhichao Li, Cheng Tan, Rongqiu Yang, Guohong Cao, Zherui Liu, and Chuanxiong Guo. 2023. Predicting GPU Failures With High Precision Under Deep Learning Workloads. In *SYSTOR 2023*. ACM, 124–135. https://doi.org/10.1145/3579370.3594777
[98] Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. 2019. What bugs cause production cloud incidents?. In *HotOS 2019*. ACM, 155–162. https://doi.org/10.1145/3317550.3321438
[99] Haopeng Liu, Xu Wang, Guangpu Li, Shan Lu, Feng Ye, and Chen Tian. 2018. FCatch: Automatically Detecting Time-of-fault Bugs in Cloud Systems. In *ASPLOS 2018*. ACM, 419–431. https://doi.org/10.1145/3173162.3177161
[100] Xuanzhe Liu, Diandian Gu, Zhenpeng Chen, Jinfeng Wen, Zili Zhang, Yun Ma, Haoyu Wang, and Xin Jin. 2023. Rise of Distributed Deep Learning Training in the Big Model Era: From a Software Engineering Perspective. *ACM TOSEM* 32, 6, Article 156 (2023), 26 pages. https://doi.org/10.1145/3597204
[101] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. 2023. Prompt Injection attack against LLM-integrated Applications. *arXiv preprint arXiv:2306.05499* (2023).
[102] Siyang Lu, BingBing Rao, Xiang Wei, Byung-Chul Tak, Long Wang, and Liqiang Wang. 2017. Log-based Abnormal Task Detection and Root Cause Analysis for Spark. In *ICWS 2017*. IEEE, 389–396. https://doi.org/10.1109/ICWS.2017.135
[103] Yuteng Lu, Kaicheng Shao, Weidi Sun, and Meng Sun. 2022. MTUL: Towards Mutation Testing of Unsupervised Learning Systems. In *SETTA 2022*, Vol. 13649. Springer, 22–40. https://doi.org/10.1007/978-3-031-21213-0_2
[104] Yuteng Lu, Weidi Sun, and Meng Sun. 2022. Towards mutation testing of Reinforcement Learning systems. *J. Syst. Archit.* 131 (2022), 102701. https://doi.org/10.1016/J.SYSARC.2022.102701
[105] Yukui Luo, Shuai Li, Kuangyuan Sun, Raul Renteria, and Ken Choi. 2017. Implementation of deep learning neural network for real-time object recognition in OpenCL framework. In *ISOCC 2017*. IEEE, 298–299. https://doi.org/10.1109/ISOCC.2017.8368905
[106] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepMutation: Mutation Testing of Deep Learning Systems. In *ISSRE 2018*. IEEE, 100–111. https://doi.org/10.1109/ISSRE.2018.00021
[107] Minghua Ma, Yudong Liu, Yuang Tong, Haozhe Li, Pu Zhao, Yong Xu, Hongyu Zhang, Shilin He, Lu Wang, Yingnong Dang, Saravanakumar Rajmohan, and Qingwei Lin. 2022. An empirical investigation of missing data handling in cloud node failure prediction. In *ESEC/FSE 2022*. ACM, 1453–1464. https://doi.org/10.1145/3540250.3558946
[108] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *ICLR 2018*. OpenReview.net. https://openreview.net/forum?id=rJzIBfZAb
[109] Kiran Maharana, Surajit Mondal, and Bhushankumar Nemade. 2022. A review: Data pre-processing and data augmentation techniques. *Global Transitions Proceedings* 3, 1 (2022), 91–99.
[110] Abdulrahman Mahmoud, Neeraj Aggarwal, Alex Nobbe, Jose Rodrigo Sanchez Vicarte, Sarita V. Adve, Christopher W. Fletcher, Iuri Frosio, and Siva Kumar Sastry Hari. 2020. PyTorchFI: A Runtime Perturbation Tool for DNNs. In *DSN Workshops 2020*. IEEE, 25–31. https://doi.org/10.1109/DSN-W50199.2020.00014

