AIOps Solutions for Incident Management: Technical Guidelines and A Comprehensive Literature Review

YOUCEF REMIL, University of Lyon, INSA Lyon, France and Infologic, France
ANES BENDIMERAD, Infologic, France
ROMAIN MATHONAT, Infologic, France
MEHDI KAYTOUE, University of Lyon, INSA Lyon, France and Infologic, France

The management of modern IT systems poses unique challenges, necessitating scalability, reliability, and efficiency in handling extensive data streams. Traditional methods, reliant on manual tasks and rule-based approaches, prove inefficient for the substantial data volumes and alerts generated by IT systems. Artificial Intelligence for Operating Systems (AIOps) has emerged as a solution, leveraging advanced analytics like machine learning and big data to enhance incident management. AIOps detects and predicts incidents, identifies root causes, and automates healing actions, improving quality and reducing operational costs. However, despite its potential, the AIOps domain is still in its early stages, decentralized across multiple sectors, and lacking standardized conventions. Research and industrial contributions are distributed without consistent frameworks for data management, target problems, implementation details, requirements, and capabilities. This study proposes an AIOps terminology and taxonomy, establishing a structured incident management procedure and providing guidelines for constructing an AIOps framework. The research also categorizes contributions based on criteria such as incident management tasks, application areas, data sources, and technical approaches. The goal is to provide a comprehensive review of technical and research aspects in AIOps for incident management, aiming to structure knowledge, identify gaps, and establish a foundation for future developments in the field.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Computing methodologies → Artificial intelligence**; • **Computer systems organization → Maintainability and Maintenance**.

1 INTRODUCTION

1.1 Context and Motivation

IT environments of today are constantly becoming larger and more complex as new technologies emerge and new work methods are adopted. They encounter challenges in ensuring efficiency and reliability. Many organizations are transitioning from product delivery to service releases, shifting from traditional static infrastructures to dynamic blends of on-premises, managed, private, and public cloud environments. Due to factors like device mobility, evolving runtime environments, frequent updates, upgrades, and online repairs, these systems are increasingly vulnerable to failures [66, 73, 196, 220]. According to Lin et al. [153], Microsoft's Azure cloud system experiences failures in approximately 0.1% of its server nodes daily. Such failures can lead to reduced system availability, financial losses, and negative user experiences [64]. Surveys conducted by the International Data Corporation (IDC) reveal that application downtime can cost businesses up to $550,000 per hour [52, 88, 108]. These significant losses trigger the need for autonomic and self-managing systems to address the root causes of failures and enhance the quality and responsiveness of IT services [46, 89, 213].

Traditional IT management solutions, relying on expert systems and rule-based engines, frequently encounter shortcomings in adaptiveness, efficiency, and scalability [136, 205]. These solutions often overlook the real-time state of a system, leading to inaccurate predictive analysis based

Authors' addresses: Youcef Remil, yre@infologic.fr, University of Lyon, INSA Lyon, CNRS UMR 5205, FR-69621, Lyon, France and Infologic, 99 avenue de Lyon, FR-26500, Bourg-Lès-Valence, France; Anes Bendimerad, abe@infologic.fr, Infologic, 99 avenue de Lyon, FR-26500, Bourg-Lès-Valence, France; Romain Mathonat, rma@infologic.fr, Infologic, 99 avenue de Lyon, FR-26500, Bourg-Lès-Valence, France; Mehdi Kaytoue, mka@infologic.fr, University of Lyon, INSA Lyon, CNRS UMR 5205, FR-69621, Lyon, France and Infologic, 99 avenue de Lyon, FR-26500, Bourg-Lès-Valence, France.

on its current condition. Additionally, they are rooted in a conventional engineering mindset that emphasizes manual execution of repetitive tasks and individual case analysis, often relying on bug reproduction steps or detailed logs [73].

These factors have sparked interest in replacing multiple conventional maintenance tools with an intelligent platform capable of learning from large volumes of data to proactively respond to incidents. Accordingly, organizations are turning to AIOps to prevent and mitigate high-impact incidents. The term AIOps was initially introduced in 2017 by Gartner to address the AI challenges in DevOps [205]. Initially, AIOps stemmed from the concept of IT Operations Analytics (ITOA). However, with the growing popularity of AI across various domains, Gartner later redefined AIOps based on public opinion, characterizing it as Artificial Intelligence for Operating systems [226]. AIOps involves the application of big data and machine learning techniques to intelligently enhance, strengthen, and automate various IT operations [73, 196, 205]. AIOps learns from a diverse range of data collected from services, infrastructures, and processes. Subsequently, it autonomously takes initiatives to detect, diagnose, and remediate incidents in real-time, leveraging this acquired knowledge [66, 73, 205].

To date, a universally accepted formal definition of AIOps is yet to emerge due to its novelty and its expansive scope bridging research and industry. While some definitions focus solely on the capabilities and benefits of AIOps, without delving into its conceptual framework and operational processes, several research efforts have taken the initiative to propose a comprehensive definition (refer to Table 1). These definitions commonly converge on two key points [196]. Firstly, AIOps entails the application of artificial intelligence to enhance, fortify, and automate a wide range of IT operating systems. Secondly, AIOps emphasizes the provision of complete visibility, control, and actionable insights into the past, present, and potentially future states of the system. Other definitions also emphasize the importance of various aspects such as robust data collection, data ingestion and effective querying capabilities, scalable infrastructure, and real-time operations. It is highly important to note that AIOps extends beyond the management of incidents and the automation of maintenance processes. We concur with the findings of [196] that AIOps includes two primary subareas: incident management and resource management, as depicted in Figure 1. While the resource management procedure covers techniques for optimal allocation and utilization of resources for IT operations, our study specifically focuses on exploring the capabilities of AIOps to assist the incident management procedure. Our objective is to redesign and categorize the entire maintenance workflow routines related to incident management. This is achieved by considering the diverse array of contributions made in AIOps and aligning them with the industrial needs through clearly defined phases.

1.2 Focus of this Review: AIOps for Incident Management

Building upon the work of [205, 226], we propose that a prototypical AIOps system comprises six fundamental abilities that give rise to various tasks within the incident management procedure.
Perception. This capability centers on the ability to gather heterogeneous data sources, including log and event data, key performance metrics, network traffic data, and more, from a multitude of sources, such as networks, infrastructure, and applications. It is essential that the ingestion process accommodates both real-time streaming and historical data analysis. Additionally, powerful data visualization, querying, and indexing mechanisms are also necessary elements.
Prevention. This process entails actively identifying potential failures and forecasting high-severity outages in the system. Preventing incidents is crucial for maintaining a healthy and robust system. Therefore, implementing an automated system that continuously monitors system health and promptly alerts administrators about potential problems is essential.

Table 1. Available AIOps definitions with corresponding capabilities.

Work	Provided Definition	Capabilities
[205]	"AIOps platforms combine big data and machine learning functionality to support all primary IT operations functions through the scalable ingestion and analysis of the ever-increasing volume, variety, and velocity of data generated by IT. The platform enables the concurrent use of multiple data sources, data collection methods, and analytical and presentation technologies"	Performance Analysis, Anomaly Detection, Event Correlation, IT Service Management, and Automation
[73]	"AIOps is about empowering software and service engineers (e.g., developers, program managers, support engineers, site reliability engineers) to efficiently and effectively build and operate online services and applications at scale with artificial intelligence (AI) and machine learning (ML) techniques"	High Service Intelligence, High Customer Satisfaction, High Engineering productivity
[216]	"AIOps is a methodology that is on the frontier of enterprise IT operations. AIOps automates various aspects of IT and utilizes the power of artificial intelligence to create self-learning programs that help revolutionize IT services"	Improving human-AI collaboration, Monitoring and Proactive IT work, Efficient time saving, Faster Mean Time To Repair
[46]	"AIOps is an emerging interdisciplinary field arising in the intersection between the research areas of machine learning, big data, streaming analytics, and the management of IT operations"	Efficient Resource Management and Scheduling, Complex failure management

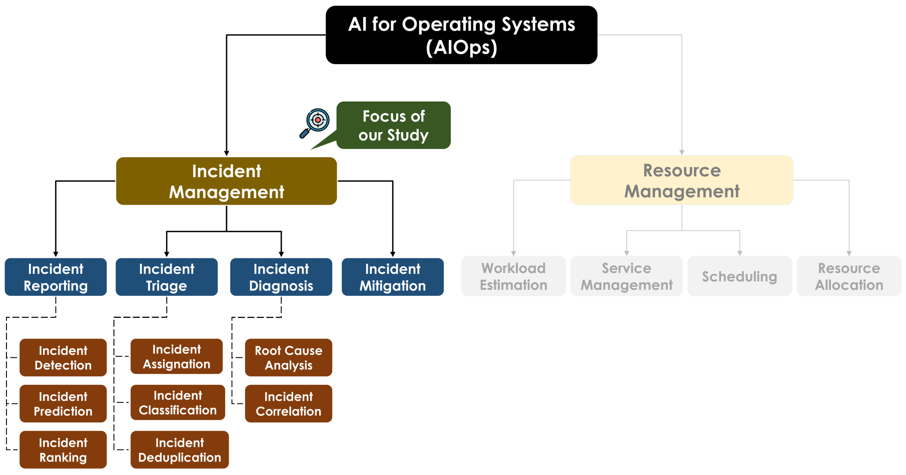


Fig. 1. Exploring the research landscape of AIOps subareas with a focus on Incident Management.

Detection. If errors occur, it is imperative for the system to detect the associated anomalies or symptoms. This is achieved by analyzing vast amounts of perceptual and historical data to identify abnormal content in either the time domain or spatial domain, or both. This process includes discovering abnormal patterns in data and detecting flexible abnormal conditions that exceed static thresholds, while minimizing noise in data, such as false alarms or redundant events.
Location. The objective of this process is to identify and analyze potential root causes and faulty actions responsible for the underlying incidents by conducting a causality and correlation study. This study must be contextualized within a unified topology to ensure its accuracy. Without the context and constraint of topology, the detected patterns, while valid, may be unhelpful and distracting. By deriving patterns from data within a topology, the number of recurrent and redundant patterns can be reduced, exceptionalities in data can be highlighted, and hidden dependencies can be identified.

Action. This includes conducting reactive triage on problems and prioritizing incidents once detected or predicted, as well as implementing a series of corrective actions based on the current scenario and past solutions that have already been provided. However, it is important to note that automatic healing actions need to be executed safely.
Interaction. It is referred to as human-computer intelligent interaction. This involves bidirectional interactive analysis between the intelligent models and the expertise of users. For instance, the system can integrate human expertise to enhance its models or similarly leverage model insights to enrich and update the user background knowledge. Furthermore, this includes facilitating communication and collaboration between different maintenance teams and with customers, promoting efficient information sharing and effective issue escalation.

Drawing of these capabilities, several companies have started dispensing AIOps tools as commodities within the last few years, while a number of technology giants have adopted an AIOps algorithmic viewpoint to maintain their on-premises or cloud computing infrastructures and manage incidents [66, 73, 136, 143, 145, 152, 206], thereby inducing the academic field to evolve and deliver more ingenious and innovative solutions. In actuality, the notion of utilizing AI to refine IT and maintenance operations, despite its recent emergence as a research field, is not entirely novel [46, 196]. Beginning in the mid-1990s, some research work explored software defects in source code by employing statistical models founded on source code metrics [49, 67, 124]. Since the start of the new decade, various techniques have been proposed to tackle online software [204, 287, 293] and hardware [138, 272, 298] failure prediction and anomaly detection [68, 189, 256]. Multiple other domains of AIOps, such as event correlation [154, 163, 263], bug triage [60, 266, 277, 278], and root cause analysis [115, 126, 139, 159], have also witnessed significant contributions over the last two decades. In fact, the reliability and maintainability of hardware and software systems have always been a prominent research focus. However, we have recently witnessed an increased interest in this field. This phenomenon is driven by two main factors: firstly, the remarkable advances achieved in the field of artificial intelligence, and secondly, the shift of numerous IT organizations from product delivery to service release, coupled with the transition from traditional to dynamic infrastructures.

Despite the promising benefits that AIOps offers, it remains federated and unstructured as a research and practical topic [46, 196, 216]. It involves a diverse array of contributions stemming from various specialized disciplines involving both industry and academia. Given its novelty and cross-disciplinary nature, AIOps contributions are widely dispersed, lacking standardized taxonomic conventions for data management, targeted areas, technical implementation details, and requirements. As such, discovering and comparing these contributions has proven to be challenging [196]. The lack of a unified terminology results in the absence of guidelines and a clear roadmap for addressing the gaps in the state-of-the-art within AIOps. In fact, while various data-driven approaches may be attributed to the AIOps research area, findings from disparate domains, such as machine learning, may not necessarily apply to software analytics domains like AIOps [165]. Therefore, it is important to determine within the purview of AIOps, the optimal taxonomy that must be driven by an industrial need necessitating domain expertise in both IT operations and AI. It is also highly important to outline the requirements (desiderata) to construct effective AIOps models, including interpretability, scalability and robustness among others. Additionally, one must also inquire about the metrics that should be employed to compare AIOps methods that belong to the same category, such as anomaly detection or root cause analysis. Metrics based on machine learning, such as contingency metrics, do not always reflect the real accuracy of models when deployed in actual scenarios and hence require contextual and/or temporal adaptation. Multiple other factors and particularities should be taken into account (e.g., human involvement in the loop.).

1.3 Outline and Contributions

Our work focuses on providing a holistic framework to the knowledge base of AIOps including technical and research facets. This framework is explicitly designed to address the challenge of adeptly managing incidents within IT environments. In pursuit of this objective, our contributions can be outlined as follows:

- ❏ We have established a unified and agnostic terminology to define the most relevant terms and key concepts that have been variably adopted in the existing research work within the field of incident management (e.g., hardware/software failure management, anomaly detection, bug triage, fault localization, etc.). Furthermore, we have expounded upon various dimensions of this process, including maintenance protocols levels.
- ❏ This effort has led us to reveal existing challenges and pain points and define the fundamental building blocks required to achieve a systematic AIOps approach for an intelligent and effective incident management procedure. This includes providing technical specifications for data management such as data collection, storage, and visualization, establishing clearly defined incident management tasks, and emphasizing crucial requirements to be considered when adopting this approach.
- ❏ Following this, based on the provided terminology, we establish a comprehensive taxonomy to categorize the notable research contributions from prominent conferences and journals in machine learning, data mining, and software engineering. This taxonomy is proposed considering clearly delineated data sources, specific research areas, properties of models and evaluation metrics.
- ❏ The proposed taxonomy also sets itself apart from previous work by closely aligning with the distinct requirements of both industry and research. It facilitates the discovery, implementation, and comparison of various methods and tools, while also addressing existing gaps in the field and hence highlighting potential areas for improvement.
- ❏ We also provide publicly available datasets across various incident tasks and application areas. This represents a valuable contribution, given that identifying the most appropriate datasets for a specific task can be a non-trivial endeavor. Moreover, it facilitates the replication and comparison of techniques within the same research area. It is noteworthy that none of the existing surveys provide such extensive coverage of datasets.

Roadmap. The paper is structured as follows. In Section 2, we begin by providing an overview of the incident management procedure, which includes clear definitions, terminology, existing protocols, and targeted maintenance layers. Next, in Section 3, we delve into the usage of AIOps to standardize the incident management process, emphasizing important considerations in designing and implementing AIOps solutions. Moving on, Section 4 introduces the taxonomy, outlining its components such as data sources and evaluation metrics. Following that, in Section 5, we review the most relevant works in the field of incident management based on the proposed taxonomy. Additionally, Section 6 presents publicly available AIOps datasets and benchmarks used for evaluating AIOps approaches. Finally, the paper concludes with a discussion that includes concluding remarks, open challenges, and areas for improvement in AIOps research.

2 STREAMLINING INCIDENT MANAGEMENT PROCEDURE

2.1 Terminology and Definitions

In the following, we aim to provide an easy-to-understand explanation of various terms that are commonly and interchangeably used in incident management and AIOps. Taking inspiration from

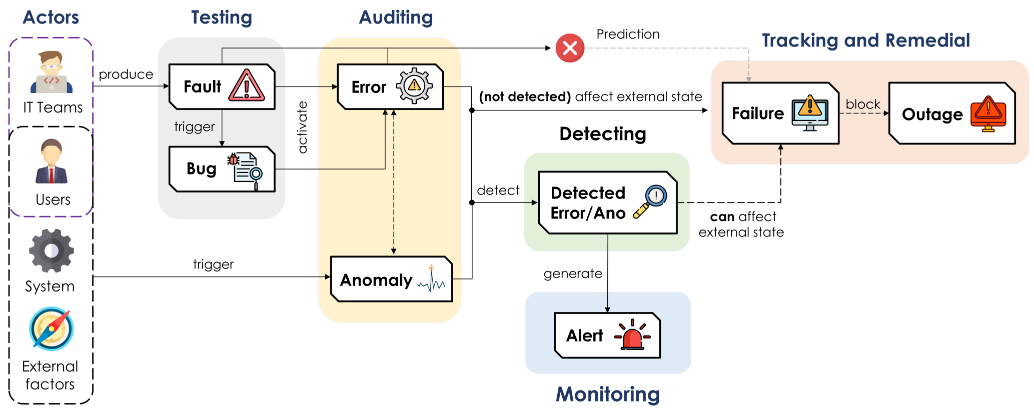


Fig. 2. Comprehensive chronological schema highlighting the distinctions and key connections among Faults, Bugs, Errors, Anomalies, Failures, and Outages.

the work of Salfner et al. [220], we build upon their terminology by offering a formal definition that helps clarify the meaning and relevance of these terms in the field.

Various terms related to incidents, such as fault, error, bug, failure, outage, and anomaly, have been widely used in the field, often without a thorough examination of their precise meanings. For instance, the most commonly used term in the literature is *failure* to indicate system downtime, hardware and software crashes, service disruptions, network traffic disturbances, etc. [138, 204, 287, 298]. On the other hand, some other methods utilize the term *outage* to refer to the most severe cases that can significantly reduce system availability and impact user experience [64, 293]. However, there are also studies that focus on *anomaly* detection, which deals with identifying abnormal behaviors that are already present in the data, often in system metrics [68, 189, 256]. Regarding the analysis of root causes, the majority of research falls under the category of *fault* localization, which generally identifies faulty components in the source code [22, 215, 264]. It may also extend to faulty actions [148], but it is agreed that this can be the initial point that ultimately leads to failures. The distinctions between these terms based on faulty behavior, underlying causes, and resulting consequences have not received sufficient attention. Although some attempts have been proposed to categorize and differentiate these terms, such as the work of [34, 220], they do not provide a comprehensive framework. In order to establish a precise terminology that clarifies the meaning of each interrelated term, we propose a coherent lexicon that covers a broader range of concepts compared to the framework proposed by [220]. Moreover, we present a chronological schema in Figure 2 illustrating the key relationships among these terms. We also identify the actors involved in initiating faulty behaviors within a system. Subsequently, we provide formal definitions for each term from our perspective.

Failures. A failure refers to an event that occurs when a system, a component, or a service is unable to fulfill its intended primary function and deviates from it. Failures are often evident and can be observed by either the user or the system, typically stemming from errors or anomalies that give rise to observable issues or malfunctions. Note that within systems, various issues may arise, but unless they result in an undesired output, they do not qualify as failures. Users typically report failures, which can prompt the need for troubleshooting or repairs to prevent outages.

Outages. An outage refers to a period in which a system, a service, or a network becomes entirely inaccessible or unavailable to users. Outages can stem from failures such as hardware malfunctions or software glitches indicating complete interruptions and unavailability of the required service. These situations often necessitate immediate palliative measures to restore normal operations before delving into the underlying problem for a curative maintenance.
Errors. Errors signify instances where the system deviates from its correct and normal state, indicating the presence of an actual issue. These errors may not always be immediately apparent to the user and can remain implicit until they manifest as failures, especially if they are not accurately detected at the opportune moment. Alternatively, errors can be identified through the use of specialized tools or specific algorithms designed for detection purposes.
Anomalies. Anomalies are defined as unexpected or abnormal behaviors of patterns that deviate from the expected state. They represent irregularities or unusual occurrences that may or may not indicate an error. Unlike errors, anomalies can serve as early indications of underlying issues, but they can also be harmless or temporary deviations that do not directly result in failures. Various factors, such as unusual data patterns or external influences like cyber attacks, can contribute to the emergence of anomalies. These anomalies can be identified and detected through the monitoring and analysis of system metrics.
Faults and Bugs. A fault pertains to an abnormality or defect discovered in a hardware or software component that exhibits incorrect behavior, potentially leading to errors and failures if not promptly detected. These faults generally arise from inherent problems or weaknesses within the system's components. They can be caused by various factors, including human interventions by end-users or administrators, design flaws, system settings, or improper handling. In software development, faults manifest as bugs, which stem from coding mistakes. Identifying faults that lead to bugs often takes place during the testing phase. Conversely, in the case of hardware or setup issues, faults directly result in errors during system operation.
Alerts. In addition to leading to failures, both undetected and detected errors and anomalies can cause system to deviate from normal behavior as a side effect. This condition is commonly referred to as symptoms [97]. These symptoms typically manifest as alerting reports, indicating a specific event or condition that demands attention or action. Alerts are usually triggered based on predefined rules or thresholds associated with the symptoms, particularly in the case of anomalies.

Figure 2 illustrates the progression of a fault or anomaly, stemming from internal or external factors, towards a failure and potentially an outage. To illustrate this, let's consider a scenario of a fault-tolerant system with a memory leak problem. The fault in this system is a missing `free` statement in the source code, which prevents the proper deallocation of memory. As long as the specific part of the software responsible for memory deallocation is never executed, the fault remains dormant and does not affect the system's operation. However, when the piece of code that should free memory is executed, the software enters an incorrect state, turning into an error. In this case, memory is consumed but never freed, even though it is no longer needed. Initially, if the amount of unnecessarily allocated memory is small, the system may still deliver its intended service without any observable failures from the outside. As the code with the memory leak is executed repeatedly, the amount of free memory gradually decreases over time. This out-of-norm behavior, where the system's parameter `free-memory` deviates from the expected state, can be considered as a symptom of the error. It serves as an indication that something is amiss within the system. At some point, when the memory leak has consumed a significant amount of available memory, there may not be enough resources left for certain memory allocations. This leads to the detection of the error, as the system encounters a situation where it cannot allocate memory when required. In a fault-tolerant system, even if a failed memory allocation occurs, it does not necessarily result in a service failure. The system may employ mechanisms such as spare units to complete the operation and maintain service delivery. Therefore, a single failed memory allocation, by itself, may not cause a service failure or outage. However, if the entire system becomes incapable of delivering its service correctly due to a series of errors or significant resource depletion, a failure occurs. This failure indicates that the system is no longer able to fulfill its intended function, impacting its users and potentially leading to an outage. During an outage, the system becomes completely unavailable, and its services cannot be accessed or utilized by users. In the context of the given example, an outage could happen if the memory leak issue is not addressed in a timely manner, leading to severe resource exhaustion that renders the system inoperable. In summary, the presence of a fault and subsequent error can be indicated by symptoms like memory consumption or depletion. Anomaly detection and monitoring can help identify deviations from expected system behavior. Alerts can be generated to notify system administrators or developers about these anomalies, allowing them to take corrective actions. If the errors and issues persist and prevent the system from delivering its services correctly, a failure occurs, potentially resulting in an outage.

Aiming to provide unified terminology and avoid confusion, in the following, we will refer to all these terms as *incidents*. This term comprises a broader scope and universally applies to any unplanned event or occurrence that disrupts the normal state, behavior, or output of a system, a network, or a service.

2.2 Existing Maintenance Protocols in Incident Management

The incident management process should adhere to standardized maintenance protocols and strategies that are universally accepted by IT organizations. These protocols dictate how incidents should be handled based on their occurrence time and the available physical and human resources. They also measure the impact of incidents based on key factors such as availability, performance, and quality. These protocols serve as a framework for assessing the impact of various incident management tasks depicted in Figure 1. Unlike some previous works that classified different incident management methods based on these protocols [75, 90], we present them as abstractions of how incident management tasks help achieve optimal and effective strategies. We do not consider these protocols as standalone phases in the incident management process since our taxonomy primarily focuses on using data-driven approaches to handle reported, detected, or predicted incidents in a phased manner, from reporting to mitigation, regardless of the chosen protocol. The different maintenance strategies can be categorized into two main approaches, Reactive and Proactive maintenance

Reactive maintenance is performed in response to incidents that are detected or reported by end users or internal maintenance staff (see detecting and tracking zones in Figure 2). On the other hand, **Proactive** maintenance aims to prevent potential problems from occurring and intervenes proactively to rectify them, typically through auditing and testing (as illustrated in Figure 2). Figure 3 provides a visual representation of the distinct patterns of maintenance strategies analyzed in our study. In reactive maintenance, there is often a time constraint, leading to palliative measures that partially or completely resolve the issue, allowing the affected activity to proceed (especially in the case of outages). For instance, a temporary technical configuration can be implemented. However, curative maintenance must be conducted to implement a stable solution and prevent the issue from recurring for the same or other customers (in the case of failures and/or outages). Proactive maintenance, on the other hand, uses a range of measures, including scheduled maintenance routines and conditional maintenance protocols. These protocols assess system functionality, identify anomalies and errors, and mitigate potential performance degradation and malfunctions (see detecting, auditing, and monitoring zones). This approach heavily relies on AIOps and leverages advanced machine learning algorithms and big data mining to forecast potential system malfunctions by analyzing historical data patterns. Prescriptive maintenance surpasses predictive maintenance by

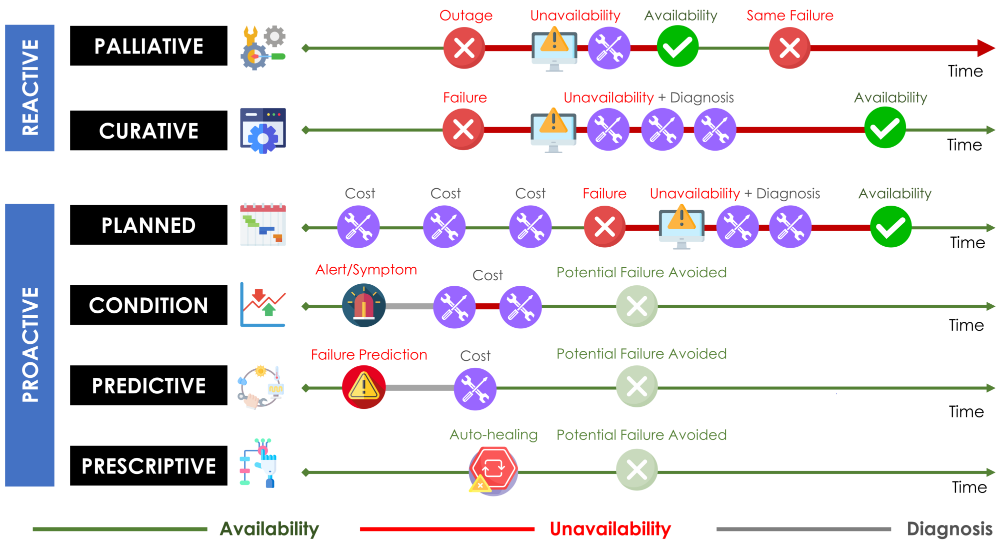


Fig. 3. Behavioral scheme of the different maintenance protocols. Adapted and improved from [90].

taking a proactive approach to intelligently schedule and plan asset maintenance. Unlike predictive maintenance, which relies solely on historical data, prescriptive maintenance incorporates current equipment conditions to provide precise instructions for repairs or replacements. Moreover, prescriptive maintenance has the capability to recommend optimal palliative or curative actions.

2.3 Target Maintenance Strata

The approach to both proactive maintenance and reactive diagnosis involves a comprehensive examination of its components across multiple layers.

- ❏ **Technical or Physical layer.** This layer focuses on the machines and their various components. For instance, the technical layer would involve monitoring the application server machine, the database, and elements such as RAM, SWAP, processor and disk usage, network identities, and connectors. It also involves monitoring and analyzing the hardware and infrastructure aspects of the system (e.g., remaining useful lifetime of physical components). Examples of checks within this layer include analyzing CPU utilization to ensure it stays within acceptable limits, monitoring disk space to prevent potential storage issues, and checking network connectivity to ensure smooth communication between components.
- ❏ **Application Layer.** This layer centers around the key components of the software application. It includes the application server, the database server, and user workstations. For example, the application layer examination would involve analyzing factors such as heap utilization, cache code performance, resource consumption, launch configuration controls, and dump files. This can help to identify incident such as memory leaks, inefficient cache utilization, or misconfigured launch settings that may impact the application performance.
- ❏ **Functional Layer.** This layer focuses on evaluating the processes executed by the software application to ensure efficient smooth data exchanges, optimal latency, accurate statistics

execution, and more. For instance, the maintenance team might analyze the response time of data retrieval or the accuracy of statistical calculations performed by the application.

❑ **Business Layer.** This layer assesses the control of critical business parameters. It involve evaluating key metrics and business-related aspects of the system to ensure that it aligns with the overall goals and objectives such as the number of transactions processed, the success rate of data transfers, or the adherence to predefined service-level agreements.

3 TOWARDS AN AUTOMATED AIOPS SOLUTION FOR INCIDENT MANAGEMENT

The application of AIOps to revolutionize incident management is a complex procedure, requiring a shift from traditional and conventional practices to a fully automated process. Designing, implementing, and deploying machine learning models in real-world software systems scenarios is not a simple task and requires careful consideration. To unlock the full potential of an integrally intelligent solution for incident management and transfer knowledge from AI models to the operating systems domain, a thorough evaluation of the current landscape is necessary. This evaluation should anticipate potential roadblocks stemming from human, resource, or knowledge factors that may arise during implementation. It is also essential to identify areas for improvement and determine the cost associated with transitioning from old practices to a new methodology while considering feedback, reviews, and the trust of practitioners. In the following, we will outline a comprehensive list of pain points and challenges that need to be addressed before building AIOps solutions to manage incidents.

3.1 Pain points and Challenges

Within the software industry, there is ongoing development and adoption of AIOps solutions, but it is still in its early stages [73]. Building and implementing AIOps solutions in real-world scenarios continues to pose challenges from both technical and non-technical standpoints [46, 65, 66, 161, 208, 216]. To elaborate further, we outline the significant challenges involved in constructing AIOps solutions.

Novelty of AIOps. AIOps remains a relatively new and unstructured area of research that lacks a clear and comprehensive definition [73, 196]. It involves various research domains such as system and pattern design, software engineering, big data, machine learning, distributed computing, and information visualization, among others. Due to its novelty and interdisciplinary nature, AIOps contributions and methods are widely scattered, lacking standardized conventions for data management, target areas, implementation details, and requirements. Consequently, selecting the most suitable techniques from these diverse contributions to achieve specific objectives has proven to be challenging. Hence, it is essential to identify the appropriate methods and tools based on defined goals, and establish an objective procedure to compare between them. Additionally, these methods should align with the IT organization's policies, such as interpretability and trustability, while also considering the available resources and scalability issues. According to Dang et al. [73], the domain of AIOps presents several unique challenges that require a comprehensive understanding of the overall problem space, including considerations of business value, data, models, as well as system and process integration.

Data Management. Efficient data management is the most critical component for the successful implementation of an AIOps framework. To achieve high observability and real-time analysis, it is essential to integrate and process large volumes of data from diverse sources. However, this integration presents several challenges that need to be addressed. The foremost challenge is to ensure that real-time or near-real-time data collection does not adversely affect the performance of the monitored applications. It is crucial to prevent overwhelming the memory or network resources of these applications. Therefore, the data management system employed must exhibit optimal performance in terms of data storage, ingestion, compression, and querying capabilities across different data types.

Data Normalization. AIOps models have specific data requirements that differ from those typically used in general machine learning models. Despite the vast amount of telemetry data collected by major cloud services, which can reach terabytes or even petabytes on a daily or monthly basis, the quality and quantity of available data still fall short of meeting the needs of AIOps solutions. Studies by Dang et al. [73], Levin et al. [136] and Chen et al. [65] highlight the challenges of working with diverse data from various sources, which often come in disparate formats and structures, making normalization and cleaning complex. This data can be unstructured or semi-structured, including logs, execution traces, source code, hierarchical and graph data, and network traffic, requiring specific preprocessing techniques. Moreover, AIOps models that heavily rely on supervised machine learning algorithms require labeled data for training. However, data often contains noise or missing values, and obtaining labeled data can be challenging. This makes it difficult to build accurate and robust AIOps models. In many AIOps scenarios, constructing supervised machine learning models poses challenges due to data quality issues, such as the absence of clear ground truth labels, the need for manual efforts to obtain high-quality labels, imbalanced datasets, and high levels of noise. Overcoming data quality limitations and managing noise and imbalanced datasets are key areas to focus on when building AIOps models.

Human Interaction with AIOps. One of the main challenges in this context is the difficulty of shifting the mindset of IT practitioners towards abandoning old maintenance routines and adopting entirely new approaches [73]. AIOps-oriented engineering is still at a very early stage, and the establishment of recognized best practices and design patterns for AIOps in the industry is far from complete. Experienced practitioners struggle to let go of their manual activities that are based on adaptation and auditing tasks. Meanwhile, the fundamental methodology of AIOps solutions revolves around learning from historical data to predict the future. The traditional engineering mindset on the other hand, which involves investigating individual cases and reproducing incident steps based on log data, is inefficient or even impractical in large-scale service scenarios.

In fact, there are two distinct perspectives and opinions among practitioners when it comes to AIOps. On one hand, there is a belief that AI can address all challenges, but this expectation is not grounded in reality [73]. On the other hand, some express doubts about the efficiency of machine learning models in the industry [216]. This skepticism stems from the fact that AIOps solutions primarily rely on learning from past experiences to predict future trends using large datasets. However, experienced IT professionals question the effectiveness of these models, even after recognizing the need for digital transformation. They may argue that past experiences do not always provide sufficient coverage to accurately forecast system states, necessitating always human interventions and checking. They might also cite instances where similar past experiences yielded different outcomes. Consequently, businesses require additional time to build confidence in the reliability and dependability of AIOps recommendations. Therefore, it is crucial to invest significant effort in both directions to simplify this transition. Ensuring trust, involving humans in the decision-making process, and providing interpretability and explainability of AIOps solutions are essential to instill confidence in these approaches [165]. Additionally, setting realistic expectations and clearly defining the scope and goals of AIOps are also crucial considerations.

Implementation and Integration of AI models. Building machine learning models for AIOps applications presents unique challenges that are not commonly encountered in other ML/AI scenarios [73]. For instance, natural language processing models, often used in machine learning, tend to generate inaccurate results when applied to software engineering-related data according to Menzies [174] and Ray et al. [207]. On the other hand, constructing a supervised machine learning model faces challenges related to data quality and availability, as mentioned earlier. These challenges include imbalanced datasets and the absence of clear ground truth labels. To address these issues, unsupervised or semi-supervised machine learning models can be explored. However, acquiring sufficient labels to understand the "what is abnormal" pattern proves difficult because of the dynamic nature of system behavior and shifting customer requirements and infrastructure. Furthermore, developing high-quality unsupervised models is difficult due to the intricate dependencies and relationships among components and services [46, 61, 161]. Lou et al. [161] argue that the diagnosis of service incidents cannot solely rely on learning models, but rather requires substantial knowledge about the service system. However, in practice, this type of knowledge is often poorly organized or inadequately documented. Moreover, the need for frequent model updates and online learning poses challenges to DevOps/MLOps practices, when it comes to complex feature engineering efforts.

Another challenge is to ensure that the behavior of the model during the training phase is consistent with its performance in the testing phase. Traditional metrics used to assess models are susceptible to the contamination zone phenomenon [91], which may lead to erroneous assessments. Indeed, Fourure et al. [91], highlight that by parameterizing the proportion of data between training and testing sets, the F1-score of anomaly detection models can be artificially increased.

Finally, the effectiveness of machine learning models is often proportional to their complexity. Highly accurate models, known as black box models, lack transparency and fail to provide explanations of their decision-making process [98, 182]. This lack of transparency significantly hampers their adoption by industry practitioners who require a clear understanding of maintenance processes and tool behavior. While leveraging robust models for cost optimization and task automation is valuable, it comes at the expense of transparency. Recent studies in the field of AIOps suggest that interpretable models, even with slightly lower performance, are preferred over high-performing yet non-interpretable models [165]. Thus, successfully automating incident management processes necessitates establishing practitioners' trust by providing explanations for model decisions, aligning with the concept of eXplainable Artificial Intelligence (XAI) [98].

3.2 AIOps Framework for Data and Incident Management Procedure

Implementing intelligent solutions for incident management procedures is a complex task that does not solely rely on implementing and training machine learning models on extensive data sets to uncover actionable patterns. One of the significant challenges in adopting AIOps is transitioning from an existing architectural platform that relies on separate modules and executing scripts [73, 208, 226]. This transition involves the adoption and integration of various digital technologies to fundamentally transform the delivery of maintenance and incident management services to end customers or internal maintenance staff. To accomplish this transformation, a reliable, scalable, and secure infrastructure architecture needs to be designed. Such an architecture should be capable of effectively collecting, ingesting, and managing large volumes of data from diverse sources with varying formats. Then, it leverages artificial intelligence approaches to extract meaningful insights from this data, following an iterative and standardized procedure that will be discussed in Section 3.3. Furthermore, this architecture needs to facilitate user interaction, catering to data scientists and engineers, MLOps engineers, and end users. The purpose is to aid in incident management by providing corrective or preemptive actions, predictive alerts, and other valuable insights. Hence, successful implementation of intelligent incident management solutions requires addressing numerous factors, including architectural transformations, digital technology integration, robust data management, and user-friendly interfaces, to fully leverage the power of AIOps in managing incidents, enhancing operational efficiency, and driving better business outcomes.

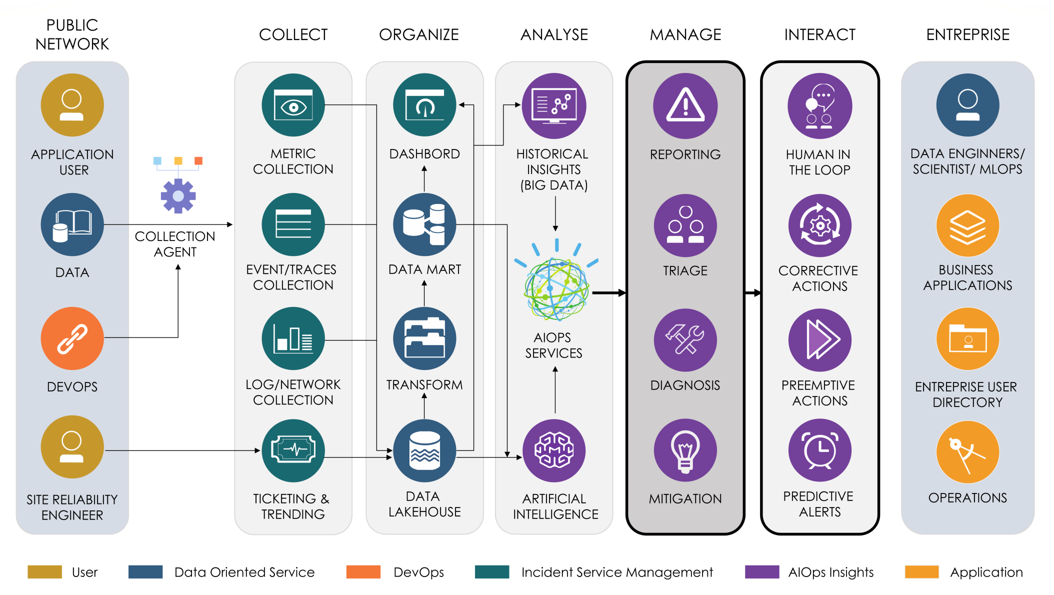


Fig. 4. Comprehensive AIOps reference architecture for Incident Management Procedure [13, 205]

Figure 4 presents an overview of a layered AIOps reference architecture designed to support intelligent incident management procedures and seamlessly integrate key tools and modules. This architecture is suitable for both traditional in-house and cloud-based application environments, providing a flexible framework for implementing AIOps solutions. In the following, we provide a brief overview of the essential modules that are part of this framework, while also covering potential available and existing open-source tools that can be used to perform the underlying tasks.

Data Collection and Ingestion. Dynamic infrastructures generate vast amounts of operational and performance data including infrastructure health, application performance, activity logs, event notifications, network traffic flows, user interactions, and more. Understanding the diverse formats, protocols, and interfaces employed by various data sources is crucial in this context. Extracting monitoring data typically involves installing and configuring a collection agent on the monitored servers. Numerous components within the infrastructures can be effectively monitored, including hardware components, application servers, associated databases, operating systems, virtual machines, and business trends. These components align with the maintenance layers previously discussed in Section 2.3.

Numerous open-source collection agents have been introduced to facilitate monitoring tasks. For example, InfluxData offers Telegraf [20], a versatile tool capable of connecting with over 300 popular applications. The ELK stack provides the Beats suite [6], which enables the collection and transmission of metrics to ElasticSearch. Another notable agent is Fluentd [10], offering the capability to collect data events and send them to various destinations such as files or DBMS. These open-source tools come with several advantages, including their popularity and ease of deployment and configuration. They have gained significant traction in the industry as they are widely accessible and customizable. However, in some cases, it may be beneficial to develop custom collection agents. This allows organizations to collect specific use-case data, leverage pre-built data manipulation libraries, and utilize existing BI tools within their infrastructure [43]. Furthermore, custom agents provide the flexibility to tailor the configuration according to specific requirements.

To handle data from different sources with varying formats, it is essential to implement a data ingester module. This module ensures efficient and reliable capture and aggregation of received data into a unified format that can be easily understood by the system. The process involves normalizing data structures, resolving data inconsistencies, and ensuring seamless interoperability. In some cases, appropriate transformations are performed to ensure consistency and compatibility across different data sources before persisting them. This may include data parsing, normalization, filtering, and enrichment. Data ingestion can be achieved through various tools and techniques, including ETL (extract, transform, load) processes, API integrations, or specialized data ingestion platforms. A range of tools and technologies are available to assist with data ingestion in AIOps systems. Open-source platforms like Kafka [2], Apache NiFi [3], RabbitMQ [18], or RocketMQ [19] offer capabilities for data streaming, message queuing, data routing, and reliable data ingestion from diverse sources. Additionally, there are proprietary/managed service tools such as AWS Kinesis [1], Azure Event Hubs [5], and Google Cloud Pub/Sub [11], among others, which provide similar functionalities for data ingestion.

Data Storage and Organization. Data collected in the context of AIOps can be categorized into structured, semi-structured, and unstructured data. This encompasses a range of data types, such as time series data that represent system metrics or performance indicators, event logs, sequential templates, network traffic that can be visualized as graphs, and incident tickets reported by end-users in text form. Due to the diversity in data characteristics, it becomes challenging to store all this data in a single warehouse using a "one size fits all" approach. To address this challenge, organizations need to understand and organize the data before feeding it into AI models. Depending on specific requirements and use cases, different approaches to store and query the data are available. Historically, there were two primary options for data persistence: Database Management Systems (DBMS) for structured data and data lakes for unprocessed data. DBMS solutions, like relational databases, are designed for structured data with predefined schemes. They offer predefined structures, enforce data integrity through tables, columns, and relationships, and provide ACID properties (Atomicity, Consistency, Isolation, Durability), ensuring transactional consistency. On the other hand, data lakes serve as storage repositories for vast amounts of raw and unprocessed data in its native format, accommodating semi-structured and unstructured data. The choice between data lakes and DBMS depends on factors such as data volume, variety, velocity, query requirements, data governance needs, and the organization's specific use cases and goals.

In the context of AIOps as shown in Figure 4, to support business intelligence, machine learning, and big data mining techniques, it is often beneficial to maintain both data structures simultaneously and link the systems together. Adopting a hybrid approach that combines a data lake and a data warehouse is preferable, which is referred to as a data lakehouse architecture. This approach allows organizations to leverage the strengths of each method for different aspects of their data management requirements. Figure 5 illustrates the data lakehouse architecture and highlights its key components and workflow. A data lakehouse is a centralized, powerful, and flexible big-data storage architecture that integrates various data types from multiple sources, assuming different formats. It combines the best features of data warehouses and data lakes. Initially, data lakehouses start as data lakes containing raw data in its original format, with no constraints on account size or file. This raw data is later processed, transformed, and structured as needed for analysis and querying, creating a data mart repository. To achieve this, data lakehouses leverage technologies like Delta Lake (built on Apache Parquet and Apache Avro) [7] to provide capabilities such as schema evolution, ACID transactions, and efficient data indexing. Additionally, they maintain a data catalog and metadata repository, which holds information about available datasets, their schemes, and other relevant details, ensuring data governance and assisting users in discovering and understanding

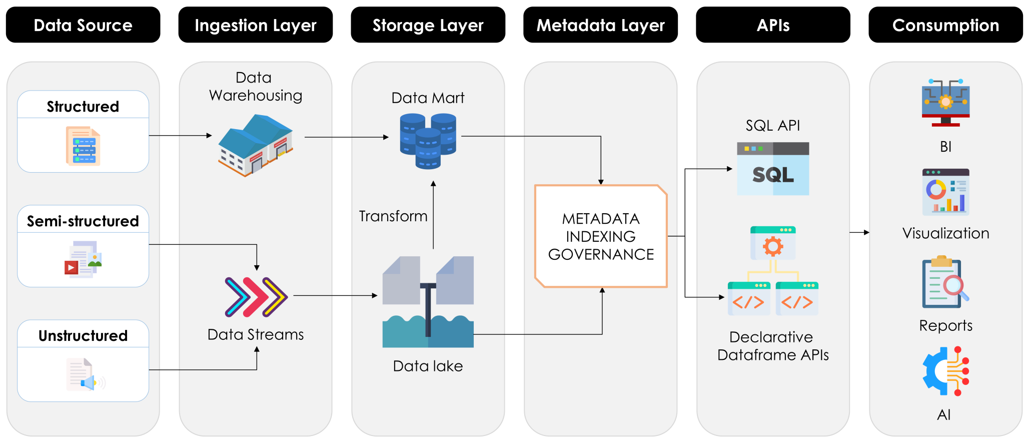


Fig. 5. Data lakehouse reference architecture

the data. Data-oriented personnel, along with other users, can access the data through SQL or other querying languages. The data lakehouse enables fast and efficient data retrieval and analysis by optimizing data for querying while preserving the raw data for exploration.

Various tools have been proposed to efficiently store and organize data, catering to different needs and use cases. For instance, Elasticsearch [9] stands out as a distributed and open-source search full-text search engine. With an HTTP web interface and schema-free JSON documents, Elasticsearch offers powerful search and indexing capabilities. It excels in managing log and event data in AIOps systems. Similarly, InfluxDB [14] serves as a time series database designed to handle high write and query loads of time-stamped data. A popular choice for storing and analyzing metrics and sensor data in AIOps systems. These databases come equipped with their own domain-specific languages (DSL). InfluxDB employs the language flux, while Elasticsearch uses ESQL. On the other hand, Clickhouse [8] has recently emerged as an open-source columnar database management system. It stands apart with its specific optimization for online analytical processing (OLAP) workloads, making it highly suitable for real-time analytics and reporting. Clickhouse distinguishes itself with its high performance, scalability, and low-latency query capabilities. Furthermore, its SQL support which is familiar to many data analysts and engineers, making it easier to integrate with existing data warehouse repositories like PostgreSQL and Oracle. In a recent study [43], the authors provided specific criteria to choose the most suitable solution for a given context. These criteria include license, notoriety and popularity, adaptability, and performance.

Data Visualization and Monitoring. In data management, whether data is organized in a data mart repository or kept within a data lake, the potential for deriving valuable insights is required through data visualization dashboards. These tools play a pivotal role in facilitating a comprehensive understanding of data, enabling in-depth inspections, and supporting data-driven decision-making processes. These visualization tools lies in their ability to offer a user-friendly interface, coupled with a range of visualization panels including versatile options like time series graphs, gauges, dynamic topologies, and more, catering to the diverse needs of IT professionals engaged in monitoring activities. By utilizing such tools, we gain the ability to continuously monitor the infrastructure, diagnose potential problems, adeptly identify intricate relationships between various components, navigate to the root causes of incidents, leading to quicker resolution times. Moreover, the tools excel in anomaly detection by visually representing abnormal behavior, thus allowing for the early identification of potential issues and threats. Among the plethora of available data visualization tools, some noteworthy open-source options include Grafana [12], Kibana [15], Metabase [17] and Apache Superset [4].

Intelligent Incident Management Procedure. AIOps insights are founded on intelligent algorithms that operate on both historical and real-time data, aiming to provide actionable patterns and prescriptive recommendations. These insights are designed to optimize operational performance, enhance situational awareness, and improve incident management procedures. More specifically, the incident management procedure can be viewed as a systematic approach that encompasses a predefined sequence of steps and processes to be consistently followed, whether incidents occur unexpectedly or are foreseen. Its primary objective is to ensure a consistent and well-coordinated response, regardless of the source of the incident report. This procedure involves the use of robust artificial intelligence algorithms to aid in reporting, classification, prioritization, assignment, diagnosis, and resolution of incidents. Throughout the entire process, it is essential to establish clear communication channels to keep stakeholders informed about the incident's status, progress in resolution, and any pertinent actions taken. Time sensitivity is also crucial, and every effort is naturally directed towards minimizing various time-related metrics, which include time to detect (TTD)[1], time to engage (TTE), time to acknowledge or diagnose (TTA), and time to repair or mitigate (TTR). Furthermore, upon the successful resolution of incidents, an essential post-incident review is initiated to record the corrective or preventive actions taken for a specific incident. This phase also plays a pivotal role in identifying valuable lessons learned and proactively preventing similar incidents in the future.

3.3 Intelligent Incident Management Procedure Tasks

Drawing from the capabilities of AIOps for incident management, our proposal aims to organize maintenance routines within a standardized approach to conduct incidents from creation through diagnosis to resolution. This redesigned process involves a sequential workflow, guiding incidents through four well-defined phases, starting from initial reporting and resulting in mitigation and postmortem analysis. Indeed, the objective is to automate as many maintenance tasks as possible, with a view to optimize the reporting time, diagnosis and triage time, and resolution time. Such a protocol should not only facilitate the resolution of incidents but also serve to document the maintenance context. Figure 6 provides a clear visual representation of this categorization[2]. It should be noted that an incident typically goes through many sub-phases in the illustrated workflow, but there may be instances where certain phases may not be necessary. In such cases, a phase may be skipped if it is considered unnecessary or doesn't contribute to resolving the reported issue. For example, if an incident is categorized in a manner that has already undergone prior investigation, there might be no need for a root cause analysis phase.

In the following, we provide from our perspective an extensive review of the essential subcategories within the incident management procedure. Our goal is to examine the most relevant approaches proposed for each of these phases. It is worth mentioning that some previous work has already defined certain terms related to incident management. For instance, Chen et al. [66] defined the incident management procedure as a three-step process involving incident reporting, triage, and mitigation. However, their definition remains generic and does not delve into the subcategories of these phases, such as addressing the problem of incident classification and correlation. On the other hand, Notaro et al. [196] focused on studying failures and developed a taxonomy based on

[1]Also known as time to report, accounting for both manual reporting and predicted incidents

[2]For clarity, detection involves both detection and prediction, while classification also includes deduplication.

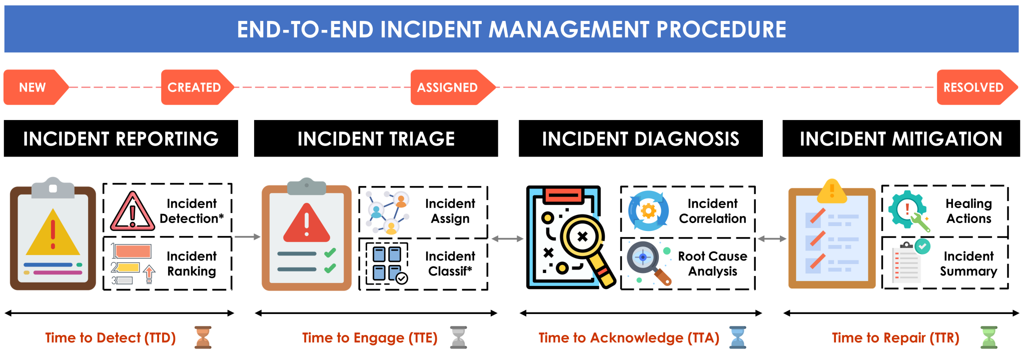


Fig. 6. Standardized end-to-end procedure proposed for Incident Management in the context of AIOps.

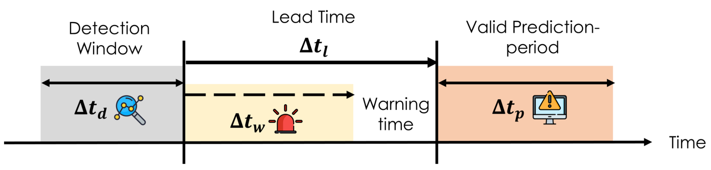


Fig. 7. Time relations in online failure prediction.

proactive and reactive approaches, with significant emphasis on the reporting phase. However, their research appears to neglect other important phases. Additionally, Zhang et al. [286] provided a formal definition that includes detailed phases like assignment, prioritization, fault localization, and mitigation. However, this survey solely concentrates on software bugs and does not generalize to other specific areas. In this work, our aim is to cover all these use cases under a unified taxonomy, regardless of the terminology used (failures, bugs, anomalies, etc.) or the specific industry focus.