[111] Tarek Makkouk, Dong Jae Kim, and Tse-Hsun Peter Chen. 2022. An Empirical Study on Performance Bugs in Deep Learning Frameworks. In *ICSME 2022*. IEEE, 35–46. https://doi.org/10.1109/ICSME55016.2022.00012
[112] Ying Mao, Vaishali Sharma, Wenjia Zheng, Long Cheng, Qiang Guan, and Ang Li. 2023. Elastic Resource Management for Deep Learning Applications in a Container Cluster. *IEEE Trans. Cloud Comput.* 11, 2 (2023), 2204–2216. https://doi.org/10.1109/TCC.2022.3194128
[113] Molin Matti and Böhme Fredrik. 2023. AI for Cybersecurity: A Study on Machine Learning and DoS Attacks AI Robustness and Bypassing Detection Methods. https://www.diva-portal.org/smash/get/diva2:1777110/FULLTEXT02. Accessed 2024.
[114] Mbsa-tud. 2024. InjectTF Github. https://github.com/mbsa-tud/InjectTF. Accessed 2024.
[115] Phillip McNelles and Lixuan Lu. 2016. Field programmable gate array reliability analysis using the dynamic flowgraph methodology. *Nuclear Engineering and Technology* 48, 5 (2016), 1192–1205. https://doi.org/10.1016/j.net.2016.03.004
[116] Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. ShortGPT: Layers in Large Language Models are More Redundant Than You Expect. *CoRR* abs/2403.03853 (2024). https://doi.org/10.48550/arXiv.2403.03853
[117] Microsoft. 2024. Microsoft Bing Chat. https://www.bing.com/chat. Accessed 2024.
[118] Mindspore-ai. 2024. MindSpore Github. https://github.com/mindspore-ai/mindspore. Accessed 2024.
[119] Aditi Mishra, Utkarsh Soni, Anjana Arunkumar, Jinbin Huang, Bum Chul Kwon, and Chris Bryan. 2023. Promptaid: Prompt exploration, perturbation, testing and iteration using visual analytics for large language models. *arXiv preprint arXiv:2304.01964* (2023).
[120] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. 2018. Scikit-Multiflow: A Multi-output Streaming Framework. *Journal of Machine Learning Research* 19, 72 (2018), 1–5. http://jmlr.org/papers/v19/18-251.html
[121] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *OSDI 2018*. USENIX Association, 561–577. https://www.usenix.org/conference/osdi18/presentation/nishihara
[122] Seyed Mahed Mousavi, Simone Alghisi, and Giuseppe Riccardi. 2024. Is Your LLM Outdated? Benchmarking LLMs & Alignment Algorithms for Time-Sensitive Knowledge. *CoRR* abs/2404.08700 (2024). https://doi.org/10.48550/ARXIV.2404.08700
[123] mpich. 2024. MPICH. https://www.mpich.org/. Accessed 2024.
[124] Ismail Muraina. 2022. Ideal dataset splitting ratios in machine learning algorithms: general concerns for data scientists and data analysts. In *7th International Mardin Artuklu Scientific Research Conference*. 496–504.
[125] Niranjhana Narayanan, Zitao Chen, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2023. Fault Injection for TensorFlow Applications. *IEEE TPDS* 20, 4 (2023), 2677–2695. https://doi.org/10.1109/TDSC.2022.3175930
[126] Gabriel L. Nazar and Luigi Carro. 2012. Fast single-FPGA fault injection platform. In *DFT 2012*. IEEE, 152–157. https://doi.org/10.1109/DFT.2012.6378216
[127] João Batista De Souza Neto, Anamaria Martins Moreira, Genoveva Vargas-Solar, and Martin A. Musicante. 2022. TRANSMUT-Spark: Transformation mutation for Apache Spark. *Softw. Test. Verification Reliab.* 32, 8 (2022), e1809. https://doi.org/10.1002/STVR.1809
[128] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H. Rogers. 2016. A large-scale study of soft-errors on GPUs in the field. In *HPCA 2016*. IEEE, 519–530. https://doi.org/10.1109/HPCA.2016.7446091
[129] Bin Nie, Ji Xue, Saurabh Gupta, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. 2017. Characterizing Temperature, Power, and Soft-Error Behaviors in Data Center Systems: Insights, Challenges, and Opportunities. In *MASCOTS 2017*. IEEE, 22–31. https://doi.org/10.1109/MASCOTS.2017.12
[130] Bin Nie, Ji Xue, Saurabh Gupta, Tirthak Patel, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. 2018. Machine Learning Models for GPU Error Prediction in a Large Scale HPC System. In *DSN 2018*. IEEE/IFIP, 95–106.
[131] Amin Nikanjam, Mohammad Mehdi Morovati, Foutse Khomh, and Houssem Ben Braiek. 2022. Faults in deep reinforcement learning programs: a taxonomy and a detection approach. *Autom. Softw. Eng.* 29, 1 (2022), 8. https://doi.org/10.1007/S10515-021-00313-X
[132] Numpy. 2024. NumPy Github. https://github.com/numpy/numpy.
[133] NVIDIA. 2024. CUDA toolkit. https://developer.nvidia.com/cuda-toolkit. Accessed 2024.
[134] NVIDIA. 2024. NVIDIA DCGM Error Injection. https://docs.nvidia.com/datacenter/dcgm/latest/user-guide/dcgm-error-injection.html.
[135] NVIDIA. 2024. NVIDIA NCCL. https://developer.nvidia.com/nccl. Accessed 2024.
[136] Daniel Alfonso Gonçalves De Oliveira, Laércio Lima Pilla, Mauricio Hanzich, Vinicius Fratin, Fernando Fernandes, Caio B. Lunardi, José María Cela, Philippe Olivier Alexandre Navaux, Luigi Carro, and Paolo Rech. 2017. Radiation-Induced Error Criticality in Modern HPC Parallel Accelerators. In *HPCA 2017*. IEEE, 577–588. https://doi.org/10.