Incident Detection. Incident detection refers to the process of identifying and recognizing deviations from normal operations indicating the presence of abnormal behavior or faulty events that may indicate the occurrence of an incident (e.g., error or anomaly). This process involves monitoring and analyzing various data sources (e.g., KPI metrics, system logs, and user reports). For instance, the *anomaly detection* research domain falls within this subcategory.

Incident Prediction. Incident prediction refers to the process of forecasting, anticipating, or estimating the likelihood of potential incidents, primarily failures, before they occur. It involves leveraging available historical data, along with other relevant factors, to proactively identify and assess potential risks and vulnerabilities. By analyzing patterns, trends, abnormal events, and anomalies in historical and current data using advanced analytics techniques, incident prediction aims to take preventive actions and minimize the impact of future incidents. Incident prediction can be categorized into offline and online methods. The offline category includes Software Defect Prediction (SDP) [79, 137], which entails assessing failure risks by executing specific functional units or analyzing portions of the source code. Another technique in this category is fault injection [190, 227], where deliberate faults are introduced into a functioning system through stress-testing to evaluate its fault tolerance level. Conversely, online prediction occurs during system runtime. It involves techniques like software rejuvenation [26, 246], which addresses resource exhaustion and prevents unexpected system failures caused by aging or accumulated faults in software systems. Additionally, online prediction involves estimating the remaining useful lifetime of the system. This category also involves real-time predictions of hardware and software failures, taking into account time constraints, as depicted in Figure 7. For a prediction to be considered valid, it must be made with a lead-time (Δt_l) greater than the minimal warning time (Δt_w). Moreover, the prediction is only considered valid if the failure occurs within a specific time period called the prediction period (Δt_p). To make these predictions, data up to a certain time horizon (Δt_d), referred to as the data window size, is used.

Incident Prioritization. Incident prioritization is the process of categorizing and ranking incidents based on their urgency, impact, and business priorities. It involves evaluating the severity of the incident, considering factors such as the affected systems, services, and the potential business impact. Incident prioritization ensures as well that resources are allocated appropriately, with relatively critical incidents receiving immediate attention and resources.

Incident Assignment. Incident assignment entails the allocation of incidents to the relevant individuals or teams responsible for investigating and resolving them. This process involves analyzing the information contained in incident reports, considering factors such as the nature and complexity of the incident, as well as the skills and availability of the assigned personnel. This process is commonly referred to as incident triage in several work [60, 290]. However, triage in a general sense, does not refer only to assignment but also to other tasks such as classification and identification of duplicate incidents.

Incident Classification. Incident classification involves the systematic grouping and/or categorization of incidents based on their distinct characteristics, symptoms, or impact. This classification process establishes a structured framework that enhances the understanding and effective management of incidents. This process can be seen as a refinement of the assignment procedure. In fact, it is necessary to categorize incidents and assign them to specific teams based on their respective topics, whenever possible. For instance, the technical service team within an IT organization may be tasked with resolving incidents related to resource saturation, processes that put a significant demand on the CPU and SWAP, security vulnerabilities, etc. To efficiently manage these issues, the technical team should assign knowledgeable staff to each topic. As such, it would be optimal, once an incident is assigned to the responsible team, to promptly identify the appropriate topic as the initial step in the procedure. Surprisingly, despite its crucial role in optimizing incident management time response, incident classification has not received sufficient coverage or extensive attention. While some studies, such as [27, 203], have approached this issue by treating it as a content optimization problem, others have included it as part of prioritization [131, 242]. Some researchers have even considered it within the scope of duplicate detection [36, 112]. However, we believe that there are inherent differences between these categories. Incident classification primarily focuses on associating the incident with a specific topic or category, regardless of the assigned personnel, incident priority, presence of similar incidents, or whether it is a new incident.

Incident Deduplication. Near-duplicate incident detection is the process that efficiently identifies incidents that are closely related or exhibit significant similarities, grouping them into specific buckets that correspond to distinct problems. This process involves real-time analysis of incoming incidents to determine their similarities, overlaps, and commonalities with historical incidents that pertain to the same topic. By identifying duplicates, incident deduplication reduces redundancy and incident management efforts and prevents unnecessary resource allocation. In fact, Anvik et al. [28] conducted a comprehensive empirical study using bug repositories from Eclipse and Firefox,

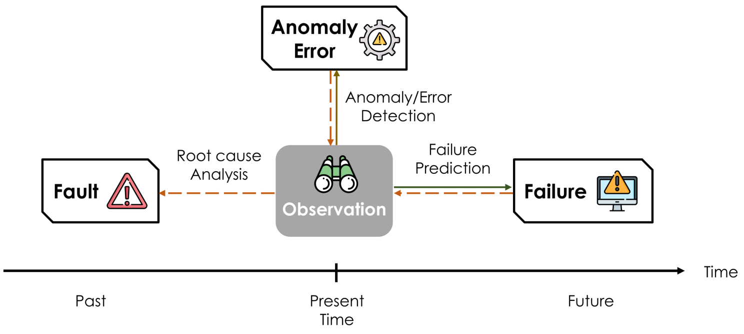


Fig. 8. Distinction between Root Cause Analysis, Anomaly Detection, and Failure Prediction.

which revealed that a significant portion (20%-40%) of bug reports are flagged as duplicates by developers. This study provides concrete evidence of the necessity to detect duplicate incidents. This process can also be considered as a further refinement of the incident classification problem.

Root Cause Analysis. Root cause analysis (RCA), also known as root cause diagnosis, plays a pivotal role in the incident management procedure. It is a systematic process that aims to investigate and identify the underlying causes and contributing factors of incidents, which we commonly refer to as faults. As depicted in Figure 8, RCA delves into the fundamental fault behind the occurrence of an incident, recognizing that it may not necessarily stem from a faulty manipulation or defective source code. Its objective goes beyond addressing the symptoms. Instead, it seeks to uncover and understand the root cause, even if it originates from external factors. In related studies, this process is often referred to as Fault Localization [261]. These studies focus on pinpointing the specific set of components (such as devices, network links, hosts, software modules, etc.) that are associated with a fault that has caused or may cause a particular failure. However, root cause analysis goes beyond component localization and extends its investigation to faulty actions or risky external factors that may lead to abnormal behaviors within the system.

Incident Correlation. Incident correlation study involves the analysis of multiple incidents or events with the aim of identifying relationships, dependencies, or shared characteristics among them. Through this process, a holistic perspective of the incident landscape can be achieved, allowing to uncover potential hidden patterns or trends. Incident correlation works alongside root cause analysis to assess how the underlying causes and faulty components or behaviors can affect other incidents. This helps in facilitating more efficient incident resolution. The task is considered challenging due to the inherent complexity and interdependence among components in software systems [31].

Incident Mitigation. Incident mitigation also known as remediation, refers to the process of minimizing the impact and severity of an incident. It involves taking proactive and automatic measures to contain, resolve, or reduce the effects of the incident. Incident mitigation can include implementing temporary workarounds, applying fixes or patches, activating backup systems, or engaging specialized resources or teams to restore normal operations. As mentioned in [196], contributions related to remediation actions have been relatively fewer compared to incident prediction, detection, triage, and diagnosis tasks. This could be attributed to the fact that once the underlying problem is identified through diagnosis, the necessary recovery steps become readily identifiable and achievable. In many cases, historical incidents with similar resolutions can be referenced, eliminating the need for using complex models.

3.4 Desiderata for Effective Intelligent Incident Management

Creating intelligent, data-driven strategies for incident management is an intricate endeavor that extends beyond the design of effective machine learning techniques. Simply relying on high-performing machine learning or big data mining models is insufficient for successfully adopting AIOps solutions. To ensure the effectiveness of such solutions, they must adhere to a set of established criteria, which we refer to as desiderata. Drawing from numerous reviewed studies, including [73, 143, 161, 165, 296], we have compiled a comprehensive list of requirements that should be considered, either fully or partially, when constructing AIOps solutions. These requirements are as follows:

- ❑ **Trustablity and Human in the loop.** The literature claims that the requirements for employees skills and mindsets change with the introduction of AIOps [208]. Manual activities tend to shift towards adaptation and auditing tasks while dealing with AI requires a different approach focused on recognizing patterns from raw data, deviating from the traditional developer mindset. This transition raises questions about trust in AI capabilities and what it can offer. Consequently, adopted AIOps approaches should incorporate years of field-tested engineer-trusted domain expertise iteratively and interactively into the learning, updating, and explanation phases of sophisticated machine learning models built on raw data. IT professionals possess valuable domain knowledge and insights acquired through years of experience in managing and troubleshooting IT systems. While not all of their best practices may scale with the AIOps trends, their expertise often extends beyond raw data analysis. They have a deep understanding of the underlying technology, infrastructure, applications, and business requirements. Hence, it is crucial to fully leverage and model this expertise into AIOps solutions. This can be achieved by providing mechanisms that incorporate the human in the loop, allowing for interaction, updates, and corrections to the models when necessary. Active Learning [214] can be specifically beneficial in this context.
- ❑ **Interpretability.** AIOps solutions should prioritize interpretability, even if it comes at the expense of model performance. In the context of AIOps, interpretable models are preferred when high-performing models lack interpretability. Model transparency enables users to fully understand, interact with, and reason about the recommendations made by the model, which can help gain support from upper management in following those recommendations. However, interpreting AIOps models comes with certain constraints and requirements. In a study by [165], different factors influencing AIOps model interpretation are investigated across three key dimensions. (1) *Internal Consistency* which assesses the similarity between interpretations derived from AIOps models trained under the same setup. It examines whether the interpretations obtained from an AIOps model are reproducible when the model is trained with the same data and implementation across multiple executions. (2) *External Consistency* which focuses on the similarity between interpretations derived from similar-performing AIOps models on a given dataset. Intuitively, interpretations derived from a low-performing interpretable model could be trustworthy only if the interpretable model has the same interpretation as other machine learning models on a given dataset. (3) *Time Consistency* which captures the similarity between interpretations derived from an AIOps model across different time periods. AIOps models should not only reflect the trends observed in the most recent training data but also capture and reflect trends observed over a longer period. It is important to note that some previous work, such as [38] in defect prediction,

has shown that models trained on one time period may not generalize well when tested on a different time period. Additionally, the size of the training data can impact the derived interpretations of the models.

❏ **Scalability.** AIOps solutions must efficiently handle large-scale data in complex IT environments where significant amounts of monitoring and log data are expected. These environments can encompass thousands to millions of components, including servers, network devices, and applications. To go beyond effective modeling and accurate results, it is crucial for AIOps solutions to be implemented within robust architectures that excel at ingesting, storing, and processing big data efficiently. Scalable architectures and data processing frameworks play a key role in distributing the workload and effectively handling the high volume of data. Additionally, when considering the adopted approaches, AIOps solutions should leverage scalable computing techniques, such as distributed and federated learning, as discussed in studies like [41, 83, 191], to enable parallel processing and distributed data analysis. Scalability also involves optimizing the utilization of computational resources. AIOps solutions should possess the capability to dynamically allocate and distribute resources based on the data volume and processing requirements.

❏ **Maintainability and Adaptability.** The concepts of maintainability and adaptability are crucial in AIOps, as they aim to minimize the need for ongoing maintenance and repetitive fine-tuning. This consideration is essential because DevOps engineers, who are responsible for managing and maintaining these solutions, often have a multitude of responsibilities and may not possess extensive expertise in machine learning. Therefore, AIOps solutions should strive for a high degree of automation and self-management of routine tasks such as data preprocessing and regular model training to reduce the reliance on continuous manual interventions. To achieve this, self-adjusting algorithms and automated pipelines can be employed [43]. In addition, leveraging advanced machine learning techniques such as transfer learning [199] and one-shot learning [249] can greatly benefit AIOps solutions. Instead of training models from scratch, pre-trained models that have been developed and fine-tuned by machine learning experts can be utilized to handle new data patterns.

❏ **Robustness.** AIOps solutions need to be built upon robust and stable machine learning models that can handle a wide range of scenarios and exhibit resilience to variations in data patterns. These models should be designed to be less sensitive to the noisy and incomplete data commonly encountered in real-world IT environments [73]. To ensure the reliability of the modeling process, robust preprocessing techniques, such as systematic data cleaning and effective imputation methods, can be employed. In addition, AIOps solutions must be capable of detecting and adapting to concept drift, which refers to the shifts in underlying data distributions that occur in dynamic IT environments [164]. Robust algorithms and models, such as those based on online learning, can be leveraged to handle concept drift and maintain up-to-date insights in the face of evolving data patterns [51, 53]. Furthermore, AIOps solutions should generalize well across different IT environments. To achieve this, they should be trained on diverse and representative data that captures the underlying patterns and relationships applicable across various scenarios.

❏ **In-context Evaluation.** Unlike conventional machine learning evaluation scenarios, such as cross-validation, which are often not directly applicable to AIOps due to the unique characteristics of real-world IT environments, the concept of in-context evaluation emphasizes the need to assess the solution's performance in a context that closely resembles its actual production usage. Traditional evaluation methods typically assume that the data used for

Table 2. Proposed taxonomy to categorize AIOps approaches for Incident Management.

Context	**Incident Task**	Refers to the specific research area in which the proposed approach fits within the incident management procedure. More precisely, it addresses one of the distinct phases when handling the incident, as identified in our categorization, including reporting, triage, diagnosis, or mitigation.
	Focus Area	Refers to the specific application area. Some methods may be exclusively dedicated to a particular application domain. Different applications have varying requirements and constraints, and the reviewed method may need to be tailored accordingly.
	Maintenance Layer	Refers to one of the different layers of the system as highlighted in Section 2.3 that are targeted by the reviewed method.
Data	**Data Source**	Refers to the nature of data that the method is built upon. The proposed taxonomy includes various types of data sources, such as log metrics, source code, key performance indicators, topology (environmental characteristics), alerting signals, execution traces, network traffic, etc.
	Data Type	Consideration should be attributed to how data is represented. Even when using the same data source, different representations are possible and then imply different methodologies (e.g., source code can be represented as an Abstract Syntax Tree (AST), a sequence of predefined frames, or plain natural language text). To facilitate this understanding, a taxonomy of data types has been adopted. This taxonomy includes structured data, sequential data, graph data, time series (both univariate and multivariate), textual data, hierarchical data, etc.
Model	**Approach**	Represents the principal approach employed to address the problem at hand, taking into consideration the context and the data utilized. It elucidates the way in which the authors formalized and tackled the problem (e.g., clustering, nearest neighbor search, dimensionality reduction, deep learning, transformers, etc).
	Paradigm	Provides an abstract demonstration of how the method was approached to attain its objective. For instance, in the case of training predictive models, it may involve discerning whether the approach was supervised, semi-supervised, or unsupervised, or whether it entailed one-shot, multitasking, reinforcement, or transfer learning, etc.
	Evaluation Metrics	Relates to the evaluation metrics used to assess how well the method performs compared to other techniques currently used in the field.
	Package Availability	This factor is highly important to reveal the accessibility of both data and model packages for reproducibility purposes.
Particularities		The final category of our taxonomy concerns essential factors that highlight specific attributes and desired outcomes associated with AIOps which aligns with the requirements detailed in Section 3.4.

evaluation is identically distributed, which may not hold true in IT environments. Real-world data often exhibits temporal dependencies, concept drift, and dynamic patterns, which require specialized evaluation techniques that consider these factors. To conduct in-context evaluation, it is important to create evaluation frameworks that capture the particularities of the production environment. This involves using datasets with a broad range of scenarios, including normal operations, various types of incidents, and different environmental and temporal conditions. In addition to dataset selection, evaluating AIOps solutions in context also requires defining appropriate evaluation metrics and benchmarks that align with the desired outcomes and objectives. (See Section 4.2 for more details)

4 PROPOSED TAXONOMY

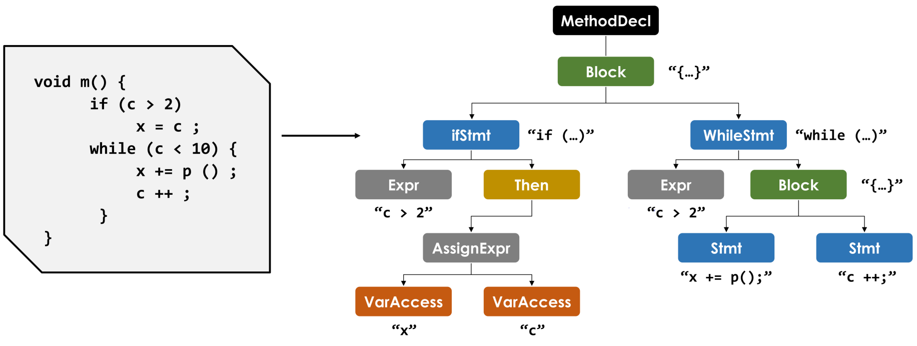
In this paper, we present a comprehensive taxonomy that categorizes research work related to AIOps for incident management. Our taxonomy encompasses primary groups, covering diverse factors that are essential to consider when evaluating the necessity, design, implementation, and reproducibility of the reviewed methods. Carefully choosing these categories enables us to analyze many dimensions that significantly impact AIOps for incident management. These dimensions include the context, representing environmental factors that drive and surround the proposed approach. Additionally, we examine factors pertaining to the characteristics of the data and their representation. Moreover, we explore the requisite measures necessary to render the data actionable for efficient analysis and interpretation. Of utmost importance is the detailed exploration of the design and implementation of the approach. We clarify the methodology adopted, the learning paradigm or research area employed, and how the approach was evaluated. Furthermore, we highlight particularities exhibited and interesting requirements validated by the proposed approach. Each of these categories is explained in detail in Table 2 to ensure inclusiveness and brings together all relevant aspects in a structured manner. Concerning the specific attributes highlighted for each method, which correspond to the requirements outlined for constructing AIOps models, our focus is solely on instances where a method explicitly addresses these criteria in the paper (e.g., robustness to noise or incorporating mechanisms for explainability). To illustrate, the adherence of a method to contextual evaluation criteria is established only if the experimental study explicitly aligns with the temporal requisites outlined in the evaluation protocol (refer to Section 4.2 for more details). An example of this would involve ensuring that anomalies within the test set must strictly postdate anomalies observed in both the training and validation sets.

In the following, we will delve into major factors that are exclusively relevant to our studied domain. Our focus is specifically to explore the different data sources and outline the evaluation methods employed for the proposed AI approaches.

4.1 Data Sources and Types

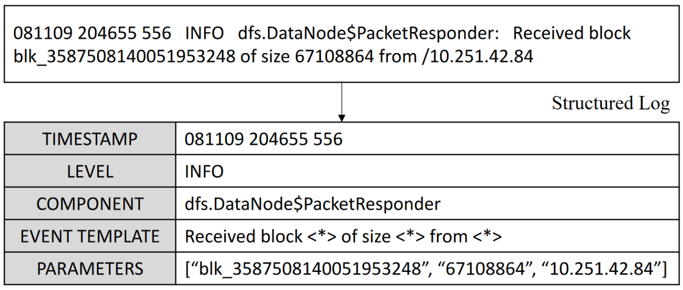
Data plays the most crucial role in incident management, serving as the fundamental building block that guides the design of the approach used to identify predictive or descriptive patterns within it. The primary objective is to use this data to effectively accomplish the desired task at hand. In an industrial setting, data is derived from a multitude of sources, including physical or software components, and can also be generated or edited by humans. One key characteristic of this data is its unstructured nature, lacking a standardized format and displaying non-homogeneity. Consequently, data collected from different sources require a pre-processing stage to perform subsequent analysis. Furthermore, data from the same source can be represented in various ways. We categorize data sources based on the following convention.

Source Code. It represents the fundamental building block of software, including various units such as functions, file codes, classes, and modules. It reflects the design, structure, and logic of different functionalities and services within the software system. In previous works, source code has been represented in diverse ways, including code metrics, which are handcrafted features derived directly from the source code. Code metrics play a critical role in software defect prediction by quantifying different aspects of the codebase that can impact software quality. These metrics cover a wide range, from static module-level metrics [172] for procedural languages, such as Lines of Code (LOC), Coupling Between Objects (CBO), Lack of Cohesion in Methods (LCOM), and Depth of Inheritance Tree (DIT), which provide insights into module complexity and potential defect-proneness. At the class level [67], metrics like Number of Methods (NOM), Weighted Methods per Class (WMC), and Response for a Class (RFC) offer indications of class complexity and the potential



```
void m() {
      if (c > 2)
            x = c ;
      while (c < 10) {
            x += p () ;
            c ++ ;
      }
 }
```

Fig. 9. Sample Java method and its Abstract Syntax Tree (AST).



```
081109 204655 556   INFO   dfs.DataNode$PacketResponder:   Received block
blk_3587508140051953248 of size 67108864 from /10.251.42.84
```

Structured Log

TIMESTAMP	081109 204655 556
LEVEL	INFO
COMPONENT	dfs.DataNode$PacketResponder
EVENT TEMPLATE	Received block <*> of size <*> from <*>
PARAMETERS	["blk_3587508140051953248", "67108864", "10.251.42.84"]

Fig. 10. Log parsing example [276].

occurrence of defects. Machine learning algorithms are trained using these metrics to identify patterns and relationships between code quality and defects. Another representation of source code is in the form of Abstract Syntax Trees (AST) [253], which capture the syntactic structure of the code by breaking it down into constituent elements such as expressions, statements, functions, classes, and variables (Figure 9). Source code has been modeled using the program spectrum [22, 55], which provides execution information from specific perspectives, such as conditional branches or loop-free intra-procedural paths and also as a sequence of tokens [193, 282] in many research works, particularly in Software Fault Localization.

Topology (Environment Features). Topology refers to the structure, either physical or logical, of the IT environment. It includes information about the components, connections, and spatial relationships within the system. This data source offers valuable insights into the overall architecture, providing details about servers, network devices, databases, etc. Additionally, topology data may include configuration settings, software versions, and other relevant features of the system. The presence of topology is important as it establishes the context necessary for identifying actionable and contextualized patterns within the data. Without the constraints and context provided by the topology, the detected patterns, while valid, may be misleading or distracting [205]. Topology has found extensive use in various areas, such as determining causality and localizing faults [154, 210], ranking incident [153], prediction of incident [143], as well as enhancing models explainability [212].

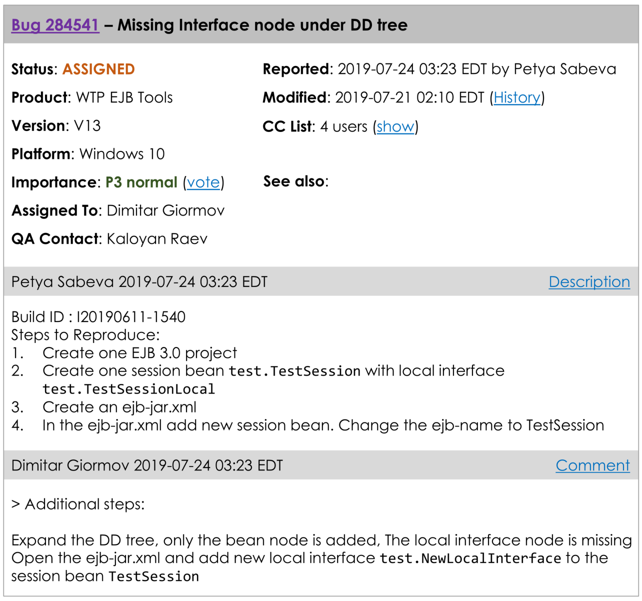
Event Logs. Logs consist of human-readable statements generated by software applications, operating systems, or devices to describe events or actions. They serve as valuable records, providing information about system activities, errors, warnings, and other relevant events. Timestamps are typically included in logs, along with details such as the event's source, severity, and description. Analyzing logs is essential for understanding the sequence of events leading to an incident, identifying abnormal behaviors, debugging issues, and ultimately pinpointing the root cause. In general, logs are semi-structured text that is produced by logging statements (e.g., `printf()`, `logger.info()`) within the source code. Once logs are collected, they need to be parsed to be utilized in various downstream log mining tasks, such as incident detection.

Parsing log messages is a crucial step in making logs usable for different analytical tasks. This process aims to transform the semi-structured log messages into structured log events by extracting constant parts and variables [104]. In the given example depicted in Figure 10, a log parsing scenario is presented, showcasing a log message obtained from the Hadoop Distributed File System (HDFS) [276]. A log message consists of two main components, the message header, and the message content. The message header, which is determined by the logging framework, is relatively straightforward to extract, including details like verbosity levels (e.g., INFO). On the other hand, extracting essential information from the message content proves to be more challenging due to its unstructured nature, primarily consisting of free-form natural language written by developers. Typically, the message content comprises both constants and variables. Constants represent fixed text provided by developers (e.g., the word "Received"), describing a particular system event. On the other hand, variables correspond to the dynamic runtime values of program variables, carrying contextual information. The set of constants forms the event template.

Numerous log parsing techniques have been proposed, including clustering-based approaches (e.g., LKE [93], LogSig [241]), heuristic-based methods (e.g., iPLoM [170], SLCT [245]), Evolutionary Algorithms such as MoLFI [177], and Frequent Pattern Mining techniques like Logram [71]. These algorithms are evaluated based on factors such as offline or online parsing mode, coverage, and alignment with domain knowledge. Parsed logs have been used in log mining algorithms with different approaches, including structured features [106, 276], log sequences [85, 173], graphs [189] and Finite State Automata (FSA) [160]. For more details, please refer to the work of [104, 106].

Key Performance Indicators (KPIs). These metrics serve as performance indicators for assessing the health status of IT infrastructures and services. They provide quantitative measurements that offer insights into system performance, availability, reliability, and response times. Examples of these metrics include service response time, error rates, disk reads, resource utilization (such as CPU, Swap, and memory consumption), etc. Typically, these metrics are represented as univariate or multivariate time series, which are utilized for various purposes such as detecting abnormal trends [150, 256], predicting failures by monitoring certain measures [138, 162], estimating the remaining useful lifetime of physical components, or storage capacity [298], identifying recurrent and unknown performance issues [151], and conducting root cause analysis and incident correlations [115, 223].

Network Traffic. This type of data involves the analysis of data packet flow within a computer network, including source and destination IP addresses, ports, protocols, and packet sizes. This data source provides valuable insights into communication patterns, network congestion, anomalies, and potential security threats. By analyzing network traffic, it is possible to identify and address network-related issues, pinpoint performance bottlenecks, and detect signs of malicious activities that could lead to significant incidents. Different approaches have been employed to model network traffic data. For instance, in one study [129], SNMP data was utilized to monitor network links and diagnose anomalies in network traffic. The authors treated flow measurements as multivariate time series collected over time, enabling the separation of traffic into normal and anomalous subspaces.


Bug 284541 – Missing Interface node under DD tree

Status: ASSIGNED	**Reported**: 2019-07-24 03:23 EDT by Petya Sabeva
Product: WTP EJB Tools	**Modified**: 2019-07-21 02:10 EDT (History)
Version: V13	**CC List**: 4 users (show)
Platform: Windows 10	
Importance: P3 normal (vote)	**See also**:
Assigned To: Dimitar Giormov	
QA Contact: Kaloyan Raev	

Petya Sabeva 2019-07-24 03:23 EDT Description

Build ID : I20190611-1540
Steps to Reproduce:
1. Create one EJB 3.0 project
2. Create one session bean `test.TestSession` with local interface `test.TestSessionLocal`
3. Create an ejb-jar.xml
4. In the ejb-jar.xml add new session bean. Change the ejb-name to TestSession

Dimitar Giormov 2019-07-24 03:23 EDT Comment

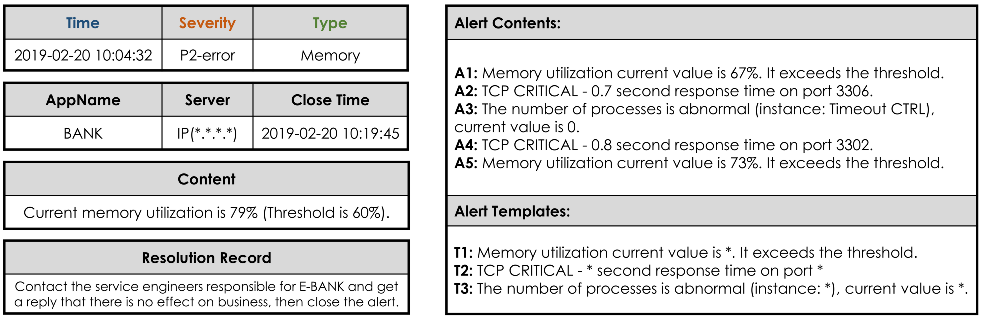
> Additional steps:

Expand the DD tree, only the bean node is added, The local interface node is missing
Open the ejb-jar.xml and add new local interface `test.NewLocalInterface` to the session bean `TestSession`

Fig. 11. Example of an incident report for a bug in Eclipse. This bug is about a missing node of XML files in Product Web Tools Platform [277].

In a subsequent work by the same authors [130], static features such as source and target destination addresses or ports were incorporated into the traffic data to detect and diagnose security threats and service outages. Another approach, presented in [255], involved representing network traffic as image-like structures using the IDX file format for encrypted traffic classification. This process involved mapping the network traffic data onto a 2D grid, where each grid cell represented a specific traffic attribute, such as packet size, source IP address, or destination port. The intensity or color of each pixel in the image reflected the value or frequency of the corresponding network traffic attribute at that particular location. Furthermore, network traffic has been represented as graphs by [35] to localize the sources of performance problems in networks. Probabilistic inference graphs were constructed from the observation of packets exchanged in the network infrastructure. Nodes of the inference graph are divided into root cause nodes (corresponding to internal IP entities), observation nodes (corresponding to clients), and meta-nodes, which model the dependencies between the first two types of nodes. Each node is also associated with a categorical random variable modeling the current state (up, troubled, down), which is influenced by other nodes via the dependency probabilities.

Incident Reports. Incident reports are valuable sources of information that comprehensively document the details, impact, discussions, and resolution of incidents. These reports are typically initiated by developers, testers, or end-users, capturing crucial information to aid incident management. As illustrated in Figure 11, incident reports commonly include an identification number and a title, along with contextual features such as the timeline of when the incident was reported and modified, the affected system or service's topology, the severity or importance of the incident, the assigned programmer responsible for resolution, and the incident's resolution status (e.g., new, unconfirmed, resolved). Of utmost importance, incident reports contain a detailed description of



Time	Severity	Type
2019-02-20 10:04:32	P2-error	Memory

AppName	Server	Close Time
BANK	IP(*.*.*.*)	2019-02-20 10:19:45

Content
Current memory utilization is 79% (Threshold is 60%).

Resolution Record
Contact the service engineers responsible for E-BANK and get a reply that there is no effect on business, then close the alert.

Alert Contents:

A1: Memory utilization current value is 67%. It exceeds the threshold.
A2: TCP CRITICAL - 0.7 second response time on port 3306.
A3: The number of processes is abnormal (instance: Timeout CTRL), current value is 0.
A4: TCP CRITICAL - 0.8 second response time on port 3302.
A5: Memory utilization current value is 73%. It exceeds the threshold.

Alert Templates:

T1: Memory utilization current value is *. It exceeds the threshold.
T2: TCP CRITICAL - * second response time on port *
T3: The number of processes is abnormal (instance: *), current value is *.

Fig. 12. [Left] Example of an alert on memory consumption from [295]. [Right] Explanation of alert template extraction.

the issue, including information on how to reproduce the problem, stack traces (in the case of a bug), and the expected behavior. Additional comments within the report may include discussions about potential solutions, diagnosis and root cause analysis, and actions taken to mitigate the incident. Attachments such as proposed patches, test cases, or screenshots may also be included to provide further context and support. Furthermore, incident reports should offer historical context to leverage past knowledge from similar incidents. Tagging relevant information from previous incidents enables the application of valuable lessons learned in the current resolution processes.

The challenge in processing incident reports lies in the diversity of data types, including structured, semi-structured, and unstructured data. For example, environment characteristics are typically presented in structured or tabular formats [66, 202, 277], while stack traces, problematic SQL queries and user traces fall into the category of semi-structured data [202, 286]. On the other hand, the description of the problem and the comments section dedicated to the analysis and diagnosis of the problem consist of unstructured natural language text [60, 132, 277], which requires data normalization. Various approaches have been employed to encode these different data types. For instance, both Chen et al. [60] and Pham et al. [202] utilized the FastText algorithm [47] for text encoding. Another approach employed in [132] was the use of Word2Vec [179], which builds pretrained subword vectors based on an external corpus and then fine-tunes them using historical incident data. Contextualization data, on the other hand, was handled using exponential family embeddings in the work of Pham et al. [202]. In addition, some researchers explored assignment information in incident reports to create what is referred to as a *Tossing Graph*, aiming to reduce the need for reassigning incidents to other developers [45, 114, 266].

Alerting Signals. Alerting signals are notifications generated by monitoring systems or tools when specific thresholds on time series metrics or conditions on event occurrences are violated. These signals serve as indications of abnormal behaviors, potential issues, or breaches of established thresholds. Examples of triggering events include high CPU usage, low disk space, high latency time, or application errors. Alerting signals act as an early warning system, allowing proactive identification and resolution of emerging problems before they escalate into failures. It is important to note that alerting signals are not raw or unprocessed data directly collected from monitored systems, such as KPI metrics or event logs. Instead, they are refined data derived from metrics or events based on a set of predefined rules. For example, Figure 12 (left) illustrates an example alert generated by the `AlertRank` Framework [295], indicating that the current memory utilization has exceeded the threshold of 79%, leading to a P2-error severity. This alert is derived from the analysis

Table 3. Contingency table.

	True Failure	**True Non-failure**	**Sum**
Prediction: Failure (Failure Warning)	True positive (TP) (Correct Warning)	False positive (FP) (False Warning)	Positives (POS)
Prediction: No failure (No Failure Warning)	False Negative (FN) (Missing Warning)	True Negative (TN) (Correctly no Warning)	Negatives (NEG)
Sum	Failures (F)	Non-Failures (NF)	Total (N)

of memory consumption over time and generated using alert templates within the source code, similar to log events, as shown in Figure 12 (right). Alerting signals are commonly used to identify and prioritize critical issues within a system [295]. They can also be utilized for categorizing similar or identical problems into specific categories [292], as well as deducing correlations among multiple simultaneous events [180].

Execution Traces. In addition to the main data sources mentioned above, which are utilized in the incident management process to develop data-driven approaches for incident detection, diagnosis, triage, and resolution, there are other data sources that are specifically relevant to certain tasks. One such example is code traces, which provide a hierarchical description of the modules and services invoked to fulfill a user request and are employed in the diagnosis task. These traces capture the flow of control, method invocations, input/output data, and interactions with external dependencies. Execution traces are particularly valuable for diagnosing complex or intermittent problems that are challenging to reproduce. Stack traces, for example, are detailed reports that provide information about the executed methods and their associated packages during a crash. They can be obtained through system calls in various programming languages. Stack traces have primarily been utilized in the context of crash deduplication, which involves identifying near-duplicate reports that indicate the same bug or error. Stack traces have been modeled using graphical representations [127]., sequence-based approaches [50, 82], or vectorization techniques such as n-grams and TF-IDF for information retrieval purposes [135, 219]. Another prominent type of execution trace comprises SQL queries executed to retrieve a service or important data. Parsing SQL queries efficiently is crucial for feeding them into analytical models to extract essential components such as tables, predicates, and projections [25, 29]. This enables their utilization as static features to identify schema issues in data models within databases and improve performance, such as recommending and selecting indexes [59, 78, 210], detecting anti-patterns [63], and identifying insider threats [128]. Alternatively, some methods treat SQL queries as natural language, employing techniques such as Query2Vec [111] to capture their semantic meaning. Heap dumps in Java memory analysis are another type of data that can be considered. They are snapshots of the Java heap memory taken at a specific moment in time. The Java heap is the region of memory where objects are allocated and deallocated during the execution of a Java application. A heap dump captures the complete state of the Java heap, including all objects and their attributes, such as instance variables and references. This provides a detailed view of the memory usage within the Java application. Heap dumps are particularly useful for analyzing memory-related issues, such as memory leaks [119, 273]. In the related literature, heap dumps are commonly represented as trees [133], graphs [171], or hierarchies [209].

Table 4. Metrics obtained from the contingency table.

Metric	Formula	Other names
Precision	$\frac{TP}{TP+FP}$	Confidence
Recall	$\frac{TP}{TP+FN}$	Support Sensitivity True positive rate (TPR)
False positive rate (FPR)	$\frac{FP}{FP+TN}$	Fall-out
Specificity	$\frac{TN}{TN+FP}$	True negative rate (TNR)
False negative rate (FNR)	$\frac{FN}{FN+TP}$	1 - recall
Negative predictive value	$\frac{TN}{TN+FN}$	
False positive error rate	$\frac{FP}{FP+TP}$	1 - precision
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	
Odds ratio	$\frac{TP \times TN}{FP \times FN}$	

4.2 Evaluation Metrics

To comprehensively evaluate the quality and performance of data-driven approaches in incident management tasks, it is crucial to assess them using appropriate metrics, also known as figures of merit. While machine learning metrics are commonly used to evaluate predictive models, it is important to note that relying solely on these metrics, such as contingency metrics, may not accurately reflect the models' performance in real-world scenarios, especially when considering time constraints. For example, in the case of incident prediction, the goal is to predict incidents while minimizing false alarms and maximizing the coverage of actual incidents. However, predicting incidents after the designated prediction period (Δt_p) is not considered accurate since the incident has already occurred, leading to suboptimal allocation of time and resources. Therefore, we introduce a set of established metrics, focusing primarily on two main tasks: detecting and predicting incidents. It is worth mentioning that these metrics can be adapted for other tasks as well, and several other metrics have been proposed to evaluate triage, diagnosis, and the effectiveness of automated remediation actions, which will be briefly covered. To organize the metrics effectively, we categorize them based on the nature of the model output.

Classification Metrics. Classification metrics are commonly derived from four cases, as shown in Table 3. A prediction is classified as a true positive if an incident occurs within the prediction period and a warning is raised. Conversely, if no incident occurs but a warning is given, the prediction is considered a false positive. If the algorithm fails to predict a true incident, it is categorized as a false negative. Finally, if no true incident occurs and no incident warning is raised, the prediction is labeled as a true negative. To compute metrics like precision and recall, the contingency table is populated with the number of true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN). The prediction algorithm is applied to test data that was not used to determine the parameters of the prediction method. This allows the comparison of prediction outcomes against the actual occurrence of incidents. The four possible cases are illustrated in Figure 13. It's worth noting that the prediction period (Δt_p) is instrumental in determining whether an incident is counted as predicted or not. Therefore, the choice of Δt_p also has implications for the contingency table and should align with the requirements of subsequent steps in the incident management process.

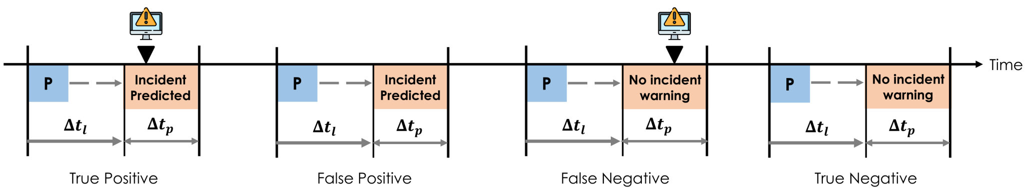


Fig. 13. A timeline showing true incidents and all four types of predictions TP, FP, FN, TN.

The metrics presented in Table 4 are derived from the contingency table (see Table 3). They are commonly used in pairs, such as precision/recall, true positive rate/false positive rate, sensitivity/specificity, and positive predictive value/negative predictive value. Different research areas may use different names for the same metrics, hence, the leftmost column indicates the commonly used terminology, while the rightmost column lists alternative names. It's important to note that improving precision (reducing false positives) often results in a decrease in recall (increasing false negatives) and vice versa. To balance the trade-off between precision and recall, the F-Measure is used as the harmonic mean of the two, assuming equal weighting. One limitation of precision and recall is that they don't consider true negative predictions. Therefore, it is necessary to consider other metrics in combination with precision and recall. The false positive rate is the ratio of incorrectly predicted incidents to the total number of non-incidents. A lower false positive rate is desirable, provided that the other metrics do not deteriorate.

Accuracy appears to be an appropriate metric for incident prediction due to the rarity of incidents. Achieving high accuracy by always classifying the system as non-faulty may be misleading because it fails to capture any incidents, resulting in a recall of zero. However, it is important to consider true negatives when assessing incident prediction techniques. Let's consider an example from [220]. Two prediction methods perform equally well in terms of true positives (TP), false positives (FP), and false negatives (FN), resulting in the same precision and recall. However, one method makes ten times more predictions than the other because it operates on more frequent measurements. The difference between these methods is reflected only in the number of true negatives (TN), which becomes apparent in metrics that include TN. True negatives are counted by considering predictions made when no incident was imminent and no warning was issued.

It is noteworthy that the quality of predictions does not depend only on algorithms but also on factors such as the data window size (Δt_d), lead-time (Δt_l), and prediction period (Δt_p). Predicting incidents at an exact point in time is highly unlikely, so predictions are typically made within a specific time interval (prediction period). The number of true positives is influenced by (Δt_p): a longer prediction period captures more incidents, increasing the number of true positives and impacting metrics like recall. Incident predictors often utilize an adjustable decision threshold. When the threshold is set low, incident warnings are raised easily, increasing the chances of capturing true incidents (resulting in high recall). However, a low threshold also leads to many false alarms, resulting in low precision. Conversely, if the threshold is set very high, the situation is reversed. To visualize this trade-off, precision/recall curves are used, plotting precision over recall for various threshold levels. Similarly, the receiver operating characteristic (ROC) curve plots the true positive rate versus the false positive rate (sensitivity/recall versus 1-specificity, respectively). This curve assesses the model's ability to distinguish between incidents and non-incidents. The

Table 5. Metrics used for regression tasks in incident management tasks.

Metric	Formula
Mean Absolute Error (MAE)	$\frac{1}{n}\sum_{i=1}^{n}\lvert\hat{y}_i - y_i\rvert$
Root Mean Squared Error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$
Mean Absolute Percentage Error (MAPE)	$\frac{1}{n}\sum_{i=1}^{n}\left\lvert\frac{\hat{y}_i - y_i}{y_i}\right\rvert \times 100\%$
R-squared (R2)	$1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$
Explained Variance Score	$1 - \frac{\mathrm{Var}(y - \hat{y})}{\mathrm{Var}(y)}$

closer the curve is to the upper-left corner of the ROC space, the more accurate the model is. The Area Under the Curve (AUC) is defined as the area between the ROC curve and the x-axis and measures the probability that a data point from an incident-prone situation receives a higher score than a data point from a non-incident-prone situation. By summarizing the capacity of a prediction algorithm to discriminate between incidents and non-incidents, the AUC converts the ROC curve into a single number. A random predictor has an AUC of 0.5, while a perfect predictor achieves an AUC of 1.

Regression Metrics. Regression metrics are commonly used in tasks such as remaining useful lifetime estimation or anomaly detection, particularly when applied to time series metric data to identify outliers. Unlike classification metrics, the metrics used for regressors remain consistent across conventional machine learning models, as shown in Table 5. Mean Absolute Error (MAE) is a metric that represents the average absolute difference between the predicted values $\hat{y}_i$ and the actual values y_i. It provides a measure of the average magnitude of errors. Several works have utilized MAE for regression tasks [197]. Root Mean Squared Error (RMSE) is similar to MAE, but it takes the square root of the average squared differences between the predicted values and the actual values. RMSE penalizes larger errors more significantly. It has been used, for example, in anomaly detection [134]. Mean Absolute Percentage Error (MAPE) measures the average percentage difference between the predicted values and the actual values. It is particularly useful when evaluating the accuracy of predictions relative to the scale of the target variable [146]. R-squared (R2) is a metric used to represent the proportion of variance in the target variable that is explained by the predicted values. R2 ranges from 0 to 1, where a value of 1 indicates a perfect fit and a value of 0 indicates no improvement over a naive baseline [227]. Finally, the explained variance score, similar to R2, measures the proportion of variance in the target variable that is explained by the predicted values. However, it is based on the variance of the residuals and can be used as an alternative metric for evaluation [227].

Other Metrics. In addition to the commonly used regression and classification metrics, there exists a set of specialized metrics that are particularly relevant for evaluating the effectiveness and performance of specific incident management tasks, such as fault localization, incident correlation, and incident deduplication. These metrics are often specifically designed for incident management purposes and may have limited applicability outside this domain. Some of these metrics have been developed in alignment with specific research studies, while others are unique to the field of incident management. For example, Wang et al. [251] have introduced the metric of "normality" for detecting anomalies within a network traffic flow sequence.

In fault localization research, several specific metrics are used to evaluate the effectiveness of different techniques. One such metric is T-Score, which estimates the percentage of code that a programmer can ignore before identifying the first faulty location in the program [157, 215]. Another metric commonly used is EXAM (Expense metric), which measures the percentage of program statements that need to be examined before encountering the first faulty statement [116, 118]. In addition to these metrics, other research work have also employed the Wilcoxon signed-rank test [258, 259] as a statistical evaluation method. This test serves as an alternative to the paired Student's t-test when the assumption of a normal distribution in the population cannot be made to compare the effectiveness of two techniques, denoted as α and β. The test examines the one-tailed alternative hypothesis that β requires the examination of an equal or greater number of statements compared to α.

In software defect prediction, there are cases where it is more useful to evaluate the classes based on their predicted number of defects in a ranking manner. One approach to assess the performance of the prediction model is by calculating Spearman's correlation coefficient [232], as demonstrated in the study conducted by [72]. Another method used in this context is the cumulative lift chart, which compares the performance of two different models or strategies by plotting the cumulative gain against the number of cases. This approach has been also employed by [86].

In intrusion detection systems within practical network settings, Mirheidari et al. [180] conducted a comprehensive comparison of alert correlation algorithms. The study aimed to evaluate the performance of these algorithms using both quantitative and qualitative measures. The quantitative assessment focused on accuracy, while the qualitative evaluation delved into additional aspects such as extendibility and flexibility. These aspects refer to the algorithm's adaptability, localizability, and capacity to adjust to new conditions. Furthermore, the evaluation considered the algorithm's ability to parallelize tasks and the associated memory requirements. This evaluation aligns with the desirable attributes emphasized in Section 3.4, which outlines the desired characteristics for AIOps solutions in incident management.

In the context of incident triage and deduplication approaches, specific performance metrics have been adopted to improve the evaluation of these processes. These metrics include Mean-reciprocal rank [70, 211], Recall rate of order k [125, 222, 248], and Average Hit Ratio [23]. It is noteworthy that there are some metrics specifically designed to assess the process, rather than a particular algorithm or data-driven approach. While we previously discussed metrics such as Mean Time To Repair (MTTR), Mean Time To Engage (MTTE), and Mean Time to Detect (MTTD), other research works [66, 205] have introduced alternative terminology. For example, MTTR may be referred to as Mean Time to Repair, and MTTD as Mean Time to Detect. Furthermore, there are additional valuable measures that have been considered, such as Mean Time Between Failures (MTBF) [48], which evaluates the average duration between consecutive incidents or failures to assess system reliability.

5 A COMPREHENSIVE REVIEW OF AIOPS-BASED DATA-DRIVEN APPROACHES

In the following sections, we present a review of research efforts centered around AIOps-driven data-centric approaches tailored specifically for incident management. We begin with an insightful overview of related and analogous studies, serving as an invaluable compass for those keenly interested in specific incident phases or application domains. We also review noteworthy surveys that offer targeted insights into particular facets of the field. Subsequently, we delve into an in-depth examination of the pioneering techniques that have found their place within the incident management literature. Each facet of incident management tasks is described based on our taxonomy through summaries and evaluations featured in dedicated tables. These tables serve as efficient reference points, providing readers with a clear and concise overview of each task. Furthermore, we assess the extent to which the reviewed papers meet the requisite criteria.

5.1 Prior Research Efforts in AIOps and Incident Management

Table 6. Summary and categorization of Incident Management related papers

					Focus Area									Findings				Practical Adoption		
Ref.	**Year**	**Study Approach**	**AIOps**	**Classif. Taxonomy**	**Detection**	**Prediction**	**Ranking**	**Assignation**	**Classification**	**Deduplication**	**Root Cause**	**Correlation**	**Mitigation**	**Trends**	**Capabilities**	**Challenges**	**Desiderata**	**Implementation Details**	**Specialized Techniques**	**Research Innovation**
[161]	2013	Experience		•	•		•	•		•	•	•	•			•	•	•	•	
[286]	2015	Survey		•			•	•	•	•	•		•						•	•
[290]	2016	Review		•			•	•	•	•	•		•			•			•	•
[140]	2017	Survey		•	•	•			•		•							•	•	
[185]	2018	Review		•	•	•							•			•	•	•	•	•
[205]	2018	Report	•	•	•	•			•		•	•	•	•	•	•	•	•	•	
[73]	2019	Experience	•												•	•	•			•
[66]	2020	Experience	•	•	•		•	•			•	•	•		•	•	•	•	•	•
[226]	2020	Experience	•		•						•	•	•		•			•	•	
[46]	2021	Survey	•	•	•						•				•	•				
[208]	2021	Review	•	•	•	•					•		•	•	•	•	•	•	•	•
[196]	2021	Survey	•	•	•	•			•		•	•	•	•	•				•	
[165]	2021	Case Study	•													•	•	•	•	•
[216]	2022	Review	•											•	•	•				
Our Survey	2023	Review	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Over the past two decades, numerous research endeavors have been dedicated to effectively tackling the automated incident management procedure. This becomes particularly important given the rise of expansive IT systems and the continually expanding scale of their infrastructures. These advancements have led to the generation of various incident types spanning multiple target domains, necessitating swift reporting, comprehension, and mitigation. Numerous research initiatives, motivated by industrial imperatives, have been engaged in structuring the incident management procedure. These efforts involve the formulation of taxonomies or comprehensive frameworks, thoughtfully considering an array of requirements and challenges, while including a review of existing methods that can be adapted to address specific aspects of the proposed framework [46, 140, 185, 195, 208, 286, 290]. Others channel their focus into practical insights gained from real-world industrial scenarios through experience papers and use case studies, offering effective implementation details, specialized techniques and research innovations [66, 73, 165].

As AIOps emerges as a novel and cross-disciplinary research frontier involving diverse domains and applications, it's worth noting that earlier works [140, 161, 185, 286, 290], particularly predating 2018, do not expressly reference the term AIOps, despite aligning with its conceptual framework and scoop. Additionally, some of these works may introduce non-data-driven methodologies, thus falling beyond the boundaries of the AIOps domain. In the following, we present a comparative analysis of related research that is either closely aligned or shares some similarities to our comprehensive examination of AIOps in incident management. The objective is to pinpoint key differentiating criteria, including their focal areas, whether a taxonomy is proposed, notable findings, and the provision of practical insights to facilitate effective implementation. Furthermore, we delve into

Table 7. Summary and categorization of specific data-driven surveys and reviews with a focus on particular Incident Management phases

			Target Area				**Data sources**									
Ref.	**Year**	**Focus and Scoop Area**	**Technical**	**Application**	**Functional**	**Business**	**Source Code**	**Topology**	**Event Logs**	**KPIs/SLOs**	**Traffic Network**	**Reports**	**Alerts**	**Traces**	**Type of AI Techniques**	**Evaluation Metrics**
[57]	2009	Cyber Intrusion Detection	•					•		•	•				Statistical Models Conventional ML Classifiers Clustering	N/A
[251]	2013	Network Anomaly Detection	•								•				Statistical Models Conventional ML Classifiers Clustering	Domain-specific metrics
[109]	2015	Bottleneck Identification	•	•				•		•	•				Statistical Models Conventional ML Classifiers Bayesian Models Conventional ML Regressors Rule-based Models Clustering	Contingency metrics
[109]	2019	IoT Systems Anomaly Detection	•	•						•	•				Statistical Models Conventional ML Classifiers Bayesian Models Conventional ML Regressors Deep Learning Models Auto Encoders Dimensionality Reduction Clustering Pattern Mining	N/A
[104]	2021	Log Anomaly Detection	•	•	•				•						Statistical Models Conventional ML Classifiers Deep Neural Networks Dimensionality Reduction Clustering Pattern Mining	N/A
[296]	2021	Log Anomaly Detection	•	•	•				•						Conventional ML Classifiers Deep Learning Models Dimensionality Reduction Clustering Pattern Mining	
[220]	2010	Online Failure Prediction	•	•	•	•		•	•	•	•				Statistical Models Conventional ML Classifiers Bayesian Models Conventional ML Regressors Dimensionality Reduction Rule-based Models Clustering Pattern Recognition	Contingency metrics ROC/AUC
[86]	2012	Software Defect Prediction			•	•	•								Conventional ML Classifiers Ranking-based Models	ROC/AUC Spearman correlation Friedman yest
[121]	2017	Software Defect Prediction			•	•	•								Statistical Models Conventional ML Classifiers Bayesian Models	Contingency metrics
[113]	2019	Faults Prediction	•	•	•	•			•	•	•				Statistical Models Conventional ML Classifiers Bayesian Models Conventional ML Regressors Deep Learning Models Rule-based Models	Contingency metrics

Ref.	Year	Focus and Scoop Area	Target Area				Data sources								Type of AI Techniques	Evaluation Metrics
			Technical	Application	Functional	Business	Source Code	Topology	Event Logs	KPIs/SLOs	Traffic Network	Reports	Alerts	Traces		
[75]	2021	Railway Failures Prediction	•							•			•		Conventional ML Classifiers Bayesian Models Conventional ML Regressors Deep Learning Approaches Auto-Encoders Generative Adversarial Model	Contingency metrics Regression metrics
[44]	2022	Remaining Useful Lifetime	•							•					Statistical Models Conventional ML Regressors Deep Learning Approaches Auto-Encoders Generative Adversarial Models Transfer Learning Reinforcement Learning	Regression metrics
[23]	2014	Bug Triage			•	•	•					•			Conventional ML Classifiers Bayesian Models EM models Information Retrieval	Contingency metrics
[231]	2014	Memory Leak Diagnosis		•			•							•	Statistical Models Graph Mining	N/A
[94]	2015	Fault Diagnosis	•	•						N/A					Statistical Models Signal-based Models	N/A
[261]	2016	Fault Localization			•	•	•								Program Spectrum-based Models Statistical Models Conventional ML Classifiers Rule-based approaches Pattern Mining	
[230]	2017	Root Cause Analysis	•	•	•	•				N/A					Statistical Models Conventional ML Classifiers Bayesian Models Rule-based Models Pattern Mining	Complexity Domain-specific metrics
[180]	2013	Intrusion Alerts Correlation	•					•			•				Statistical Models Conventional ML Classifiers Similarity-based Models	Accuracy Domain-specific metrics

surveys and reviews that are centered on specific phases within incident management, such as failure prediction or root cause analysis, without offering an exhaustive examination of the entire process [23, 44, 57, 75, 86, 94, 104, 109, 109, 113, 121, 180, 220, 230, 231, 251, 261, 296]. Additionally, we explore works that target specific data sources, such as anomaly detection in log event data.