1109/HPCA.2017.41
[137] OpenAI. 2024. OpenAI Chatgpt. https://openai.com/chatgpt. Accessed 2024.
[138] OpenAI. 2024. OpenAI Incidents. https://status.openai.com/history. Accessed 2024.
[139] George Ostrouchov, Don Maxwell, Rizwan A. Ashraf, Christian Engelmann, Mallikarjun Shankar, and James H. Rogers. 2020. GPU Lifetimes on Titan Supercomputer: Survival Analysis and Reliability. In *SC 2020*. IEEE, 1–14. https://doi.org/10.1109/SC41405.2020.00045
[140] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS 2019*. Curran Associates Inc., 8024–8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html
[141] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. 2023. Examining Zero-Shot Vulnerability Repair with Large Language Models. In *S&P 2023*. IEEE, 2339–2356. https://doi.org/10.1109/SP46215.2023.10179324
[142] PyTorch. 2024. PyTorch issue #4546. https://github.com/pytorch/pytorch/issues/4546. Accessed 2024.
[143] PyTorch. 2024. PyTorch Issue#42265. https://github.com/pytorch/pytorch/issues/42265/. Accessed 2024.
[144] Pytorchfi. 2024. Torchfi Github. https://github.com/pytorchfi/pytorchfi. Accessed 2024.
[145] Xiangyu Qi, Kaixuan Huang, Ashwinee Panda, Peter Henderson, Mengdi Wang, and Prateek Mittal. 2024. Visual Adversarial Examples Jailbreak Aligned Large Language Models. In *AAAI 2024*. AAAI Press, 21527–21536. https://doi.org/10.1609/AAAI.V38I19.30150
[146] Lili Quan, Qianyu Guo, Xiaofei Xie, Sen Chen, Xiaohong Li, and Yang Liu. 2022. Towards Understanding the Faults of JavaScript-Based Deep Learning Systems. In *ASE 2022*. ACM, 105:1–105:13. https://doi.org/10.1145/3551349.3560427
[147] Mihaela Radu. 2014. Reliability and fault tolerance analysis of FPGA platforms. In *LISAT 2014*. IEEE, 1–4. https://doi.org/10.1109/LISAT.2014.6845211
[148] Ali Rahmati, Seyed-Mohsen Moosavi-Dezfooli, Pascal Frossard, and Huaiyu Dai. 2020. GeoDA: A Geometric Framework for Black-Box Adversarial Attacks. In *CVPR 2020*. Computer Vision Foundation / IEEE, 8443–8452. https://doi.org/10.1109/CVPR42600.2020.00847
[149] Abhinav Rao, Sachin Vashistha, Atharva Naik, Somak Aditya, and Monojit Choudhury. 2023. Tricking llms into disobedience: Understanding, analyzing, and preventing jailbreaks. *arXiv preprint arXiv:2305.14965* (2023).
[150] Ray. 2020. Fix bug that test_multi_node.py::test_multi_driver_logging hangs when GCS actor management is turned on. https://github.com/ray-project/ray/pull/9539.
[151] Ray. 2020. Fix Issue #10319 Dashboard autoscaler crash. https://github.com/ray-project/ray/pull/10323.
[152] Ray. 2021. Fix memory leak in JNI. https://github.com/ray-project/ray/pull/17177.
[153] Ray. 2021. Fix unexpected error when handling the process that has exited in memory monitor. https://github.com/ray-project/ray/pull/14932.
[154] Ray. 2021. [windows] correct symlinks for files (node.py). https://github.com/ray-project/ray/pull/16817.
[155] Ray. 2022. Fix SciPy pinning. https://github.com/ray-project/ray/pull/25148.
[156] Ray. 2022. Fixed MRO for DerivedActorClass. https://github.com/ray-project/ray/pull/22113.
[157] Ray. 2022. Fixing flaky TestRemovingLeasingPlacementGroup caused by race condition. https://github.com/ray-project/ray/pull/29694.
[158] Ray. 2022. [Tune] Catch SyncerCallback failure with dead node. https://github.com/ray-project/ray/pull/29438.
[159] Ray. 2022. [Tune] [PBT] Maintain consistent TrialTrialRunner state when pausing and resuming trial. https://github.com/ray-project/ray/pull/28511.
[160] Ray-project. 2024. Ray Github. https://github.com/ray-project/ray. Accessed 2024.
[161] Michal Rozsíval and Ales Smrcka. 2023. NetLoiter: A Tool for Automated Testing of Network Applications using Fault-injection. In *DSN 2023*. IEEE, 207–210. https://doi.org/10.1109/DSN-W58399.2023.00057
[162] Behzad Salami, Osman S. Unsal, and Adrián Cristal Kestelman. 2018. On the Resilience of RTL NN Accelerators: Fault Characterization and Mitigation. In *SBAC-PAD 2018*. IEEE, 322–329. https://doi.org/10.1109/CAHPC.2018.8645906
[163] Felix Salfner, Maren Lenk, and Miroslaw Malek. 2010. A survey of online failure prediction methods. *ACM Computing Survey* 42, 3 (2010), 10:1–10:42. https://doi.org/10.1145/1670679.1670680
[164] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. Everyone Wants to Do the Model Work, Not the Data Work: Data Cascades in High-Stakes AI. In *CHI 2021*. ACM, Article 39, 15 pages. https://doi.org/10.1145/3411764.3445518
[165] Sebastian Schelter, Tammo Rukat, and Felix Biessmann. 2021. JENGA - A Framework to Study the Impact of Data Errors on the Predictions of Machine Learning Models. In *EDBT 2021*. OpenProceedings.org, 529–534. https://doi.org/10.5441/002/EDBT.2021.63
[166] Scikit-learn. 2024. Scikit-learn Github. https://github.com/scikit-learn/scikit-learn.