Table 6 compiles an overview of the most significant AIOps and Incident Management research papers. Our objective is to assist readers in locating pertinent works aligned with their specific inquiries and research subjects. Moreover, we aim to pinpoint research gaps and derive innovative ideas from existing literature. We classify the selected research into distinct factors. This classification assists readers in various approaches, including survey papers that provide a holistic snapshot of the current state-of-the-art, literature reviews that delve into analyzed literature and conduct technical comparisons based on objective criteria, and studies that offer practical insights such as industrial experience papers and use cases. Notably, since 2018, pioneering studies in incident management have increasingly incorporated the AIOps acronym. For each of these papers, we assess the extent to which they cover all phases of incident management as defined in Section 3.3. While a majority of these works concentrate on the reporting phase (detection and prediction), along with root cause analysis and mitigation, a clear void exists in the coverage of aspects such as ranking, assignment, and deduplication. It is also noteworthy to highlight that some research papers have a limited scope concerning the broad concept of incidents. For example, Zhang et al. [286] address solely software bug management, thereby overlooking other areas like remaining useful lifetime estimation. Our analysis also extends to insights into the implementation of AI models in real-world scenarios. We report whether these papers provide a comprehensive depiction of related challenges, requirements, and articulate gaps for potential future research. The study conducted by Notaro et al. [196] aligns closely with our work. Nevertheless, it exhibits gaps in addressing certain aspects, such as assignment and deduplication. Furthermore, it lacks an in-depth technical exploration of the challenges and requirements for establishing AIOps in production scenarios.

In Table 7, we present an overview of the most relevant surveys and reviews that focus on specific phases within the incident management procedure, rather than offering a comprehensive analysis of the entire process. These surveys delve into particular aspects, such as anomaly detection [57, 251], online failure prediction [220], and more. We organize this research into categories based on their primary focal points and scopes. For instance, certain surveys narrow their scope to specific industries, such as the work by [75] concentrating solely on predicting failures within the railway sector. Meanwhile, Mirheidari et al. [180] address the issue of correlating alerts for cyber intrusion detection systems. Importantly, this does not exclude the potential applicability of the reviewed approaches in other practical domains. Furthermore, we outline the targeted maintenance levels, available data sources, and data-driven methodologies employed in these surveys. Some of these studies also emphasize the evaluation metrics used to assess the reviewed methods. In specific cases, like the research conducted by [230] focusing on root cause analysis, domain-specific metrics are employed, as discussed in Section 4.2 (e.g., these particular metrics assess aspects such as the ability to pinpoint multiple faults simultaneously, in identifying unknown faults, etc.).

5.2 Incident Detection Methods

Incident detection approaches are reactive methods of incident management that aim to track and identify abnormal states or behaviors in a system. Their purpose is to either anticipate failures before they occur or mitigate the consequences of failures after they have happened. This is driven by the understanding that, despite employing advanced prediction techniques, it is impossible to completely eliminate the occurrence of failures. These methods also aid in comprehending the causal relationships, as well as understanding the temporal characteristics that lead to incidents. Incident detection methods often leverage unsupervised learning approaches, primarily because acquiring high-quality, sufficient, and balanced data labels poses significant challenges. Among the notable techniques employed in this context, we find clustering methods, dimensionality reduction techniques, and auto-encoders. Additionally, other approaches, such as graph mining, and statistical models, have been employed in this context.

At the network level, Lakhina et al. [129] propose an anomaly detection method for network traffic analysis using SNMP data. The authors apply Principal Component Analysis (PCA) to link flow measurements collected over time to separate traffic into normal and anomalous subspaces. Anomalies are identified by reconstructing new observations using abnormal components. If the reconstruction error exceeds a predefined threshold based on explained variance, the data point is considered anomalous. In a subsequent work [130], the same authors argue that by analyzing the distributions of packet features (IP addresses and ports) in flow traces, it is possible to reveal the presence and structure of various anomalies in network traffic. They use entropy as a summarization tool to quantify the randomness or irregularity in these feature distributions. The method has demonstrated its ability to adapt to new kinds of anomalies. However, Ringberg et al. [217] have raised concerns about the application of PCA in [129] by identifying its susceptibility of false positive rates to small normal subspace noises. Thus, Pascoal et al. [200] introduce another approach

Table 8. Summary of reviewed AIOps Incident Detection methods.

			Target Area				**Data sources**												
Ref.	**Year**	**Focus**	**Technical**	**Application**	**Functional**	**Business**	**Source Code**	**Topology**	**Event Logs**	**KPIs/Metrics**	**Traffic Network**	**Reports**	**Alerts**	**Traces**	**Approach**	**Paradigm**	**Evaluation Metrics**	**Code**	**Dataset**
[129]	2004	Network	●								●				PCA	UNSUP	ACC, FPR		●
[130]	2005	Network	●					●			●				FD	UNSUP	ACC, FPR		●
[200]	2012	Network	●								●				PCA	UNSUP	PREC, REC, FPR		
[120]	2015	Network	●								●				K-Means, PSO	UNSUP	F1, PREC, REC, FPR		●
[201]	2017	Network	●							●	●				ARIMA, PL, ACODS	UNSUP	REC, FPR, ROC		●
[37]	2017	Network	●								●				HsMM	UNSUP	FPR, FNR, TNR		
[270]	2017	Network	●							●	●				KL-Div	UNSUP	FPR, ACC, ROC		●
[80]	2021	Network	●	●							●				GNN	UNSUP	F1, PREC, REC	●	●
[283]	2018	Edge Strm	●								●				AE, K-Means	UNSUP	AUC	●	●
[281]	2019	Edge Strm	●								●				Anomalousness Score	UNSUP	PREC	●	●
[58]	2021	Edge Strm	●								●				Frequency Factorization	UNSUP	AUC	●	●
[225]	2013	Cloud	●	●						●					KNN, HMMs, K-Means	UNSUP	PREC, REC, ACC, FPR		●
[247]	2014	Cloud	●	●	●					●					ESD Test	UNSUP	F1, PREC, REC	●	
[158]	2015	Cloud		●						●					RF	SUP	AUC		
[228]	2017	N/A	●	●						●					EV Theory	UNSUP	ROC	●	●
[274]	2018	Web Apps		●						●					VAE, KDE	UNSUP	F1, AUC, Alert Delay	●	●
[150]	2018	Web Apps		●						●					DBSCAN	UNSUP	NMI, F1		●
[213]	2019	Cloud		●						●			●		SR-CNN	UNSUP	F1, PREC, REC	●	●
[294]	2021	Software Changes		●	●	●			●	●					Multimodel LSTM	UNSUP	F1, PREC, REC, MTTD	●	●
[256]	2022	Software Changes		●	●					●					LSTM	SELFSUP	F1, PREC, REC	●	●
[233]	2019	N/A	●	●						●					Stochastic RNN	UNSUP	F1, PREC, REC	●	●
[285]	2019	N/A	●	●						●					CNN-LSTM	UNSUP	F1, PREC, REC	●	●
[33]	2020	N/A	●	●						●					GAN	Adversial	F1, PREC, REC	●	●
[149]	2021	N/A	●	●						●					Hierarchical VAE, MCMC	UNSUP	F1, Interpretation Score	●	●
[93]	2009	Log Anom		●	●				●						FSM	UNSUP	FPR	●	●
[276]	2009	Log Anom		●	●		●		●						PCA	UNSUP	ACC	●	●
[160]	2010	Log Anom		●	●				●						Mining Invariants	UNSUP	ACC, FPR	●	●
[105]	2018	Log Anom		●	●				●						Hierarchical Clustering	UNSUP	F1, PREC, REC	●	
[85]	2017	Log Anom		●	●				●						LSTM	UNSUP	F1, PREC, REC, FPR, FNR	●	●
[173]	2019	Log Anom		●	●				●						Template2VEC, LSTM	UNSUP	F1, PREC, REC	●	●
[291]	2019	Log Anom		●	●				●						Word2VEC, biLSTM	UNSUP	F1, PREC, REC	●	●
[267]	2021	Log Anom		●	●				●						LSTM-based GAN	UNSUP, Adversial	F1, PREC, REC		●
[99]	2021	Log Anom		●	●				●						BERT	SELFSUP	F1, PREC, REC	●	●

Table 9. Study of particularities of AIOps Incident Detection methods.

Ref.	Interpretability	Scalability	Robustness	Temporal Evaluation	Human-in-the-loop
[37, 129]	●	●			
[130]	●				
[200]	●		●	●	
[120, 201]	●	●	●		
[270]	●	●	●	●	
[80]	●		●		
[283]		●		●	
[281]	●	●		●	
[58]	●	●	●	●	
[150, 225, 228, 247]	●	●	●	●	
[158, 213]	●	●	●	●	●
[274]	●		●	●	
[294]	●	●			●
[256]		●	●		
[233, 285]	●		●		
[33]		●	●	●	
[149]	●		●	●	
[93]	●			●	
[276]	●	●	●	●	●
[105, 160]	●	●	●		
[85]	●			●	●
[173]				●	●
[291]		●	●	●	
[267]		●			

that incorporated a resilient PCA detector combined with a robust feature selection technique in order to obtain adaptability to distinct network contexts and circumstances. Moreover, this approach eliminates the need for flawless ground-truth data during training, addressing a limitation highlighted in the conventional PCA approach present in [129]. Karami and Guerrero-Zapata [120] introduce a clustering approach for identifying attacks and anomalies, ranging from Denial of Service (DoS) to privacy breaches, within content-centric networks. This approach operates through two distinct phases. Firstly, a unique hybridization of the Particle Swarm Optimization (PSO) and K-means algorithm is employed in the training phase to accurately determine the optimal number of clusters and avoid being trapped in a locally optimal solution. In the subsequent detection phase, a fuzzy approach is employed by combining two distance-based methods, to effectively identify anomalies within newly monitored data in which false positive rates are reduced with reliable detection of intrusive activities.

Statistical approaches have also been leveraged within this context. For instance, Pena et al. [201] propose a Correlational Paraconsistent Machine (CPM) that employs non-classical paraconsistent logic (PL) in conjunction with two unsupervised traffic characterization techniques. These techniques involve Ant Colony Optimization for Digital Signature (ACODS) and Auto-Regressive Integrated Moving Average (ARIMA). The purpose of these methods is to analyze historical network traffic data and generate two distinct network profiles, known as Digital Signatures of Network Segment using Flow Analysis (DSNSF), which describe normal traffic behavior. The detection of anomalies is linked to the degrees of certainties and contradictions produced by paraconsistent logic when correlating two prediction profiles with real traffic measurements. In a different study, Bang et al. [37] propose an Intrusion Detection System (IDS) utilizing a Hidden Semi-Markov Model

(HsMM) tailored specifically for detecting advanced LTE signaling attacks on Wireless Sensor Networks (WSNs). The authors argue that traditional Hidden Markov Models (HMM) are limited in representing various potential transition behaviors, while HsMM overcomes this limitation due to its arbitrary state sojourn time, making it more suitable for analyzing time-series behavior. HsMM is effectively employed to model the spatial-temporal characteristics of the wake-up packet generation process. The detector then compares observed spatiotemporal features of a server's wake-up packet generation with the normal criteria established by the HsMM. Subsequently, an alert is triggered whenever a significant divergence occurs. Xie et al. [270] present an algorithm to detect long-term anomalies in WSNs, where a group of neighboring nodes is consistently affected by such anomalies over a substantial time frame, particularly in the presence of DoS and sinkhole attacks. To achieve this, the authors employ the Kullback-Leibler (KL) divergence to measure the differences between the global Probability Density Functions (PDFs) across consecutive time intervals. The resulting time series derived from this function was subject to analysis based on an adaptive threshold to identify any noteworthy deviations from the norm.

More recently, Deng and Hooi [80] address the challenge of detecting anomalous events in the context of cybersecurity attacks by analyzing high-dimensional time series data, specifically sensor data. The authors propose an approach that combines structure learning with an attention-based graph neural network, which learns a graph of the dependence relationships between sensors and identifies and explains deviations from these relationships. In order to provide useful explanations for anomalies, a forecasting-based approach is used to predict the expected behavior of each sensor at each time based on the past. This allows the user to easily identify the sensors which deviate greatly from their expected behavior. Moreover, the user can compare the expected and observed behavior of each sensor, to understand why the model regards a sensor as anomalous. The approach is also highly versatile, offering extendibility for application across a wide array of use cases beyond just cybersecurity attacks.

Another concern lies in detecting anomalies in edge streams, which exhibit some specificities since they are commonly used to capture interactions in dynamic and evolutionary networks. Yu et al. [283] introduce `NetWalk`, a novel approach for real-time detection of structural anomalies in dynamic networks prevalent in fields like social media, security, and public health. Unlike traditional methods designed for static networks, `NetWalk` focuses on dynamic environments where the network representations need constant updating. The proposed approach employs a new embedding approach to encode vertices into vector representations. This process minimizes the pairwise distance between vertex representations derived from dynamic network walks and utilizes deep autoencoder reconstruction error as global regularization. The vector representations are computed efficiently using a sampling technique that requires constant memory space. Based on the learned low-dimensional vertex representations, `NetWalk` employs a clustering-based technique to detect network anomalies incrementally and in real-time as the network evolves. This approach can be applied to various types of networks, making it flexible for different application domains. In the same context, Yoon et al. [281] introduced `AnomRank` to specifically address the challenge of detecting sudden anomalous patterns, like link spam and follower boosting in dynamic graph streams by employing a unique two-pronged approach, introducing novel metrics for measuring anomalousness. These metrics track derivatives of node scores or importance functions, enabling the detection of abrupt changes in node significance. `AnomRank` demonstrates through the experiments its scalability, processing millions of edges in mere seconds, as well as theoretical soundness by providing guarantees for its two-pronged approach. `F-Fade` [58] is another approach to detect anomalies in edge streams which proposes a frequency-factorization technique to model the time-evolving distributions of interaction frequencies between node pairs. Anomalies are then detected based on the probability of the observed frequency of each incoming interaction. This approach has also proven to effectively operate in an online streaming setting while requiring constant memory.

Numerous alternative methods involve the analysis of Key Performance Indicators and metric observations to effectively identify anomalies occurring across both technical, application and functional levels. Typically, these methods employ unsupervised models that are trained on either univariate or multivariate time series data. For instance, `CloudPD` [225] is a fault management framework for clouds that introduced conventional machine learning techniques for anomaly detection in cloud environments, addressing both the physical and application layers. The approach involved utilizing various measures at the virtual machine (VM) and application machine levels, including operating system variables and application performance metrics. The paper proposed to combine three unsupervised machine learning methods: k-nearest neighbors(k-NN), HMMs, and K-means clustering. Similarly, Vallis et al. [247] addressed the challenge of detecting long-term anomalies in cloud computing environments on Twitter. The authors proposed a new statistical learning approach using the generalized Extreme Studentized Deviate test (ESD) and incorporated time series decomposition and robust statistics to identify anomalies such as a piecewise approximation of the underlying long-term trend reducing the number of false positive while taking into account both intra-day and weekly seasonalities to further minimize false positives. The proposed technique is applicable to various types of time series data, including application metrics like Tweets Per Second and system metrics like CPU utilization. On the other hand, Liu et al. [158] introduced a novel supervised approach called `Opprentice` (Operators' apprentice). In this approach, operators need to periodically label anomalies in performance data using a user-friendly tool. Then, multiple existing detectors are applied to the data in parallel to extract anomaly features. These features, along with the labeled data, are used to train a random forest classifier. This classifier automates the selection of suitable detector-parameter combinations and thresholds. In [228], the authors propose a novel approach that employs Extreme Value Theory for identifying outliers in streaming univariate KPIs. The proposed approach eliminates the need for manual threshold setting and does not assume any specific data distribution. The key parameter in this method is the risk level, which controls the rate of false positives. Apart from outlier detection, the approach can also automatically set thresholds, making it applicable to a wide range of scenarios including intrusion detection.

`Donut` [274] addresses the challenge of detecting anomalies in seasonal Key Performance Indicators (KPIs) within Web applications of large Internet companies. The authors proposed an unsupervised approach based on Variational Autoencoders (VAE) and window sampling. Importantly, the paper also introduces a novel interpretation of Kernel Density Estimation (KDE) for reconstructing anomalies in Donut, providing a solid theoretical foundation and making it the first VAE-based anomaly detection method with a well-defined theoretical explanation. In [150], the authors propose a KPI clustering-based strategy wherein millions of KPIs are grouped into a smaller number of clusters, allowing models to be selected and trained on a per-cluster basis. This approach addresses various challenges due to the unique nature of KPIs, such as their extended length and susceptibility to various distortions like noise, anomalies, phase shifts, and amplitude differences. Therefore, the paper presents a framework named `ROCKA`, which involves time series preprocessing, baseline extraction, density-based spatial clustering of applications with noise (DBSCAN) with Dynamic Time Warping (DTW) distance, and assignment steps. Ren et al. [213] introduced a sophisticated time-series anomaly detection service developed at Microsoft, emphasizing its role in continuous monitoring and incident alerting. The paper offers a detailed insight into the service's underlying pipeline and algorithm, highlighting three key modules: data ingestion, an experimentation platform, and online computation. Notably, the paper introduces a novel algorithm that combines Spectral Residual (SR) and Convolutional Neural Network (CNN) techniques on univariate KPIs.

In a slightly similar context, Zhao et al. [294] conducted a study utilizing real-world data from a prominent commercial bank to quantify the substantial impact of software changes on incidents, including code defects, configuration errors, resource conflicts, and software version discrepancies. This quantitative analysis is complemented by qualitative insights that expose limitations in current detection practices, particularly when dealing with heterogeneous multi-source data inherent in software modifications. Building upon these insights, the paper introduces an approach dubbed SCWarn that leverages multimodal learning using LSTM model to identify detrimental changes and generate interpretable alerts. The authors also quantitatively demonstrate the efficiency of their approach in reducing the mean time required to detect adverse changes. Wang et al. [256] addressed a similar problem by introducing Kontrast, a self-supervised and adaptive approach that employs contrastive learning. This approach focuses on comparing KPI time series before and after a software change to ensure the system's stability post-change. To facilitate contrastive learning, a unique data augmentation technique inspired by self-supervised learning is proposed, generating data with pseudo labels. Through the experiments conducted in the paper, the proposed model has shown impressive processing speeds and cross-dataset adaptability.

More recently, a plethora of approaches have emerged to address the challenge of anomaly detection in multivariate time series data. Two benefits of conducting a multi-dimensional analysis include the capacity to depict the interconnections among diverse metrics and the generation of understandable findings for root-cause investigation [196]. `OmniAnomaly` [233] focuses on capturing normal patterns in multivariate time series using a stochastic recurrent neural network framework, by learning robust representations through techniques like stochastic variable connection and planar normalizing flow. The approach involves reconstructing input data based on these representations and using reconstruction probabilities to identify anomalies. Additionally, `OmniAnomaly` provides insightful interpretations for detected anomalies by analyzing the reconstruction probabilities of constituent univariate time series. Multi-Scale Convolutional Recurrent Encoder-Decoder (`MSCRED`) [285] constructs multi-scale signature matrices to represent different system statuses at various time steps. It utilizes a convolutional encoder to capture inter-sensor correlations and an attention-based Convolutional Long-Short Term Memory (ConvLSTM) network to identify abnormal time steps by capturing temporal patterns. The utilization of residual signature matrices aids in anomaly detection and diagnosis. `USAD` [33] employs adversarial training in the auto-encoder architecture to effectively isolate anomalies while maintaining fast training speeds. Experimental results focus also on the scalability and robustness of the approach. Li et al. [149] introduced `InterFusion` that employs a hierarchical variational auto-encoder with two stochastic latent variables to capture normal patterns, learning low-dimensional inter-metric and temporal embeddings. The paper also presents a Markov chain Monte Carlo (MCMC) method to obtain meaningful reconstructions for interpreting anomalies.

Log-based approaches have surfaced as efficient means for detecting anomalies in application, functional, and business services. Fu et al. [93] proposed a clustering-based technique in which log entries are mapped to their corresponding template versions, enabling the identification of line templates. Subsequently, a Finite State Machine (FSM) was learned to model program workflows based on the log evidence. This FSM-based model facilitated the verification of correct program execution and the detection of software problems. In another study by [276], text analysis and information retrieval techniques were applied to console logs and source code for anomaly detection in large-scale data centers. State variables and object identifiers were automatically extracted from parsed logs, and their frequency across different documents was analyzed using PCA and term inverse-document frequency (TF-IDF). Anomalies were detected using a threshold-based rule on the reconstruction error. Lou et al. [160] were the first to explore mining invariants from log messages for system anomaly detection. They identified two types of invariants that capture relationships among different log messages: invariants in textual logs describing equivalence relations, and invariants expressed as linear equations representing linear independence relations. `Log3C` [105] presents a cascading clustering algorithm designed to efficiently detect significant system issues by utilizing both log sequences and system KPIs. This method involves iterative sampling, clustering, and matching of log sequences. Subsequently, the correlation between clusters of log sequences and system KPIs is employed to pinpoint and prioritize impactful problems through the application of a multivariate linear regression model.

Significant efforts have been directed towards exploring sequential Recurrent Neural Networks (RNNs) and their variations. `DeepLog` [85] introduced the use of LSTM networks to learn patterns from logs and predict the probability distribution of the next log key based on the observation of previous log keys. The identification of an anomalous log key occurred when it failed to appear among the top-k keys ranked by probability. Furthermore, the research introduced an online learning approach based on user feedback and tried to provide explanations using a finite-state machine. `LogAnomaly` [173] is a semantic-aware representation framework, that employed the language model method and used template embeddings (template2vec) to automatically combine similar log keys. This approach eliminated the requirement for human input. The issues of log-specific word embedding and out-of-vocabulary are both addressed. Zhang et al. [291] discovered that log data frequently include log events or sequences that have not been encountered before, indicating log unpredictability and noise in log processing. To address this issue, they introduced `LogRobust`, a method that extracts the meaning of log events using readily available word vectors. Subsequently, they employed a bidirectional LSTM model and attention mechanisms to compute an anomaly score directly, rather than predicting the next probable log keys. Xia et al. [267] proposed `logGAN`, an LSTM-based generative adversarial network framework, incorporating permutation event modeling for log-level anomaly detection. This approach addresses both sequential characteristics and out-of-order issues of logs, while the generative adversarial network component counteracts class imbalance, thereby enhancing anomaly detection performance. More recently, a novel self-supervised framework known as `LogBERT` [99] has been introduced, harnessing the power of Bidirectional Encoder Representations from Transformers (BERT). This approach introduces two innovative self-supervised training tasks: masked log message prediction and volume of hypersphere minimization. These tasks enable `LogBERT` to capture patterns inherent in normal log sequences. As a result, `LogBERT` identifies anomalies by detecting deviations from these established patterns.

5.3 Incident Prediction Methods

Incident prediction approaches are proactive methods designed to prevent failures (outages in extreme cases) by addressing both static aspects, such as source code, and dynamic aspects, such as the availability of computing resources. The ultimate objective is to suggest preventive measures or take immediate action as early as possible. These strategies vary extensively regarding the taxonomy we proposed (i.e., data used, area of application, etc.).

Software Defect Prediction. SDP is an approach used to estimate the likelihood of encountering a software bug within a functional unit of code, such as a function, class, file, or module. The core assumption linking SDP to failure occurrence is that code with defects leads to errors and failures during execution. Traditionally, defect-prone software is identified using code metrics to construct defect predictors, as discussed in 4.1. Nagappan et al. [187] propose an SDP approach based on code complexity metrics. However, they highlight the challenge of multicollinearity among these metrics, making the problem more complex. To address this, they employ PCA to obtain a reduced set of uncorrelated features and use linear regression models for post-release defect prediction. They also

Table 10. Summary of reviewed AIOps Incident Prediction methods.