[167] Sadaf Shafi and Assif Assad. 2023. Exploring the Relationship Between Learning Rate, Batch Size, and Epochs in Deep Learning: An Experimental Study. In *SocProS 2022*. Springer, 201–209.
[168] Sarah Shah, Yasaman Amannejad, and Diwakar Krishnamurthy. 2021. Diaspore: Diagnosing Performance Interference in Apache Spark. *IEEE Access* 9 (2021), 103230–103243. https://doi.org/10.1109/ACCESS.2021.3098426
[169] Qingchao Shen, Haoyang Ma, Junjie Chen, Yongqiang Tian, Shing-Chi Cheung, and Xiang Chen. 2021. A comprehensive study of deep learning compiler bugs. In *ESEC/FSE 2021*. ACM, 968–980. https://doi.org/10.1145/3468264.3468591
[170] Weijun Shen, Jun Wan, and Zhenyu Chen. 2018. MuNN: Mutation Analysis of Neural Networks. In *QRS 2018*. 108–115. https://doi.org/10.1109/QRS-C.2018.00032
[171] Farhad MortezaPour Shiri, Thinagaran Perumal, Norwati Mustapha, and Raihani Mohamed. 2023. A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU. *CoRR* abs/2305.17473 (2023). https://doi.org/10.48550/ARXIV.2305.17473
[172] Jonathan Sillito and Esdras Kutomi. 2020. Failures and Fixes: A Study of Software System Incident Response. In *ICSME 2020*. IEEE, 185–195. https://doi.org/10.1109/ICSME46990.2020.00027
[173] Carl-Johann Simon-Gabriel, Noman Ahmed Sheikh, and Andreas Krause. 2021. PopSkipJump: Decision-Based Attack for Probabilistic Classifiers. In *ICML 2021*, Vol. 139. PMLR, 9712–9721. http://proceedings.mlr.press/v139/simon-gabriel21a.html
[174] Dokyung Song, Julian Lettner, Prabhu Rajasekaran, Yeoul Na, Stijn Volckaert, Per Larsen, and Michael Franz. 2019. SoK: Sanitizing for Security. In *S&P 2019*. 1275–1295. https://doi.org/10.1109/SP.2019.00010
[175] Spark. 2021. A memory leak occurs when we clone the spark session. https://issues.apache.org/jira/browse/SPARK-34087.
[176] Amir Taherin, Tirthak Patel, Giorgis Georgakoudis, Ignacio Laguna, and Devesh Tiwari. 2021. Examining Failures and Repairs on Supercomputers with Multi-GPU Compute Nodes. In *DSN 2021*. IEEE, 305–313. https://doi.org/10.1109/DSN48987.2021.00043
[177] Florian Tambon, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, and Giuliano Antoniol. 2023. Mutation Testing of Deep Reinforcement Learning Based on Real Faults. In *ICST 2023*. IEEE, 188–198. https://doi.org/10.1109/ICST57152.2023.00026
[178] Florian Tambon, Amin Nikanjam, Le An, Foutse Khomh, and Giuliano Antoniol. 2024. Silent bugs in deep learning frameworks: An empirical study of Keras and TensorFlow. *Empirical Software Engineering* 29, 1 (2024), 10. https://doi.org/10.1007/s10664-023-10389-6