			Target Area				**Data sources**												
Ref.	**Year**	**Focus**	**Technical**	**Application**	**Functional**	**Business**	**Source Code**	**Topology**	**Event Logs**	**KPIs/Metrics**	**Traffic Network**	**Reports**	**Alerts**	**Traces**	**Approach**	**Paradigm**	**Evaluation Metrics**	**Code**	**Dataset**
[187]	2006	SDP			●	●	●								PCA, Lin. Reg	SUP	Spearmann, Pearson, R^2		
[175]	2007	SDP			●	●	●								Naive Bayes	SUP	ROC		●
[87]	2008	SDP			●	●	●								SVM	SUP	F1, ACC, PREC, REC		●
[79]	2013	SDP			●	●	●								Bayesian Nets	SUP	AUC		●
[198]	2005	SDP			●	●	●								Poisson GLM	SUP	ACC		
[184]	2008	SDP			●	●	●								Ensemble Model	SUP	PREC, REC, FPR, ROC		●
[188]	2013	SDP			●	●	●								Log. Reg	Transfert	F1, PREC, REC		●
[253]	2016	SDP			●	●	●								DBN	SUP	F1, PREC, REC		●
[137]	2017	SDP			●	●	●								CNN	SUP	F1, PREC, REC		●
[169]	2020	SDP			●	●	●								LSTM	SUP	F1, PREC, REC, ACC	●	●
[275]	2021	SDP			●	●	●								LDA, GNN	SUP	F1, PREC, REC, ACC, AUC		●
[244]	2022	SDP			●	●	●								biLSTM, BERT	SUP	F1, PREC, REC		●
[95]	1998	Software Aging		●						●					Kendall-test, Lin. Reg	Forecast	Autocorrelation Plots		
[246]	1999	Software Aging		●						●					SMM, Lin. Reg, K-Means	Forecast	time to exhaustion		
[26]	2010	Software Aging	●	●						●					Lin. Reg Ensemble	Forecast	MAE		●
[235]	2014	Software Rejuv.	●	●						●					Neural Network	SUP	MAPE		
[30]	2014	Software Rejuv.	●	●						●					MLP	SUP	MAPE		
[297]	2010	Disk Fail.	●							●					HMM, HSMM	SUP	ROC		●
[257]	2013	Disk Fail.	●							●					mRMR	SUP	ROC		●
[272]	2016	Disk Fail.	●							●					RNN	SUP	REC		●
[269]	2018	Disk Fail.	●							●					Online RF	SUP	REC, FPR		●
[76]	2017	Datacenter Networks	●							●					MLP, RNN	SUP	ACC		●
[288]	2017	Switch Fail.	●						●						FT-Tree, HsMM	SUP	F1, PREC, REC		
[237]	2019	DRAM Fail.	●						●	●					TCNN	SUP	F1, PREC, REC		●
[123]	2020	Circuit Fail.	●						●	●					FFT, PCA, CNN	SUP	F1, PREC, REC, ACC		
[298]	2017	RUL	●							●					LSTM	Forecast	RMSE		●
[265]	2018	RUL	●							●					Vanilla-LSTM	Forecast	MSE		●
[166]	2020	RUL	●							●					CNN-LSTM	Forecast	RMSE, MAE		●
[42]	2019	RUL	●							●					Sequential Decision	Transfer, Reinforc.	Reward		
[142]	2020	RUL	●							●					GAN	Adversial	RMSE, MAE, MAPE		●

Ref.	Year	Focus	Target Area: Technical	Target Area: Application	Target Area: Functional	Target Area: Business	Data sources: Source Code	Data sources: Topology	Data sources: Event Logs	Data sources: KPIs/Metrics	Data sources: Traffic Network	Data sources: Reports	Data sources: Alerts	Data sources: Traces	Approach	Paradigm	Evaluation Metrics	Code	Dataset
[69]	2004	Web Apps		●						●					TAN	SUP	ACC, FPR, TPR		
[56]	2012	Datacenters		●						●					ARMA	UNSUP	RMSE, PREC, REC, FPR		
[92]	2013	Log Fail.		●					●						SVM	SUP	FPR, TPR, TNR		
[287]	2016	Log Fail.		●					●						LSTM	SUP	AUC		
[110]	2017	Cloud		●	●					●				●	LSTM	SUP	F1, ACC, PREC, TPR, TNR, FPR		●
[204]	2018	Cloud		●	●			●		●					Bayesian Nets, ARIMA	SUP	AUC, PREC, REC, FPR	●	●
[153]	2018	Cloud		●				●		●					LSTM, RF	SUP	F1, PREC, REC		
[293]	2020	Cloud		●									●		Xgboost, LIME	SUP	F1, PREC, REC		

Table 11. Study of particularities of AIOps Incident Prediction methods.

Ref.	Interpretability	Scalability	Robustness	Temporal Evaluation	Human-in-the-loop
[79, 187, 188, 198]	●		●		
[175]	●	●			
[87]			●		
[184]	●				
[253]		●	●		
[137]			●		
[169, 275]		●			
[26, 95]	●		●	●	
[246]	●			●	
[297]	●			●	
[257, 269]	●		●	●	
[272]		●		●	
[76]			●	●	
[288]	●	●		●	
[237]		●		●	
[123]		●	●	●	●
[265, 298]			●	●	
[166]				●	
[42]		●	●	●	
[142]		●		●	
[69]	●	●	●	●	●
[56]	●	●		●	●
[92]		●	●		●
[287]	●	●	●	●	
[110]		●		●	
[204]	●			●	
[153]		●			
[293]	●	●		●	

try to employ models learned from one project when testing on different projects to evaluate cross-project applicability. This yields varied outcomes, leading to the assertion that project similarity is a crucial factor for successful transfer learning. Menzies et al. [175], shift their focus from static code metrics to the choice of prediction models, advocating for the use of Naive Bayes with logarithmic features. In [87], the authors examined the effectiveness of SVM in predicting defect-prone software modules, including functions in procedural software and methods in object-oriented software. They rely on code metrics, particularly McCabe metrics[172] and Halstead metrics [101], and compare its predictive performance with eight well-known statistical and machine learning models across four NASA datasets. Dejaeger et al. [79] explored 15 different bayesian network classifiers to identify alternatives to Naive bayes that could lead to simpler networks with fewer nodes and arcs while maintaining or improving predictive performance. The study also examines the applicability of the Markov blanket principle for feature selection within the BN framework. Through the conducted evaluations, the paper demonstrates that these alternative BN classifiers can yield simpler, yet comprehensible networks with competitive predictive performance compared to the Naive Bayes classifier. The findings also emphasize the importance of balancing the interpretability and predictive power of these models.

While the previously mentioned SDP contributions mainly centered around single-release viewpoints, another set of studies, referred to as changelog approaches, emphasize the significance of software history as a more impactful element for predicting defect density. For instance, factors like code age or the count of previous defects can serve as indicators for estimating the occurrence of new bugs [96]. Ostrand et al. [198] delve into the analysis of changes within extensive software systems and their correlation with previous faults, aiming to anticipate the count of defects in upcoming releases. They employ a Poisson generalized linear model, utilizing maximum likelihood estimates of model parameters to assess the significance of diverse metrics. The model is evaluated within the release cycle of an internal inventory system, encompassing various new file-level metrics such as programming language, edit flags, and age. Findings conclude that the top 20% of files with the highest projected fault count encompassed an average of 83% of subsequently identified faults. Conducting a comparative assessment of two sets of SDP metrics (code and change metrics), Moser et al. [184] explored the Eclipse project repository. They employed three distinct machine learning methods including Naïve Bayes, logistic regression, and decision trees. The analysis demonstrated that individual usage of change metrics is more effective than relying solely on code metrics to identify defective source files. Furthermore, a combined approach shows modest enhancements or comparable results to a change metric-oriented approach.

Nam et al. [188] tackled the cross-project defect prediction challenge, specifically focusing on scenarios involving new or resource-limited projects. In such cases, the authors sought to leverage training data from established source projects and adapt it to target projects. To achieve this, they employed a cutting-edge transfer learning technique known as TCA (Transfer Component Analysis) to harmonize feature distributions across the source and target projects. Additionally, they introduced an extended version called TCA+ that incorporates these aligned features into a logistic regression model for accurate prediction of faulty module files. The approach has been evaluated on a benchmark SDP dataset AEEM proposed by D'Ambros et al. [86]. Critiques of traditional code metrics point out their handcrafted and simplistic nature. An alternative approach involves parsing the source code using ASTs. Wang et al. [253] question the ability of code metrics to capture semantics and distinguish between code regions with the same structure but different semantics. They propose using latent semantic representations and training a Deep Belief Network (DBN) on AST-parsed code to learn semantic features in an unsupervised fashion. Li et al. [137] presented DP-CNN, a novel framework called Defect Prediction via Convolutional Neural Network. This approach involves the extraction of a subset of AST nodes that capture diverse semantic operations during parsing. These nodes are then transformed into numerical features through mapping and word embedding processes. Subsequently, the transformed features are inputted into a 1D convolutional architecture, to automatically learn semantic and structural program features.

Unlike existing techniques that operate on coarse-grained units such as modules, classes, or files, Majd et al. [169] introduced a novel approach called SLDeep, which aims to identify fault-prone areas of code at a much finer-grained statement level. SLDeep defines a comprehensive suite of 32 statement-level metrics, encompassing factors like the utilization of binary and unary operators within a statement. The selected learning model for implementation is the LSTM model. The experiments were conducted on a diverse set of 119,989 C/C++ programs from the Code4Bench benchmark Majd et al. [168]. In another study, Xu et al. [275] proposed an approach for characterizing software defects using defect subtrees in ASTs. This approach incorporates information from fix-inducing changes and code concepts. Initially, a topic model is developed to summarize functional concepts related to defects. Each node within the defect subtrees is enriched with attributes such as types, fix-inducing changes, and code concepts. Subsequently, a GNN classifier is employed, where subtrees are represented as directed acyclic graph structures. More recently, Uddin et al. [244] utilized a BiLSTM model in conjunction with a BERT-based semantic feature approach (SDP-BB) to predict defects within software. This combination captures semantic features of the code by extracting contextual information from token vectors learned by the BERT model using the BiLSTM. An attention mechanism is also integrated to capture the most crucial features for prediction. This methodology is enhanced by a data augmentation technique that generates supplementary training data. The evaluation involves both within-project defect prediction and cross-project defect prediction experiments.

Software Aging and Rejuvenation. Software aging describes a phenomenon in which a software system experiences a gradual decline in performance and reliability as time passes. Recognized causes of software aging include memory leaks, bloats, unreleased file locks, data fragmentation, and the accumulation of numerical errors [54, 196]. Conversely, software rejuvenation pertains to proactive or corrective actions taken to eliminate accumulated error conditions and liberate system resources. These actions may include garbage collection, flushing kernel tables, re-initializing internal data structures, and similar strategies. Garg et al. [95] propose a method for estimating the time-to-exhaustion of various system resources, including free memory, file, and process table sizes, and used swap space. They utilize regression techniques and seasonal testing to identify trends and quantify the exhaustion time. In a related study, Vaidyanathan and Trivedi [246] explore the impact of software aging resulting from the current system workload. They develop a semi-Markov reward model based on available workload and resource data, where different workload scenarios are represented as model states. The association to a specific state is determined using k-means clustering. To estimate the time-to-exhaustion of memory and swap space, a non-parametric regression technique is employed separately for each workload state. The challenge of non-linear and piece-wise linear resource consumption is tackled by [26] by utilizing an ensemble of linear regression models. These models are selected using a decision tree based on the same input features as the regression model, which consists of a combined set of hardware and software host metrics. In the study by Sudhakar et al. [235], the authors advocate the utilization of a neural network architecture to capture intricate non-linear connections between resource usage and time to failure in cloud systems.

Hardware Failures Prediction. In large-scale computing infrastructures, ensuring hardware reliability is crucial for achieving service availability goals. However, due to the sheer number of components involved and the necessity to use commodity hardware in data centers, hardware failures pose a significant challenge. For example, Google has reported that 20-57% of disks experience at least one sector error over a 4-6 year period [178]. Hard drives are the most frequently replaced components in large cloud computing systems, and they are a leading cause of server failure [250]. To address this, hard-drive manufacturers have implemented self-monitoring technologies like

SMART metrics in their storage products. In the approach presented by Zhao et al. [297], Hidden Markov and Semi-Markov Models are used to estimate likely event sequences based on SMART metric observations from a dataset of around 300 disks (with approximately two-thirds being healthy). Two models, one trained from healthy disk sequences and the other from faulty disk sequences, are used to estimate the sequence log-likelihood at test time, with the class being determined by the highest score. Wang et al. [257] propose a similarity-based detection algorithm that selects relevant SMART features using Minimum Redundancy Maximum Relevance (mRMR) and projects the input data into a Mahalanobis space constructed from the healthy disk population. This approach aims to detect faulty disks that deviate more from the distribution. Xu et al. [272] introduce the use of RNNs to model the long-term relationships in SMART data. Unlike binary classification approaches, their model is trained to predict the health status of disks, providing additional information on the remaining useful life and serving as a ranking approach. These approaches are typically used in an online setting after an offline training step. However, integrating additional data and updating the characteristics of faulty disks as new failures occur presents a challenge. To address this, the approach presented by [269] proposes the use of online random forests, a model that can adaptively evolve with changing data distributions through online labeling.

Regarding other hardware components, FailureSim [76] presents an approach to evaluate the status of hardware in cloud data centers by employing both multi-layer perceptrons and RNNs. The proposed method concentrates on assessing 13 distinct host failure states associated with specific components such as CPU, memory, and I/O. On the other hand, addressing switch failures within datacenter networks, Zhang et al. [288] introduced a novel model termed the frequent template tree (FT-tree). This model identifies frequent word combinations within the historical system log and employs them as message templates, which are then correlated with faulty behavior. Subsequently, a Hidden Semi-Markov Model is trained using the identified templates to predict failures. Sun et al. [237] put forward a proactive hardware failure prediction scheme based on deep learning. This scheme primarily focuses on disk and memory faults, utilizing both SMART data and system logs. Overcoming the challenge of effectively handling discrete time-series data, the scheme normalizes attribute distributions from diverse vendors. The authors introduce a specialized Temporal Convolution Neural Network (TCNN) model designed to handle temporal noise, incorporating a tailored loss function for training with highly imbalanced samples. On a different note, Khalil et al. [123] introduced an approach aimed at anticipating potential hardware failures attributed to aging or varying conditions of Open circuits. The approach employs Fast Fourier Transform (FFT) to capture fault frequency signatures, utilizes Principal Component Analysis (PCA) to distill critical data with reduced dimensionality, and employs a CNN for learning and classifying faults. This noteworthy work stands out as the first to address fault prediction at the transistor level for hardware systems, encompassing aging, short-circuit, and open-circuit faults.

Remaining Useful Lifetime Estimation. The Remaining Useful Life (RUL) stands as a pivotal real-time performance indicator for operating systems throughout their operational lifespan, representing the time remaining until the system ceases to function. Similar to software aging techniques, precise RUL estimation holds significant importance in planning condition-based maintenance tasks, with the aim of minimizing system downtime. A multitude of data-driven approaches have emerged to model the intricate behavior of system components. As highlighted in [44], LSTM emerges as a highly suitable tool for handling dynamic data in RUL problems. For instance, an LSTM model presented by Zheng et al. [298] adeptly harnesses sensor sequence information to uncover latent patterns tied to various operating conditions, faults, and degradation models. Another study by Wu et al. [265] delves into a vanilla LSTM model, a prevalent variant of LSTM often applied in language processing, for predicting the RUL of aircraft engines. Ma and Mao [166] conducted a novel hybrid approach called convolution-based long short-term memory (CLSTM). This approach combines CNN with LSTM networks and is designed to predict the RUL of rotating machinery, a crucial aspect of prognostics and health management (PHM). The CLSTM architecture retains the advantages of LSTM while simultaneously integrating time-frequency features. This enables the model to capture long-term dependencies and extract time-frequency domain features concurrently. Through the stacking of multiple CLSTM layers and the creation of an encoding-forecasting architecture, a deep learning model is formulated for RUL prediction. Notably, Reinforcement Learning has also found application in enhancing model adaptability through learning from experiences, even in cases of erroneous decisions. Simulation environments prove particularly advantageous in this context. For instance, a transfer learning approach developed in [42] leverages states, actions, and rewards to generate an optimal reward policy. These algorithms are specifically geared towards sequentially predicting RUL for a specific type of pumping system. Addressing the challenges surrounding the determination of initial prediction times for remaining useful life, Li et al. [142] introduced a GAN approach that learns data distributions from healthy machine states. A health indicator is used to determine the initial prediction time, and adversarial training is employed to align data from various machines, thereby extracting generalized prognostic knowledge.

Software Failure Prediction. Predicting system failures from an application perspective involves exploring potential failures that may arise across various dimensions, including jobs, tasks, processes, VMs, containers, or nodes. Existing approaches to address this issue predominantly rely on system metrics, service states, traces, and topology. For instance, Cohen et al. [69] presented an approach centered on Tree-augmented Bayesian Networks (TANs) to establish connections between observed variables and abstract service states. This strategy facilitates the anticipation and prevention of Service Level Objective (SLO) breaches and failures in the context of three-tiered Web services. The system monitors crucial system metrics such as CPU time, disk reads, and swap space, constructing a model that captures the intricate dependencies between them. By employing a heuristic selection process, the optimal graph structure, encompassing the most relevant input metrics, is determined. An approach to forecast system availability within datacenters has been introduced in a study by [56]. This framework employs ARMA model and fault-tree analysis. The framework identifies symptoms at the component level, such as elevated CPU temperature, malfunctioning disk sectors, and memory depletion, which function as the terminal points of the fault tree. By incorporating these symptoms and the underlying tree structure, a model encompassing dependencies in combinational logic deterministically establishes the availability state. This enables the proactive detection of errors before they escalate into failures.

In [92], the authors addressed the task of predicting system failures through a log file analysis approach. This involves processing log files using the Random Indexing technique, which effectively captures intricate contextual information embedded within sequences of log messages. This ensures a nuanced representation of changes in the system's state over time. Subsequently, a weighted Support Vector Machine (SVM) approach is implemented to classify these sequences, assigning them to either impending failure scenarios or non-failure instances. Notably, the method accounts for the common challenge of imbalanced datasets, particularly emphasizing the need to maintain robust true positive rates. Similarly, the work by Zhang et al. [287] unveils a method for predicting system failures through real-time analysis of console logs. This novel approach introduces log pattern extraction via clustering, which groups together log formats and content sharing similarities. Treating log patterns as analogous to words and organizing them into discrete time intervals as documents, this approach significantly simplifies the feature space, providing valuable insights into the system's current state. These identified states are then utilized as inputs to an LSTM model, facilitating the prediction of potential failures. LSTM networks also find application in an extensive

Table 12. Summary of reviewed AIOps Incident Priorization methods.

			Target Area				Data sources												
Ref.	**Year**	**Focus**	**Technical**	**Application**	**Functional**	**Business**	**Source Code**	**Topology**	**Event Logs**	**KPIs/Metrics**	**Traffic Network**	**Reports**	**Alerts**	**Traces**	**Approach**	**Paradigm**	**Evaluation Metrics**	**Code**	**Dataset**
[243]	2013	Software Bugs			•	•		•				•			Lin. Reg	SUP	F1, PREC, REC		•
[156]	2018	N/A		•				•					•		Hierarchical Bayesian Net	SUP	ROC, PRC		
[102]	2019	IDS			•			•					•		Diffusion Model	UNSUP	ROC		
[295]	2020	N/A	•	•	•	•		•	•	•			•		XGBoost	SUP	F1, PREC, REC		
[61]	2020	Software Bugs	•	•	•	•		•				•	•		CNN	SUP	AUC, PREC, REC	•	•
[100]	2022	Software Bugs			•	•	•					•			TOPSIS, BFOA, BAR	SUP	F1, PREC, REC		•

characterization study carried out by Islam and Manivannan [110] on a workload trace dataset obtained from Google. Within this study, the focus lies on predicting failures at both the job and task levels. A job comprises multiple tasks, wherein each task corresponds to a single-machine command or program. Failures are forecasted by leveraging resource usage, performance metrics, and task particulars, encompassing completion status as well as attributes linked to users, nodes, and jobs.

Drawing on a holistic strategy, the `HORA` prediction system [204] capitalizes on architectural knowledge along with live KPIs data to forecast potential Quality of Service (QoS) violations and service disruptions within distributed software systems. Central to this approach are Bayesian Networks, which play a pivotal role in constructing models that depict component interdependencies and the propagation of failures. These models establish connections between anticipated component failures, derived through auto-regressive predictors from system metrics, and larger-scale systemic challenges. Lin et al. [153] introduced an approach called MING to predict potential node failures within cloud service systems, which integrates LSTM and Random Forest models to incorporate both temporal and spatial data. Additionally, the technique employs a ranking model and a cost-sensitive function to effectively assess and rank nodes based on their susceptibility to failure. Demonstrating its practical effectiveness, the approach finds successful application in industrial scenarios, notably within the context of Microsoft services. `eWarn` [293] is an approach tailored for online service systems that capitalizes on historical data and real-time alert information to forecast the likelihood of upcoming incidents. This is achieved through a combination of innovative techniques: effective feature engineering to represent pertinent alert patterns, integration of multi-instance learning to mitigate the influence of irrelevant alerts, and the generation of interpretable prediction reports using the LIME explanation technique.

5.4 Incident Prioritization Methods

As a large number of incidents can be reported simultaneously, it becomes time-consuming and resource-intensive to handle all of them at once. However, certain incidents require immediate attention due to their importance or severity. To address this issue, various data-driven approaches have been proposed to rank incidents or alerts based on prioritization factors. Some of these

Table 13. Study of particularities of AIOps Incident Prioritization methods.

Ref.	Interpretability	Scalability	Robustness	Temporal Evaluation	Human-in-the-loop
[243]	●				
[156]	●	●		●	
[102, 295]	●	●	●	●	●
[61]		●	●	●	●
[100]	●		●		

techniques can also be applied to other scenarios, as they are not solely dedicated to ranking, but also involve detection or diagnosis mechanisms.

In their work, Tian et al. [243] introduced the DRONE framework, which provides recommendations for prioritizing bug reports. This is achieved by considering a multitude of factors, including temporal information, textual content, author details, related reports, severity, and product information. These factors are extracted as features from incident reports and subsequently used to train a discriminative model adept at handling ordinal class labels and imbalanced data. The model employs linear regression to capture the connection between the features and priority levels, followed by a thresholding approach to calibrate class labels (i.e., priority levels). In response to the challenge of characterizing and ranking diverse categorical alerts from an anomaly detection system, Lin et al. [156] introduced Collaborative Alert Ranking (CAR). CAR addresses both mining alert patterns and reducing false positives. Initially, a hierarchical Bayesian model is constructed to capture short-term and long-term dependencies within alert sequences. An entity embedding-based model is then devised to uncover content correlations among alerts based on their categorical attributes. By integrating temporal and content dependencies within a unified optimization framework, CAR provides rankings for individual alerts as well as their associated patterns. Similarly, aiming to mitigate threat alert fatigue and prioritize true attacks over false alarms, Hassan et al. [102] introduced NODOZE. This ranking system constructs causal dependency graphs for alert events using contextual and historical information. Anomaly scores are assigned to edges based on event frequency and are propagated using a novel network diffusion algorithm.

AlertRank, proposed by Zhao et al. [295], is an automatic and adaptive framework for identifying severe alerts. This approach leverages a variety of interpretable features, including textual and temporal alert features, as well as domain knowledge-based univariate and multivariate KPIs. The XGBoost ranking algorithm is employed to identify severe alerts, with continuous labeling utilized to obtain ground-truth labels for training and testing. Chen et al. [61] conducted a large-scale empirical analysis of incidents from real-world online service systems. Their findings highlight a category of incidents labeled as *Incidental Incidents*, often considered less significant and not prioritized for immediate resolution. To address this, the authors proposed *DeepIP* (Deep learning-based Incident Prioritization), which employs an attention-based CNN to identify and prioritize incidental incidents using historical incident descriptions and topology information. On a distinct note, Gupta et al. [100] adopted a unique approach by combining the multi-criteria fuzzy Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) with the Bacterial Foraging Optimization Algorithm (BFOA) and Bar Systems (BAR) optimization. This amalgamation serves to rank software bugs and simultaneously select suitable developers.

5.5 Incident Assignment Methods

Numerous data-driven approaches have been proposed to optimize the process of incident assignment (referred also to as routing, or triaging) by automatically assigning incidents to the

Table 14. Summary of reviewed AIOps Incident Assignment methods.

			Target Area				Data sources												
Ref.	**Year**	**Focus**	**Technical**	**Application**	**Functional**	**Business**	**Source Code**	**Topology**	**Event Logs**	**KPIs/Metrics**	**Traffic Network**	**Reports**	**Alerts**	**Traces**	**Approach**	**Paradigm**	**Evaluation Metrics**	**Code**	**Dataset**
[186]	2004	Software Bugs			•	•						•			Bayesian Net	SUP	ACC		•
[224]	2008	N/A	•	•	•	•								•	Markov Model	SUP	ACC		
[278]	2010	Software Bugs			•	•						•			Bayesian Net, EM	SemiSUP	ACC		•
[45]	2010	Software Bugs			•	•						•		•	Naive Bayes	SUP	ACC		•
[24]	2013	Software Bugs			•	•						•			Naive Bayes	SUP	F1, PREC, REC		•
[254]	2014	Software Bugs			•	•						•		•	Similarity Computation	UNSUP	ACC, Diversity		•
[266]	2019	Software Bugs			•	•						•		•	GRU	SUP	ACC		•
[132]	2017	Software Bugs			•	•						•			CNN	SUP	ACC		
[60]	2019	N/A	•	•	•	•		•				•			CNN, GRU	SUP	ACC		
[202]	2020	N/A	•	•	•	•		•				•			CNN	SUP	ACC		
[212]	2021	N/A	•	•	•	•		•		•		•	•		Subgroup Discovery	SUP	F1, Fidelity	•	

Table 15. Study of particularities of AIOps Incident Assignment methods.

Ref.	Interpretability	Scalability	Robustness	Temporal Evaluation	Human-in-the-loop
[186]	•				
[212, 224]	•	•	•		
[278]	•			•	
[45]	•	•	•	•	•
[24]	•	•			
[254]	•	•		•	
[266]		•		•	•
[132, 202]		•			
[60]		•	•	•	

appropriate service team and individual. Typically, these approaches involve training a classifier using historical incident reports that contain textual information, topology data, or prioritization scores. The trained classifier is then used to assign new incidents.

The preceding works have primarily relied on text preprocessing methods and supervised classification models within traditional machine learning and statistical frameworks. For instance, Murphy and Cubranic [186] employed a Bayesian learning approach to incident reports, representing them as a bag of words from a predefined vocabulary, which conveniently adapts to multi-class classification. However, this method's accuracy in predicting report assignments to developers was found to be limited, achieving only 30% correct predictions. In [224], a unique perspective is taken by solely utilizing incident resolution sequences without accessing the incident ticket content. This is achieved by developing a Markov model that captures the decision-making process behind successful problem resolution paths. The order of the model is thoughtfully chosen based on conditional entropy derived from ticket data. Addressing the challenge of limited labeled incident reports, Xuan et al. [278] presented a semi-supervised text classification technique. They combine a Naive Bayes classifier with expectation-maximization, leveraging both labeled and unlabeled incident reports. An iterative process involves initial classifier training with labeled bug reports, followed by iterative labeling of unlabeled bug reports to enhance classifier training. A weighted recommendation list further refines performance, utilizing multiple developer weights during classifier training.