[179] Gou Tan, Pengfei Chen, and Min Li. 2023. Online Data Drift Detection for Anomaly Detection Services based on Deep Learning towards Multivariate Time Series. In *QRS 2023*. IEEE, 1–11. https://doi.org/10.1109/QRS60937.2023.00011
[180] Olivier Temam. 2012. A defect-tolerant accelerator for emerging high-performance applications. In *ISCA 2012*. IEEE, 356–367. https://doi.org/10.1109/ISCA.2012.6237031
[181] Yingjie Tian and Yuqi Zhang. 2022. A comprehensive survey on regularization strategies in machine learning. *Inf. Fusion* 80 (2022), 146–166. https://doi.org/10.1016/J.INFFUS.2021.11.005
[182] Devesh Tiwari, Saurabh Gupta, George Gallarno, Jim Rogers, and Don Maxwell. 2015. Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility. In *SC 2015*. 1–12. https://doi.org/10.1145/2807591.2807666
[183] Devesh Tiwari, Saurabh Gupta, James H. Rogers, Don Maxwell, Paolo Rech, Sudharshan S. Vazhkudai, Daniel Oliveira, Dave Londo, Nathan DeBardeleben, Philippe Olivier Alexandre Navaux, Luigi Carro, and Arthur S. Bland. 2015. Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In *HPCA 2015*. IEEE, 331–342. https://doi.org/10.1109/HPCA.2015.7056044
[184] Toxiproxy. 2024. Toxiproxy Github. https://github.com/Shopify/toxiproxy.
[185] Trailofbits. 2024. A BPF-based syscall fault injector. https://github.com/trailofbits/ebpfault. Accessed 2024.
[186] Timothy Tsai, Siva Kumar Sastry Hari, Michael B. Sullivan, Oreste Villa, and Stephen W. Keckler. 2021. NVBitFI: Dynamic Fault Injection for GPUs. In *DSN 2021*. IEEE, 284–291. https://doi.org/10.1109/DSN48987.2021.00041
[187] Rua-Huan Tsaih, Hsin-Lu Chang, Chih-Chun Hsu, and David C. Yen. 2023. The AI Tech-Stack Model. *Commun. ACM* 66, 3 (2023), 69–77. https://doi.org/10.1145/3568026
[188] Muhammad Uzair and Noreen Jamil. 2020. Effects of hidden layers on the efficiency of neural networks. In *INMIC 2020*. IEEE, 1–6. https://doi.org/10.1109/INMIC50486.2020.9318195
[189] Chengcheng Wan, Shicheng Liu, Henry Hoffmann, Michael Maire, and Shan Lu. 2021. Are Machine Learning Cloud APIs Used Correctly?. In *ICSE 2021*. IEEE, 125–137. https://doi.org/10.1109/ICSE43902.2021.00024
[190] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. 2020. A comprehensive survey of loss functions in machine learning. *Annals of Data Science* (2020), 1–26. https://doi.org/10.1007/s40745-020-00253-5
[191] Yixiang Wang, Jiqiang Liu, Xiaolin Chang, Ricardo J. Rodríguez, and Jianhua Wang. 2022. DI-AA: An interpretable white-box attack for fooling deep neural networks. *Inf. Sci.* 610 (2022), 14–32. https://doi.org/10.1016/J.INS.2022.07.157

[192] Yuxiang Wei, Chunqiu Steven Xia, and Lingming Zhang. 2023. Copiloting the Copilots: Fusing Large Language Models with Completion Engines for Automated Program Repair. In *FSE 2023*. ACM, 172–184. https://doi.org/10.1145/3611643.3616271

[193] Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. 2023. Data collection and quality challenges in deep learning: a data-centric AI perspective. *VLDB J.* 32, 4 (2023), 791–813. https://doi.org/10.1007/S00778-022-00775-9

[194] Mingyuan Wu, Yicheng Ouyang, Husheng Zhou, Lingming Zhang, Cong Liu, and Yuqun Zhang. 2020. Simulee: detecting CUDA synchronization bugs via memory-access modeling. In *ICSE 2020*. ACM, 937–948. https://doi.org/10.1145/3377811.3380358

[195] Mingyuan Wu, Lingming Zhang, Cong Liu, Shin Hwei Tan, and Yuqun Zhang. 2019. Automating CUDA Synchronization via Program Transformation. In *ASE 2019*. 748–759. https://doi.org/10.1109/ASE.2019.00075

[196] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated Program Repair in the Era of Large Pre-trained Language Models. In *ICSE 2023*. IEEE/ACM, 1482–1494. https://doi.org/10.1109/ICSE48619.2023.00129

[197] Dawen Xu, Ziyang Zhu, Cheng Liu, Ying Wang, Shuang Zhao, Lei Zhang, Huaguo Liang, Huawei Li, and Kwang-Ting Cheng. 2021. Reliability evaluation and analysis of FPGA-based neural network acceleration system. *IEEE Transactions on VLSI Systems* 29, 3 (2021), 472–484. https://doi.org/10.1109/TVLSI.2020.3046075

[198] Yilin Yang, Tianxing He, Zhilong Xia, and Yang Feng. 2022. A comprehensive empirical study on bug characteristics of deep learning frameworks. *IST* 151 (2022), 107004. https://doi.org/10.1016/j.infsof.2022.107004