Bhattacharya and Neamtiu [45] tackled the task by employing various techniques, including refined classification with additional attributes, intra-fold updates during training, a precise ranking function for recommending developers in tossing graphs, and multi-feature tossing graphs. Results not only exhibit a noteworthy accuracy but also demonstrate a substantial reduction in tossing path lengths. Similarly, Alenezi et al. [24] employed a series of selection term techniques to allocate software bugs to experienced developers. The process is initiated with the application of a conventional text processing methodology, transforming textual data into a coherent and meaningful representation. Subsequently, a bug-term matrix was crafted, incorporating term frequency weights. Various term selection methods were then employed to effectively mitigate both data dimensionality and sparsity issues. These included techniques like Log Odds Ratio, Term Frequency Relevance Frequency, and Mutual Information. Following this, a Naive Bayes classifier was trained on the resulting representations. Wang et al. [254] introduced `FixerCache`, an unsupervised approach that leverages developers' component-specific activeness to create dynamic developer caches for each product component. When a new bug report arises, `FixerCache` recommends highly active fixers from the developer cache to participate in resolving the bug. The developer cache is dynamically updated following the verification and resolution of each bug report. Xi et al. [266] emphasized the role of tossing sequence paths in predicting suitable service teams or individuals for handling software bugs. Their proposal, iTriage, employs a sequence-to-sequence model to jointly capture features from textual content and tossing sequences, integrating them through an encoder-decoder classification model.

In recent times, the focus has shifted towards advanced natural language processing techniques combined with sophisticated deep learning approaches. For example, Lee et al. [132] pioneered the use of a convolutional neural network and pre-trained Word2Vec embeddings for incident assignment. `DeepCT` [60] presents an approach that views incident triage as a continuous process involving intensive discussions among engineers. It employs a GRU-based model with an attention-based mask strategy and a revised loss function to learn knowledge from discussions and update triage results incrementally. `DeepTriage` [202] was devised to address incident assignment challenges such as imbalanced incident distribution, diverse input data formats, scalability, and building engineers' trust. This approach employs CNN models to recommend the responsible team for each incident. Focusing on interpretability, Remil et al. [212] proposed a framework that includes preprocessing steps like lemmatization using a pre-trained transformer model. Incident textual information is vectorized using TF-IDF and fed into an LSTM-based attention model for predicting the suitable service team. Furthermore, a Subgroup Discovery approach groups predicted incident tickets and labels into subgroups supporting the same explanation of the model's decision-making process.

5.6 Incident Classification Methods

As discussed earlier, the primary objective of incident classification is to enhance the diagnosis of incidents, thereby centralizing the efforts of the maintenance team. However, this category

Table 16. Summary of reviewed AIOps Incident Classification methods.

			Target Area				Data sources												
Ref.	**Year**	**Focus**	**Technical**	**Application**	**Functional**	**Business**	**Source Code**	**Topology**	**Event Logs**	**KPIs/Metrics**	**Traffic Network**	**Reports**	**Alerts**	**Traces**	**Approach**	**Paradigm**	**Evaluation Metrics**	**Code**	**Dataset**
[280]	2014	Software Bugs			•	•						•			LDA	UNSUP	F1, PREC, REC, MRR		•
[151]	2014	N/A		•						•					HMRF	UNSUP	PREC, REC		•
[268]	2016	Software Bugs			•	•		•				•			MTM	UNSUP	ACC		•
[284]	2017	N/A	•	•	•	•						•		•	Hierarchical Classification	SUP	F1, PREC, REC		
[292]	2020	N/A	•	•				•					•		EVT, DBSCAN	UNSUP	F1, PREC, REC		

Table 17. Study of particularities of AIOps Incident Classification methods.

Ref.	Interpretability	Scalability	Robustness	Temporal Evaluation	Human-in-the-loop
[280]	•		•		
[151]	•	•		•	
[268]	•	•	•	•	
[284]	•	•			•
[292]	•	•	•		

has often been overlooked in the review process and is generally associated with deduplication, triage, or prioritization. Nevertheless, there exist few research works that align perfectly with this category, offering approaches to organize a large volume of incidents into representative sets of issues or topics.

Yang et al. [280] employed LDA to extract topics from bug reports and identify relevant bug reports corresponding to each topic. Their methodology involves initially determining the topics of a new bug report and subsequently employing multiple features (such as component, product, priority, and severity) to identify reports with shared features as the new bug report. Another notable contribution, presented by [151], proposes a Hidden Markov Random Field (HMRF) based approach for automatically identifying recurrent performance issues in large-scale software systems. Their approach formulates the problem as an HMRF-based clustering problem, which involves learning metric discretization thresholds and optimizing the clustering process. Xia et al. [268] introduced an innovative bug-triaging framework utilizing a specialized topic modeling algorithm known as a multi-feature topic model (MTM). MTM extends LDA by incorporating product and component information from bug reports, effectively mapping the term space to the topic space. Additionally, they introduced an incremental learning method named `TopicMiner`, which exploits the topic distribution of a new bug report to assign an appropriate fixer based on their affinity with the topics.

In another context, Zeng et al. [284] developed a methodology to identify the underlying categories of IT problems from ticket descriptions. To achieve this, they employed a hierarchical multi-label classification technique for categorizing monitoring tickets, introducing a novel contextual hierarchy. A new greedy algorithm, `GLabel`, is employed to address the optimization challenge.

Table 18. Summary of reviewed AIOps Incident Deduplication methods.

Ref.	Year	Focus	Target Area				Data sources								Approach	Paradigm	Evaluation Metrics	Code	Dataset
			Technical	Application	Functional	Business	Source Code	Topology	Event Logs	KPIs/Metrics	Traffic Network	Reports	Alerts	Traces					
[107]	2006	Software Bugs			•	•						•			K-Means	UNSUP	F1, PREC, REC		•
[218]	2007	N/A	•	•	•	•		•				•			Cosine Similarity	UNSUP	RR@k		
[239]	2010	Software Bugs			•	•						•			N-gram similarity	UNSUP	RR@k		•
[236]	2011	Software Bugs			•	•		•				•			BM25F Measure	SUP	RR@k		•
[36]	2012	Software Bugs			•	•						•			LCS Measure	UNSUP	RR@k		•
[299]	2012	Software Bugs			•	•		•				•			LTR	SemiSUP	MRR, RR@k		•
[271]	2018	Software Bugs		•	•	•						•			CNN	SUP	F1, ACC		•
[103]	2020	Software Bugs		•	•	•						•			BERT	SUP	F1, ACC, AUC		•
[176]	2022	Software Bugs		•	•							•			BERT, MLP	SUP	F1, PREC, REC		•

Table 19. Study of particularities of AIOps Incident Assignment methods.

Ref.	Interpretability	Scalability	Robustness	Temporal Evaluation
[107, 218, 236]	•	•		•
[239]	•		•	•
[36, 299]	•			•

The approach also leverages domain expert knowledge alongside ticket instances to guide the hierarchical multi-label classification process. In addressing the issue of alert storms, Zhao et al. [292] presented a two-stage approach: alert storm detection and alert storm summary. During the alert storm summary phase, an alert denoising method filters out irrelevant alerts by learning patterns from the system's normal states. Subsequently, alerts indicating service failures are clustered based on textual and topological similarities. From each cluster, the most representative alert is selected, thereby forming a concise set of alerts for further investigation.

5.7 Incident Deduplication Methods

Incident deduplication aims to identify the most similar incidents among a set of historical incidents, which exhibit slight differences but primarily address the same problem. This work can be categorized into two main categories. The first category of research is centered around techniques for detecting duplicate incident reports based on their descriptions and characteristics.

In [107], an initial attempt was made to identify duplicate bug reports using textual information. Their approach involved building a model that clustered similar reports, represented as document vectors via TF-IDF, into representative centroids. When a new incident report is submitted, the method calculates cosine similarity to each centroid, looking for instances of high similarity to identify potential duplicates. Runeson et al. [218] adopted a similar approach, enhancing it by considering additional textual features such as software versions, testers, and submission dates. Effective preprocessing steps like stemming and stop word removal were also incorporated. Sureka and Jalote [239] proposed a distinctive N-gram-based model, setting it apart from the previous word-based methods. This study explored the utility of low-level character features, offering benefits like resilience against noisy data and effective handling of domain-specific term variations. Incorporating a comprehensive similarity assessment, Sun et al. [236] introduced a retrieval function (REP). This function considered textual content similarity in summary and description fields as well as the similarity of non-textual attributes such as product, component, and version. The paper extended the frequently used BM25F similarity measure in information retrieval.

Recognizing the limitations of word-based approaches that may identify reports discussing different problems as duplicates due to shared common words, Banerjee et al. [36] introduced `FactorLCS` approach that employed common sequence matching when calculating the textual similarity between incident reports. Zhou and Zhang [299] introduced `BugSim`, leveraging a learning-to-rank concept by incorporating both textual and statistical features. A similarity function for bug reports was established based on these features. The model was trained using a set of duplicate and non-duplicate report pairs, adjusting feature weights with the stochastic gradient descent algorithm. Candidate duplicate reports for a new bug report were retrieved using this trained model.

Recently, deep learning techniques have gained traction. Xie et al. [271] introduced `DBR-CNN`, utilizing a CNN model and word embedding techniques to assess semantic similarities between pairs of incident reports. This approach departed from prior methods that primarily relied on common words or sequences of words for lexical similarity computation. Similarly, He et al. [103] incorporated a CNN while introducing a unique bug report pair representation known as the dual-channel matrix. This matrix was formed by concatenating two single-channel matrices, each representing a bug report. These pairs were then fed into a CNN model named `DC-CNN`, designed to capture interconnected semantic relationships. In Messaoud et al. [176], BERT-MLP was proposed. This approach utilized a pre-trained language model, BERT, to process unstructured bug report data and capture contextual relationships. The BERT model's output was then input to a multilayer perceptron classifier to predict duplicate bug reports.

The second category primarily relies on designing similarity metrics that can reflect the semantic crash similarity between execution reports, specifically for stack traces, which we discuss briefly. Lerch and Mezini [135] employed the TF-IDF-based scoring function from Lucene library [16]. Sabor et al. [219] proposed DURFEX system which uses the package name of the subroutines and then segment the resulting stack traces into N-grams to compare them using the Cosine similarity. Some alternative techniques propose to compute the similarity using derivatives of the Needleman-Wunsch algorithm [192]. In [50], the authors suggested adjusting the similarity based on the frequency and the position of the matched subroutines. Dang et al. [74] proposed a new similarity measure called PDM in their framework Rebucket to compute the similarity based on the offset distance between the matched frames and the distance to the top frame. More recently, TraceSim [248] has been proposed to take into consideration both the frame position and its global inverse frequency. Moroo et al. [183] present an approach that combines TF-IDF coefficient with PDM. Finally, we outline some earlier approaches that used edit distance, as it is equivalent to optimal global alignment [39, 181].

5.8 Root Cause Analysis Methods

Root cause analysis approaches aim to determine the underlying faults that give rise to software bugs, errors, anomalies, or hardware failures. More concretely, root cause analysis is a diagnostic task that needs to be performed when reporting incidents, either after or in parallel with the triage

Table 20. Summary of reviewed AIOps Root Cause Analysis methods.

			Target Area				Data sources												
Ref.	**Year**	**Focus**	**Technical**	**Application**	**Functional**	**Business**	**Source Code**	**Topology**	**Event Logs**	**KPIs/Metrics**	**Traffic Network**	**Reports**	**Alerts**	**Traces**	**Approach**	**Paradigm**	**Evaluation Metrics**	**Code**	**Dataset**
[35]	2007	Network Faults	•					•		•	•				Inference Graph	UNSUP	Qualitative Eval.		
[31]	2012	Software Faults		•	•		•			•					DIFT	UNSUP	FPR		
[194]	2013	Cloud	•	•				•		•					Makrov Model	UNSUP	PRC		
[62]	2014	Cloud		•				•		•					Causality Graph	UNSUP	PREC, REC		
[154]	2016	Cloud		•				•		•					FPM	UNSUP	F1, PREC, REC		
[238]	2018	Cloud		•				•		•					MCTS	UNSUP	F1, PREC, REC		
[147]	2019	Cloud		•				•	•	•					FPM	UNSUP	F1	•	•
[221]	2019	Cluster		•						•					HHMM	UNSUP	REC, FNR, FPR		•
[115]	2019	Datacenters	•							•					Causality Graph	UNSUP	Ranking ACC, Success rate		
[167]	2020	Databases		•	•					•				•	Bayesian Net	UNSUP	F1, PREC, REC		
[210]	2021	Databases		•	•			•		•			•	•	Subgroup Discovery	SUP	PREC, Confidence	•	•
[148]	2022	Cloud	•	•				•		•					CNN-GRU	SUP	MAR, RR@k	•	•
[139]	2022	Cloud		•				•		•					Bayesian Net	UNSUP	RR@k	•	•
[215]	2003	SFL			•	•	•								K-NN	SUP	Score_U, Score_{NN}		
[259]	2013	SFL			•	•	•								Suspiciousness Score	SUP	EXAM, Wilcoxon Signed-Rank Test	•	•
[279]	2014	SFL			•	•	•								RankBoost	SUP	Wasted effort		•
[157]	2006	SFL			•	•	•								Statistical Debugging	SUP	T-Score		•
[22]	2009	SFL			•	•	•								Bayesian Net	SUP	Wasted effort	•	•
[81]	2005	SFL			•	•	•								Association Rules	SUP	–		
[55]	2008	SFL			•	•	•								FCA	SUP	–		
[289]	2014	SFL			•	•	•								Markov Logic	SUP	ACC		•
[229]	2017	SFL			•	•	•								Genetic Prog. SVM	SUP	ACC, Waster Effort, PREC		
[141]	2019	SFL			•	•	•								RNN, MLP	SUP	MAR, RR@k, MFR	•	•
[144]	2021	SFL			•	•	•								CNN-GRU	SUP	MAR, RR@k, MFR	•	•

process. In complex systems, it is necessary to first isolate and restrict the analysis to the faulty component or functionality, a process known as Fault Localization in many research communities. Specifically, Fault Localization involves identifying a set of components (devices, hosts, software modules, etc.) that serve as the initial trigger for an error within the system. It is important to note that fault localization can operate at different layers, including the technical, application, functional, and business layers. To simplify, we can distinguish between two types of fault localization. The first is technical and applicative fault localization or troubleshooting, which spans network, hardware, and application layers (e.g., faulty components in cloud environments). The second is software fault localization, which revolves around the functional and business layers, essentially addressing flaws

Table 21. Study of particularities of AIOps Root Cause Analysis methods.

Ref.	Interpretability	Scalability	Robustness	Temporal Evaluation	Human-in-the-loop
[35, 147, 194, 238]	●	●	●		
[31]	●	●	●	●	
[154]	●		●		●
[210, 221]	●	●			●
[62, 115, 139, 148, 167]	●	●	●		●
[55, 81, 215, 229, 259]	●				
[22, 279, 289]	●	●			
[157]	●	●	●		
[141]		●			

that lead to bugs. In the context of software fault localization, the central focus lies in analyzing the source code, whether the software is deployed across numerous machines or a more limited set.

Technical and Application Troubleshooting. The study conducted by Bahl et al. [35] introduced the `Sherlock` approach, which focuses on localizing performance issues within enterprise networks. This is achieved by constructing probabilistic inference graphs based on packet exchange observations in the network infrastructure. These graphs consist of three node types: root cause nodes (representing internal IP entities), observation nodes (representing clients), and meta-nodes that capture dependencies between these types. Each node is associated with a categorical random variable indicating its state (up, troubled, down). The state is influenced by other nodes through probability dependencies. The learning process of the inference graph involves monitoring packet exchanges between nodes during regular network operations. Once established, the graph enables the utilization of data from observation nodes to derive sets of state-node assignment vectors, reflecting the anticipated network operational state. `X-Ray` [31] addresses the challenge of troubleshooting performance issues by providing insights into the reasons behind specific events during performance anomalies. The key contribution is a performance summarization technique, achieved by instrumenting binaries while applications execute. This technique attributes performance costs to each basic block and utilizes dynamic information flow tracking to estimate the likelihood that a block was executed due to each potential root cause. The overall cost of each potential root cause is then summarized by aggregating the per-block cost multiplied by the cause-specific likelihood across all basic blocks. This technique can also be differentially applied to explain performance differences between two similar activities.

`FChain` [194] is a fault localization system designed for identifying faulty components in online cloud environments. It operates as a black-box solution, leveraging low-level system metrics to detect performance anomalies. Anomalies are sorted based on manifestation time, and a discrete Markov model is used to sequentially examine these components. Techniques such as analyzing interdependencies between components and studying the propagation trend are employed to filter out spurious correlations. Similarly, `CauseInfer` [62] is a black-box cause inference system that constructs a two-layered hierarchical causality graph to aid in identifying performance problem causes. The system employs statistical methods, including a novel Bayesian change point detection method, to infer potential causes along the causal paths present in the graph.

Lin et al. [155] proposed `LogCluster`, where a knowledge base is used to expedite the retrieval of logs associated with recurring issues, thus streamlining the process of problem identification. The methodology treats logs as sequences consisting of discrete log events, which are then subjected to an initial transformation into a vector format through the application of Inverse Document

Frequency (IDF), where these individual events are considered as terms. Following this, an agglomerative hierarchical clustering technique is employed to group these logs, resulting in the selection of representative instances for each cluster. During the system's operational phase, queries are directed towards this condensed set of representatives, significantly alleviating the effort required for log retrieval based on similarity.

Lin et al. [154] proposed iDice, a pattern mining approach based on frequent pattern mining and change detection within time series data. The approach is applied to identify effective attribute combinations associated with emerging issues in topology data. Given a volume of customer issue reports over time, the goal is to identify an attribute combination that divides the multi-dimensional time series dataset into two partitions: one with a significant increase in issue volume and the other without such an increase. Similarly, Sun et al. [238] introduced HotSpot, which employs Monte Carlo Tree Search (MCTS) to efficiently explore attribute combinations and measure their correlation with sudden changes in the Page View metric. The approach proposes a potential score based on the ripple effect, quantifying how anomalies propagate through different attribute combinations. A hierarchical pruning strategy is used to narrow down the search space, focusing on attribute combinations with the highest potential to be the root cause. Another akin approach, Squeeze by Li et al. [147], applies a pattern mining approach to structured logs for identifying attribute combinations responsible for abnormal behaviors. Squeeze utilizes a bottom-up, then top-down searching strategy, efficiently exploring the multi-dimensional search space by first identifying potentially relevant attributes and then pinpointing the root cause.

Another work conducted by [221] employed Hierarchical Hidden Markov Models (HHMM) to associate resource anomalies with root causes in clustered resource environments. Markov models are constructed on different levels and trained with the Baum-Welch algorithm using response time sequences as observations. Jeyakumar et al. [115] introduced ExplainIt, an approach for unsupervised root-cause analysis in complex systems like data centers, utilizing KPIs data. This system enables operators to articulate causal hypotheses systematically ranked to identify root causes behind significant events. Using a declarative language akin to SQL, ExplainIt facilitates generating hypotheses that probe the complex probabilistic graphical causal model in the system.

The iSQUAD framework, introduced by Ma et al. [167], addresses the detection and diagnosis of Intermittent Slow Queries (iSQs) in cloud databases. These queries, arising from external intermittent performance issues, can pose significant risks to users. Unlike typical slow queries, iSQs are less predictable and present more intricate diagnostic challenges. Employing a machine learning-based approach, iSQUAD leverages Anomaly Extraction, Dependency Cleansing, Type-Oriented Pattern Integration Clustering (TOPIC), and Bayesian Case Model components. In their work, Remil et al. [210] focused on SQL workload analysis for identifying schema issues and automatically pinpointing subsets of queries that share specific properties while targeting certain performance measures. These measures encompass slow execution times, concurrency issues, high I/O communications, and more. The approach involves parsing queries to extract key attributes (such as tables, fields, and predicates), and augmenting queries with pertinent information such as performance metrics, environmental features, and anomaly alerts. Utilizing a pattern mining approach dubbed Subgroup Discovery, the authors uncover a subset of queries exhibiting anomalies within patterns formed by conjunctions of conditions on these attributes. Furthermore, the integration of a visual tool allows iterative and interactive learning from the obtained outcomes.

The proposed methodology by Li et al. [148] introduces DejaVu, an interpretable approach designed to localize recurring failures within online service systems. These recurring failures manifest as repeated instances of the same type across various locations. Engineers can identify indicative metrics for each failure type, aiding in recognizing underlying issues and guiding mitigation actions based on domain expertise. These sets of indicative metrics facilitate the identification of candidate failure units associated with recurring issues. To capture intricate system dependencies, a failure dependency graph (FDG) is established, linking failure units with interdependencies. When a failure occurs, the monitoring system triggers `DejaVu`, which is trained on historical failures. DejaVu then utilizes the latest FDG and metric data to recommend suspicious failure units from the candidate set on the FDG, streamlining the fault localization process. Li et al. [139] proposed Causal Inference-based Root Cause Analysis (`CIRCA`), which frames the root cause analysis challenge as intervention recognition within a novel causal inference task. The core principle centers around a sufficient condition for monitoring variables to serve as root cause indicators. This involves a probability distribution change conditioned on parents within a Causal Bayesian Network (CBN). Specifically tailored to online service systems, CIRCA constructs a graph of monitoring metrics based on system architecture knowledge and a set of causal assumptions.

Software Fault Localization. A conventional software fault localization strategy generally yields a collection of statements or source code blocks that could potentially be linked to bugs. Unlike the SDP strategy, this approach relies on failure patterns observed during production runs and unit tests, rather than predictions regarding the probability of a code component transitioning into a faulty state. Numerous approaches have been developed to tackle the problem of software fault localization. One prominent category is program spectrum-based techniques, which rely on the similarity between program execution profiles obtained from execution traces. These profiles represent both successful and faulty runs of programs. For example, Renieres and Reiss [215] employs nearest neighbor search to compare a failed test with a similar successful test, using the Hamming distance as a measure of similarity. Another well-known technique, `Tarantula` [117], utilizes coverage and execution results to compute the suspiciousness score of each statement. This score is based on the number of failed and successful test cases covering the statement, as well as the total number of successful and failed test cases. Subsequent research in this field has proposed refinements to the suspiciousness scoring also known as ranking metrics, such as `Ochiai` [21], `Crosstab` [260] and `DStar` [259]. Xuan and Monperrus [279] addressed the challenge of finding an optimal ranking metric for fault identification by combining multiple existing metrics. The approach dubbed MULTRIC employs a two-phase process involving learning and ranking: it learns from faulty and non-faulty code elements to build a ranking model and then applies this model to compute the suspiciousness score for program entities when new faults emerge. Statistical debugging approaches have also been explored. Liu et al. [157] introduce a statistical debugging method called `SOBER`, which analyzes predicate evaluations in failing and passing runs. By estimating the conditional probability of observing a failure given the observation of a specific predicate, the approach identifies predicates with higher probabilities, indicating their potential involvement in software bugs or their proximity to them. Abreu et al. [22] later proposes a Bayesian reasoning approach known as `BARINEL`, which incorporates a probabilistic framework for estimating the health probability of components. This model, based on propositional logic, captures the interaction between successful and failed components. Furthermore, data mining techniques have shown promise in fault localization due to their ability to unveil hidden patterns in large data samples.

Denmat et al. [81] introduced an approach that reinterprets Tarantula as a data-mining challenge. In this method, association rules denoting the connection between an individual statement and a program failure are extracted using coverage information and test suite execution outcomes. The significance of these rules is assessed through two well-known classical data mining metrics, namely, *confidence* and *lift*. These values can be understood as indicators of the potential for a statement to be suspicious, hinting at the presence of bugs. Cellier et al. [55] discussed a data mining approach that relies on a combination of association rules and Formal Concept Analysis (FCA) as a mean to

Table 22. Summary of reviewed AIOps Incident Correlation methods.

			Target Area				Data sources													
Ref.	**Year**	**Focus**	**Technical**	**Application**	**Functional**	**Business**	**Source Code**	**Topology**	**Event Logs**	**KPIs/Metrics**	**Traffic Network**	**Reports**	**Alerts**	**Traces**	**Approach**	**Paradigm**	**Evaluation Metrics**	**Code**	**Dataset**
[163]	2014	N/A		•						•			•		K-NN	UNSUP	F1		
[234]	2019	N/A		•						•					Cross-Correlation	UNSUP	F1, PREC, REC, PRC		
[240]	2013	IDS	•								•				Mahalanobis Distance	UNSUP	REC, FPR, ACC		•
[40]	2013	IDS	•								•				CART	UNSUP	Completeness, Soundness, FCR		•

assist in fault localization. This technique aims to identify rules that associate statement coverage with corresponding execution failures, measuring the frequency of each rule. A threshold is set to determine the minimum number of failed executions covered by a selected rule. The generated rules are then partially ranked using a rule lattice, and the ranking is examined to locate the fault. In [289], the authors proposed an approach that utilizes multi-relational data mining techniques for fault localization. Specifically, this technique utilizes Markov logic, combining first-order logic and Markov random fields with weighted satisfiability testing for efficient inference, alongside a voted perceptron algorithm for discriminative learning. When applied to fault localization, Markov logic integrates various information sources including statement coverage, static program structure details, and prior bug knowledge to enhance the accuracy of fault localization efforts.

Recently, there has been a notable shift towards employing machine and deep learning approaches to tackle the software fault localization problem. For instance, Sohn and Yoo [229] extended spectrum-Based Fault Localization with code and change metrics, to enhance fault localization precision. They applied Genetic Programming (GP) and linear rank Support Vector Machines for learning to rank, utilizing both suspiciousness values and additional metrics. Li et al. [141] introduced `DeepFL`, a deep learning approach designed for learning-based fault localization. It addresses challenges posed by increasing feature dimensions in advanced fault localization techniques, by automatically identifying effective existing and latent features. `DeepFL` is demonstrated using suspiciousness-value-based, fault-proneness-based, and textual-similarity-based features collected from fault localization, defect prediction, and information retrieval domains. `DeepRL4FL` [144] is another deep learning approach that treats fault localization as an image pattern recognition problem. It locates buggy code at statement and method levels by introducing code coverage representation learning and data dependencies representation learning for program statements. These dynamic information types in a code coverage matrix are combined with static code representation learning of suspicious source code. Inspired by crime scene investigation, the approach simulates analyzing crime scenes (failed test cases and statements), related individuals (statements with dependencies), and usual suspects (similar buggy code). `DeepRL4FL` organizes test cases, marks error-exhibiting statements, and leverages data dependencies for comprehensive fault identification. A Convolutional Neural Network classifier is employed, utilizing fused vector representations from multiple sources to detect faulty statements/methods effectively.

Table 23. Study of particularities of AIOps Incident Correlation methods.