[199] Xin Yao, Yong Liu, and Guangming Lin. 1999. Evolutionary programming made faster. *IEEE Trans. Evol. Comput.* 3, 2 (1999), 82–102. https://doi.org/10.1109/4235.771163

[200] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In *WWW 2021*. ACM, 3087–3098. https://doi.org/10.1145/3442381.3449905

[201] Guangba Yu, Pengfei Chen, Pairui Li, Tianjun Weng, Haibing Zheng, Yuetang Deng, and Zibin Zheng. 2023. LogReducer: Identify and Reduce Log Hotspots in Kernel on the Fly. In *ICSE 2023*. IEEE, 1763–1775. https://doi.org/10.1109/ICSE48619.2023.00151

[202] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable Fine-Grained Root Causes Analysis for Microservices on Multi-modal Observability Data. In *ESEC/FSE 2023*. ACM, 553–565. https://doi.org/10.1145/3611643.3616249

[203] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *HotCloud 2010*. USENIX Association. https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets

[204] Chong Zhang, Mingyu Jin, Qinkai Yu, Chengzhi Liu, Haochen Xue, and Xiaobo Jin. 2024. Goal-guided Generative Prompt Injection Attack on Large Language Models. *CoRR* abs/2404.07234 (2024). https://doi.org/10.48550/ARXIV.2404.07234

[205] Jeff Zhang, Zahra Ghodsi, Siddharth Garg, and Kartheek Rangineni. 2020. Enabling Timing Error Resilience for Low-Power Systolic-Array Based Deep Learning Accelerators. *IEEE Des. Test* 37, 2 (2020), 93–102. https://doi.org/10.1109/MDAT.2019.2947271

[206] Jeff Jun Zhang, Kanad Basu, and Siddharth Garg. 2019. Fault-Tolerant Systolic Array Based Accelerators for Deep Neural Network Execution. *IEEE Des. Test* 36, 5 (2019), 44–53. https://doi.org/10.1109/MDAT.2019.2915656

[207] Ru Zhang, Wencong Xiao, Hongyu Zhang, Yu Liu, Haoxiang Lin, and Mao Yang. 2020. An Empirical Study on Program Failures of Deep Learning Jobs. In *ICSE 2020*. ACM, 1159–1170. https://doi.org/10.1145/3377811.3380362

[208] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *CoRR* abs/2205.01068 (2022). https://doi.org/10.48550/ARXIV.2205.01068

[209] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An Empirical Study on TensorFlow Program Bugs. In *ISSTA 2018*. ACM, 129–140. https://doi.org/10.1145/3213846.3213866

[210] Zhao Zhang, Lei Huang, Ruizhu Huang, Weijia Xu, and Daniel S. Katz. 2019. Quantifying the Impact of Memory Errors in Deep Learning. In *CLUSTER 2019*. IEEE, 1–12. https://doi.org/10.1109/CLUSTER.2019.8890989

[211] Zhixue Zhao, George Chrysostomou, Kalina Bontcheva, and Nikolaos Aletras. 2022. On the Impact of Temporal Concept Drift on Model Explanations. In *EMNLP 2022*. Association for Computational Linguistics, 4039–4054. https://doi.org/10.18653/V1/2022.FINDINGS-EMNLP.298

[212] Yang Zheng, Zhenye Feng, Zheng Hu, and Ke Pei. 2021. MindFI: A Fault Injection Tool for Reliability Assessment of MindSpore Applicacions. In *ISSREW 2021*. 235–238. https://doi.org/10.1109/ISSREW53611.2021.00068

[213] Mohamed Tarek Ibn Ziad, Sana Damani, Aamer Jaleel, Stephen W. Keckler, and Mark Stephenson. 2023. cuCatch: A Debugging Tool for Efficiently Catching Memory Safety Violations in CUDA Applications. *Proc. ACM Program. Lang.* 7, PLDI (2023), 124–147. https://doi.org/10.1145/3591225

Received 2024-05-18; accepted xxxx-xx-xx