Ref.	Interpretability	Scalability	Robustness
[40, 163]	●	●	
[234, 240]	●	●	●

Table 24. Summary of reviewed AIOps Incident Mitigation methods.

			Target Area				Data sources												
Ref.	Year	Focus	Technical	Application	Functional	Business	Source Code	Topology	Event Logs	KPIs/Metrics	Traffic Network	Reports	Alerts	Traces	Approach	Paradigm	Evaluation Metrics	Code	Dataset
[300]	2016	-	●	●	●	●		●				●			K-NN, LDA	UNSUP	ACC, MAP, avgSim		
[252]	2017	-	●	●	●	●		●				●			Ontology Model	SUP	F1, PREC, REC		
[152]	2018	-	●	●	●	●		●	●			●			FastText	UNSUP	–		
[84]	2012	-		●	●				●					●	FCA	UNSUP	REC, ROC		

5.9 Incident Correlation Methods

Incident correlation research typically concentrates on analyzing correlations among alerting signals, occurring incidents, or the associations between alerting signals and incidents. Existing correlation algorithms primarily evaluate raw key performance indicators, or they transform these KPIs into events and then analyze their correlations.

Luo et al. [163] focused on correlating events with KPIs data. They formulated the correlation problem as a two-sample problem to assess the correlation between KPI time series and event sequences in online service systems. They employ the nearest neighbors method to evaluate the existence of the correlation and analyze temporal relationships and monotonic effects. `CoFlux` [234] is an unsupervised approach for correlating KPIs in internet service operations management. It introduced the concept of KPI flux-correlation, which involves identifying interactions between KPIs through fluctuations, especially under anomalous situations. The study emphasized the challenge of accurately distinguishing fluctuations from normal variations in KPIs with various structural characteristics. The proposed approach automatically determines flux-correlation between two KPIs, including temporal order and direction of fluctuations, without manual algorithm selection or parameter tuning through robust feature engineering and cross-correlation.

Tan et al. [240] proposed a multivariate correlation analysis system for attack detection by extracting geometric correlations between characteristics of network traffic. Their solution utilizes the Mahalanobis distance to measure the similarity between traffic records. Bateni and Baraani [40] presented an Enhanced Random Directed Time Window (ERDTW) alert selection policy based on sliding time windows analysis. ERDTW classifies time intervals into relevant (safe) and irrelevant (dangerous) based on attributes described in mathematical logic rules. For example, if a time interval contains numerous alerts with the same IP address, it is more likely to be flagged as dangerous.

5.10 Incident Mitigation Methods

Through the triage and diagnosis steps of incident management, valuable knowledge is gained, including the identification of incident scope, retrieval of historical duplicates, and analysis of root

Table 25. Study of particularities of AIOps Incident Mitigation methods.

Ref.	Year	Interpretability	Scalability	Robustness	Temporal Evaluation	Human-in-the-loop
[300]	2016	●		●		
[252]	2017	●	●	●	●	●
[84]	2012	●			●	

causes. This knowledge enables the initiation of automatic repair actions known as mitigation or remediation actions. Incident mitigation has received less attention compared to reporting and diagnosis tasks, as it is often a consequence of the outcomes of those processes. Once the underlying problem is clarified through diagnosis, the recovery steps become readily identifiable and attainable without the need for complex models. However, our commitment extends to providing a list of research works that focus on resolution tasks, even when triage or diagnosis are involved.

In a study conducted by [300], similarity-based algorithms are proposed to suggest resolutions for recurring problems based on incident tickets. The approach retrieves k suggestions for ticket resolution using a k-NN approach. The similarity between tickets is evaluated using a combination of numerical, categorical, and textual data, with individual and aggregate similarity measures defined. The solution is further extended to address false-positive tickets in both historical and incoming data. This is achieved by classifying tickets using a binary classifier and weighing ticket importance based on the prediction outcome. The final solution recommendation considers both importance and similarity. The paper also explores ideas for improving feature extraction, such as topic discovery and metric learning. Wang et al. [252] propose a cognitive framework based on ontologies to construct domain-specific knowledge and suggest recovery actions for IT service management tickets. The approach involves analyzing free-form text in ticket summaries and resolution descriptions. Domain-specific phrases are extracted using language processing techniques, and an ontology model is developed to define keywords, classes, relations, and a hierarchy. This model is then utilized to recommend resolution actions by matching concept patterns extracted from incoming and historical tickets using similarity functions like the Jaccard distance. Lin et al. [152] employed natural language processing techniques to predict repair actions for hardware failures based on closed incident tickets. Through the analysis of raw text logs, up to five repair actions are recommended. Ding et al. [84] proposed an automated mining-based approach for suggesting appropriate healing actions. The method involves generating signatures of an issue using transaction logs, retrieving historical issues based on these signatures, and suggesting a suitable healing action by adapting actions used for similar past issues.

6 PUBLICLY AVAILABLE DATASETS AND BENCHMARKS FOR AIOPS METHODS

In this section, we present a comprehensive overview of the key publicly available datasets and benchmarks relevant to the field of AIOps, specifically in the context of incident management procedures across various application areas. Table 26 serves as a valuable resource, not only listing datasets used in prior research methodologies but also highlighting additional datasets that have remained untapped in previous studies, yet hold significance for the respective research application domains. We organize these datasets by incident task or application area, specifying their data sources, cross-referencing them with the methodologies they have been utilized in, and, importantly, providing direct links to access the datasets along with detailed descriptions. It's worth noting that datasets designed for incident triage can also serve incident prioritization purposes and sub-categories, including assignment, classification, and deduplication.

Table 26. Publicly Available Datasets and Benchmarks for AIOps Methods.

Application Area	**Dataset/Bench**	**Data Source**	**Refs.**	**Link for data and description**
Intrusion Detection Systems	ABILENE	Network Traffic	[129, 130]	https://tinyurl.com/2p8d3ufb
Intrusion Detection Systems	SNDLIB	Network Traffic	[130]	https://tinyurl.com/yc6dfdkd
Intrusion Detection Systems	SWAT/WADI	Network Metrics	[33, 80, 149]	https://tinyurl.com/5n87td8c
Intrusion Detection Systems	CIC-IDS2017	Network Traffic	N/A	https://tinyurl.com/bdezufhc
Denial of Service Attacks	CIC-DDoS2019	Network Traffic	N/A	https://tinyurl.com/8ywj34sf
Packet Injection Attacks	NETRESEC	Network Traffic	N/A	https://tinyurl.com/cf3a42m9
Edge Streams	SNAP	Network Traffic	[283]	https://tinyurl.com/2nv4h37s
Edge Streams	DBLP	Network Traffic	[58, 283]	https://tinyurl.com/34n576wp
Edge Streams	DARPA	Network Traffic	[58, 281]	https://tinyurl.com/55ev547r
Edge Streams	ENRON	Network Traffic	[58, 281]	https://tinyurl.com/3k96xfat
Traffic Anomaly Detection	MAWI	Network Metrics	[228]	https://tinyurl.com/4n2uc2te
Anomaly Detection	Yahoo	Univariate TS	[213]	https://tinyurl.com/5awjuj85
Software Changes	Alibaba	Univariate TS	[294]	https://tinyurl.com/mrjddhvk
Software Changes	AIOps2018	Univariate TS	[256]	https://tinyurl.com/bdzzeuwz
Anomaly Detection	NASA-SMAP	Multivariate TS	[33, 233]	https://tinyurl.com/m8pnwvkf
Anomaly Detection	SMD	Multivariate TS	[33, 149, 233]	https://tinyurl.com/yj5au5me
Anomaly Detection	ASD	Multivariate TS	[149]	https://tinyurl.com/yj5au5me
Anomaly Detection	NASA-MSL	Multivariate TS	[233]	https://tinyurl.com/ypcafk99
Anomaly Detection	Power Plant	Multivariate TS	[285]	https://tinyurl.com/ynuz78s8
Log Anomaly Detection	OpenStack	Log Events	[85]	https://tinyurl.com/yhhckue3
Log Anomaly Detection	HFDS/ Hadoop	Log Events	All log methods	https://tinyurl.com/2j8ebupx
Log Anomaly Detection	BGL	Log Events	[99, 173, 267]	https://tinyurl.com/2j8ebupx
Log Anomaly Detection	Thunderbird	Log Events	[99]	https://tinyurl.com/2j8ebupx
Log Anomaly Detection	Spark/Apache/HPC	Log Events	N/A	https://tinyurl.com/2j8ebupx
Software Defect Prediction	PROMISE	Code Metrics	[137, 175, 244, 253]	https://tinyurl.com/bd9zz34m
Software Defect Prediction	Eclipse	Code Metrics	[79, 184]	https://tinyurl.com/2p899sdd
Software Defect Prediction	NASA	Code Metrics	[79, 87]	https://tinyurl.com/2a5869vz
Software Defect Prediction	Code4BENCH	Defect Code Benchmark	[168]	https://tinyurl.com/3bk7fnu3
Software Defect Prediction	AEEM	Code and Process Metrics	[188]	https://tinyurl.com/zr6cmsb9
Software Defect Prediction	GHPR	Java Code and Metrics	[275]	https://tinyurl.com/3zpp4d7a
Software Defect Prediction	JIRA	Open Source Project	N/A	https://tinyurl.com/zr6cmsb9
Disk Failures	SMART	Multivariate TS	[257, 272, 297]	https://tinyurl.com/2vexmjmu
Disk Failures	Backblaze	Multivariate TS	[269]	https://tinyurl.com/rfp4j8jw
DRAM Failures	DRAM	Multivariate TS	[237]	https://tinyurl.com/3yyz7e67
Remaining Useful Lifetime	C-MAPSS	Multivariate TS	[166, 265, 298]	https://tinyurl.com/sbyu5584
Remaining Useful Lifetime	Milling	Multivariate TS	[298]	https://tinyurl.com/2p8zjevy
Remaining Useful Lifetime	PHM2008	Multivariate TS	[142, 298]	https://tinyurl.com/2r2955ss
Remaining Useful Lifetime	XJTU-SY	Multivariate TS	[142]	https://tinyurl.com/272nxd3v
Remaining Useful Lifetime	PRONOSTICA	Multivariate TS	N/A	https://tinyurl.com/4c8exsk9
Software Prediction	Google Cluster	Traces and Metrics	[110]	https://tinyurl.com/4abzuz8m
Bug Triage	Eclipse	Bug Reports	[45, 100, 186, 243, 278] [24, 254, 266, 280] [107, 236, 239, 268] [36, 103, 176, 299]	https://tinyurl.com/352v7ddc
Bug Triage	Mozilla	Bug Reports	[45, 61, 254, 266, 280] [176, 236, 239, 268]	https://tinyurl.com/352v7ddc
Bug Triage	HFDS/ Hadoop	Bug Reports	[271]	https://tinyurl.com/352v7ddc
Bug Triage	Thunderbird	Bug Reports	[176]	https://tinyurl.com/352v7ddc
Bug Triage	NetBeans	Stack Traces	[248]	https://tinyurl.com/ycxxyjfd
Bug Triage	Cassandra/Mesos/ JDT/Spark	Bug Reports	N/A	https://tinyurl.com/352v7ddc
Database Issues	Infologic	SQL Queries Alerts, ASH	[210]	https://tinyurl.com/3fyb2s76
Software Fault Localization	Defect4JS	Bug Reports	[144]	https://tinyurl.com/4uvcu24y
Software Fault Localization	MEGA	Bug Reports	[141]	https://tinyurl.com/mpas9yan
Intrusion Detection Systems	KDDCup1999	Network Metrics	[240]	https://tinyurl.com/mrxxfftu

Table 27. Summary of all reviewed AIOps methods for Incident Management Procedure.

Incident Task	**Application Area**	**Type of AI Techniques**	
Incident Detection		Auto-Encoders (Others) [149, 274, 283]	FSM [93]
		Auto-Regressive Models [201]	GANs [33, 267]
		Clustering [105, 120, 150, 225, 283]	Graph Models [80, 160]
		CNNs [213, 285]	Language Models [99, 173, 291]
		Combinatorial Optimization [120]	Markov Models [37, 225]
		Conventional Classifiers [158]	Nearest Neighbors Search [225]
		Dimensionality Reduction [129, 200, 276]	RNNs [85, 173, 233, 256, 267, 285, 291, 294]
		Fourier Transform [213]	Statistical Models [130, 201, 270, 281]
			Continued ... [58, 228, 247, 274]
Incident Prediction	SDP	Bayesian Models [79, 175, 184, 253]	Graph Models [275]
		CNNs [137]	Language Models [244]
		Conventional Classifiers [87, 188]	RNNs [169, 244]
		Convention Regressors [187, 198]	Topic Modeling [275]
		Dimensionality Reduction [187]	
	Hardware Failures Aging/Rejuv. RUL	Clustering [246]	GANs [142]
		CNNs [123, 166, 237]	Markov Models [246, 288, 297]
		Conventional Classifiers [76, 269]	Pattern Mining [288]
		Conventional Regressors [26, 30, 95, 235, 246]	RL [42]
		Dimensionality Reduction [123]	RNNs [76, 166, 265, 272, 298]
		Fourier Transform [123]	Statistical Models [95, 257]
	Software Failures	Auto-Regressive Models [56]	Conventional Classifiers [92, 153, 293]
		Bayesian Models [69, 204]	RNNs [110, 153, 287]
Incident Prioritization		Bayesian Models [102, 156]	Combinatorial Optimization [100]
		CNNs [61]	Conventional Regressors [243]
		Ranking Models [295]	
Incident Assignment		Bayesian Models [24, 45, 186, 278]	Pattern Mining [212]
		CNNs [60, 132, 202]	RNNs [60, 212, 266]
		Markov Models [224]	Similarity-based [254]
Incident Classification		Clustering [292]	Markov Models [151]
		Conventional Classification [284]	Topic Modeling [268, 280]
Incident Deduplication		Clustering [107]	Ranking Models [299]
		CNNs [271]	Similarity-based [36, 218, 236, 239]
		Language Models [103, 176]	Other Similarity-based [50, 74, 135, 219]
			Continued ... [39, 181, 183, 248]
Root Cause Analysis	Technical Troubleshooting	Bayesian Models [139, 167]	Markov Models [194, 221]
		CNNs [148]	Pattern Mining [147, 154, 210, 238]
		Graph Models [31, 35, 62, 115]	RNNs [148]
	SFL	Bayesian Models [22]	Nearest Neighbors Search [215]
		CNNs [144]	Pattern Mining [55, 81]
		Combinatorial Optimization [229]	Ranking Models [279]
		Conventional Classifiers [141, 229]	RNNs [141, 144]
		Markov Models [289]	Statistical Models [157, 259]
Incident Correlation		Conventional Classifiers [40]	Similarity-based [234, 240]
		Nearest Neighbors Search [163]	
Incident Mitigation		Language Models [152]	Pattern Mining [84, 252]
		Nearest Neighbors Search [300]	Topic Modeling [300]

We acknowledge the paramount importance of this information, both for newcomers venturing into the domain of AIOps and for practitioners seeking to replicate existing research findings, make informed comparisons among similar techniques within the same research domain, and evaluate their own contributions. While some previous surveys have offered lists of datasets tailored to specific domains, our contribution, to the best of our knowledge, represents the first comprehensive compilation spanning multiple application areas.

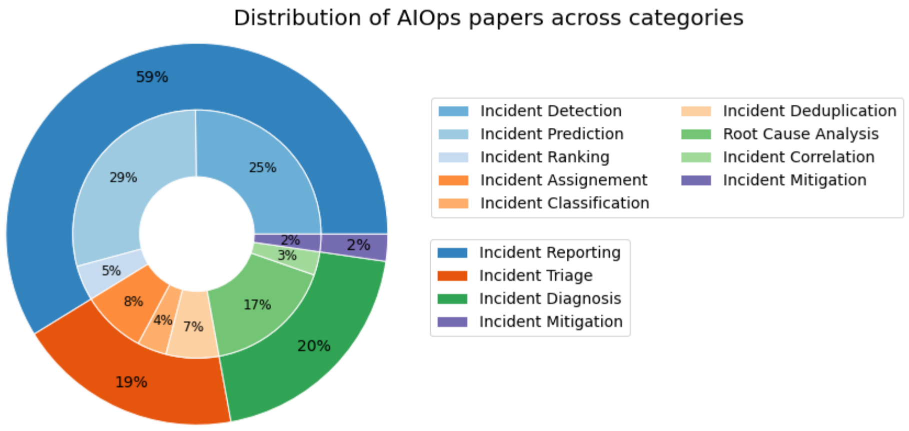


Fig. 14. Distribution of analyzed AIOps papers across incident tasks and their subcategories.

7 CONCLUSION AND OPEN CHALLENGES

As stated in the introduction, we have observed a notable surge of interest in the AIOps domain in recent years, both within the research and industry sectors. However, it is imperative to acknowledge that this field still lacks centralization and a structured framework. This arises from the need to combine diverse specialized disciplines, encompassing software engineering, machine learning, big data analysis, optimization, and more. AIOps, characterized by its novelty and inherently interdisciplinary nature, has yet to establish a distinct and cohesive identity as a clearly defined area of study. This presents substantial challenges for both practitioners and researchers to fully comprehend the current state of the art and identify potential limitations and gaps. The absence of standardized terminology and conventions for aspects such as data management, problem targeting, focus areas, implementation details, requirements, and capabilities further exacerbates the situation. This absence not only result in the absence of technical guidelines and a coherent roadmap for implementing effective AIOps solutions but also makes it difficult for new researchers entering the field to discover and compare contributions from various disciplines addressing the same problems.

As a result, this survey serves as an introductory resource to the AIOps domain, specifically tailored for incident management procedures. Our primary goal was to establish a foundational knowledge base for AIOps, which can benefit both industries looking to transition to AIOps infrastructures and future research endeavors. In addition to presenting the fundamentals, including the essential building blocks required for a systematic AIOps approach in intelligent and effective incident management, while addressing pain points and desired outcomes, we offer a comprehensive exploration of existing approaches designed to handle various tasks within the defined incident management process.

In Figure 14, we analyze the distribution of AIOps papers reviewed in this survey across incident tasks and their subcategories. It is evident that certain research categories have garnered more attention than others, resulting in an imbalance in contributions across each phase. This disparity can be attributed to several factors. For instance, the incident prediction and detection phases collectively constitute more than half of the papers analyzed in this survey, sourced from prestigious conferences and journals. Conversely, categories such as incident classification, correlation, and mitigation have received relatively less attention. This discrepancy aligns logically with the fundamental requirements of an AIOps framework, where detection, prediction, and root cause analysis are of paramount importance. Furthermore, as discussed earlier, some categories may not be as critical to pursue. For example, if the root cause can be efficiently identified by referencing a past duplicate incident, there may be no need for an automated mitigation technique. Similarly, some categories may become less essential based on specific use cases. For instance, if an incident is already routed to a qualified individual with no competing priorities, the ranking process may be of reduced significance. However, it remains beneficial to encourage diverse contributions employing various techniques and the latest algorithms to ensure that tasks, when required, can be executed accurately and swiftly.

In relation to each incident task, we observed specific aspects that have garnered considerable research focus. This includes identifying prevalent contexts within the related research, extensively utilized data for those contexts, as well as the most commonly employed model types and learning paradigms, among other factors. For example, when examining incident detection methods, a predominant focus emerges on addressing technical aspects, including hardware and network layers, as well as application layers. This is typically accomplished by analyzing Key Performance Indicators (KPIs), system metrics, network traffic data, and event logs. The prevailing models tend to be unsupervised, with significant attention placed on statistical models, dimensionality reduction approaches, and general auto-encoders like Recurrent Neural Networks (RNNs). In incident prediction tasks, research efforts have also addressed failures occurring in the functional strata. Many of these methods are supervised and leverage patterns from past failures or outages. They frequently make use of forecasting methods, especially for time series data, particularly in contexts like software aging and estimating remaining useful lifetimes.

When it comes to incident prioritization, topology and alerting signals data play a prominent role, often using supervised techniques. In contrast, incident reports serve as the primary source of information for tasks related to assignment and deduplication. However, for these latter two tasks, most methods predominantly focus on addressing software-related issues, particularly software bugs. In terms of root cause analysis methods, interpretability is a key consideration. For software fault localization, the approach predominantly involves analyzing source code through supervised methods. On the other hand, technical fault localization methods tend to be unsupervised and draw from descriptive models in pattern mining, often incorporating topology data to contextualize faults.

In addition to the significant challenge addressed in this survey, which involves organizing the body of knowledge within the AIOps research area, we have identified several other concerns, pain points, and open challenges that require attention. These include aspects related to the design, unique characteristics, availability, and reproducibility of AIOps models.

To begin, our initial observation highlights that despite the myriad challenges faced by predictive models, including issues like the absence of clear ground truth labels, the need for manual efforts to obtain high-quality data, extremely imbalanced datasets, and the complexity of dependencies and relationships among components and services, AIOps continues to place a significant emphasis on the development of predictive models, particularly for incident detection and prediction. However, there exists a lesser-known yet highly valuable approach that has been underutilized. This approach involves employing descriptive models, such as pattern mining in general, and specifically, techniques like supervised rule discovery [32, 262] and formal concept analysis [55, 122]. These models harness the power of data mining to extract informative patterns from data, which can be instrumental in detecting, diagnosing, and resolving issues. Descriptive models offer distinct advantages when it comes to tackling challenges related to data diversity, complexity, and quality. This makes them particularly valuable in scenarios involving tasks like deduplication of incidents and dealing with intricate dependencies. Therefore, it is imperative to shift our focus towards enhancing descriptive models in conjunction with predictive models, recognizing the unique strengths they bring to the AIOps landscape.

Additionally, it is crucial to emphasize the importance of evolving the evaluation methods for these models. While the majority of research examined in this survey has centered on contingency table metrics, only a few have considered the contextual and temporal aspects outlined in Sections 4.2 and 3.4. For instance, the challenge also is to ensure that the behavior of the model during the training phase is consistent with its performance in the testing and production phases. Traditional metrics used to assess models are susceptible to the contamination zone phenomenon [91], which may lead to erroneous assessments. Indeed, Fourure et al. [91], highlight that by parameterizing the proportion of data between training and testing sets, the F1-score of anomaly detection models can be artificially increased.

When it comes to addressing interpretability concerns, many research efforts that employ black box models to carry out their tasks acknowledge the importance of incorporating an interpretation layer into their methods. They claim that the output of their models is accompanied by explanations regarding the reasons behind their predictions. However, several critical questions persist in this regard. It remains unclear how these interpretations remain internally consistent when the model encounters new data, or how they compare externally when different models produce the same results but with differing interpretations. Additionally, the stability of model interpretations over time, particularly with updates and improvements, raises important questions (see Section 3.4 for further details). Furthermore, only a limited number of methods involve human expertise in the process. In reality, practitioners' insights can significantly guide both the learning and implementation of these models. Interestingly, in the context of pattern mining approaches, several techniques, such as Subjective Interestingness [77], can facilitate the process of updating a user's knowledge about the most interesting data over time. Scalability is another major concern. In AIOps environments, it is essential not only to focus on the efficiency of the model but also on its overall performance. While optimizing (TTx) times (including detection, engagement, and mitigation), which have received comparatively less attention, is crucial for the successful implementation of automated incident management procedures, performance evaluation is often overlooked when comparing different models addressing the same research field. In practical scenarios, a model that takes less time to execute while maintaining a 90% F-Score for detection may be preferred over a model with a 95% F-Score that requires a longer execution time. This emphasizes the need to strike a balance between model performance and efficiency in real-world AIOps applications.

Finally, to advance this field significantly, it is imperative to advocate for a closer partnership between academia and industry. Furthermore, open-sourcing initiatives should be encouraged. While it is totally understandable that publishing datasets may raise potential rights or confidentiality concerns, and sharing models employed in production scenarios could be susceptible to reverse-engineering, there is a critical need to find common ground, especially in the research community. Facilitating the reproducibility of certain results can be a significant catalyst for advancing research in this field. This collaborative approach, where academia and industry work together and contribute to open-source initiatives, can pave the way for the development of more robust and impactful AIOps solutions.

REFERENCES

[1] 2023. Amazon Kinesis. https://aws.amazon.com/fr/kinesis/
[2] 2023. Apache Kafka. https://kafka.apache.org/
[3] 2023. Apache Nifi. https://nifi.apache.org/
[4] 2023. Apache Superset. https://superset.apache.org/
[5] 2023. Azure Event Hubs. https://learn.microsoft.com/fr-fr/azure/event-hubs/
[6] 2023. Beats from the ELK stack. https://www.elastic.co/fr/beats/
[7] 2023. Build Lakehouses with Delta Lake. https://delta.io/
[8] 2023. Clickhouse. https://clickhouse.com/
[9] 2023. Elasticsearch. https://www.elastic.co/
[10] 2023. Fluentd. https://www.fluentd.org/
[11] 2023. Google Cloud Pub/Sub. https://cloud.google.com/pubsub
[12] 2023. Grafana. https://grafana.com/
[13] 2023. IBM. https://www.ibm.com/cloud/architecture/architectures/sm-aiops/reference-architecture/
[14] 2023. InfluxDB. https://www.influxdata.com/
[15] 2023. Kibana from the ELK stack. https://www.elastic.co/fr/kibana/
[16] 2023. Lucene Apache. https://lucene.apache.org/
[17] 2023. Metabase. https://www.metabase.com/
[18] 2023. RabbitMQ. https://www.rabbitmq.com/
[19] 2023. RocketMQ. https://rocketmq.apache.org/
[20] 2023. Telegraf from InfluxData. https://www.influxdata.com/time-series-platform/telegraf/
[21] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. 2006. An evaluation of similarity coefficients for software fault localization. In *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. IEEE, 39–46.
[22] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. 2009. Spectrum-based multiple fault localization. In *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 88–99.
[23] V Akila, G Zayaraz, and V Govindasamy. 2014. Bug triage in open source systems: a review. *International Journal of Collaborative Enterprise* 4, 4 (2014), 299–319.
[24] Mamdouh Alenezi, Kenneth Magel, and Shadi Banitaan. 2013. Efficient Bug Triaging Using Text Mining. *J. Softw.* 8, 9 (2013), 2185–2190.
[25] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. 2014. Similarity measures for OLAP sessions. *Knowledge and information systems* 39, 2 (2014), 463–489.
[26] Javier Alonso, Jordi Torres, Josep Ll Berral, and Ricard Gavalda. 2010. Adaptive on-line software aging prediction based on machine learning. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE, 507–516.
[27] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2008. Is it a bug or an enhancement? A text-based approach to classify change requests. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*. 304–318.
[28] John Anvik, Lyndon Hiew, and Gail C Murphy. 2005. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*. 35–39.
[29] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. 2006. Clustering-based materialized view selection in data warehouses. In *East European conference on advances in databases and information systems*. Springer, 81–95.
[30] Jean Araujo, Rubens Matos, Vandi Alves, Paulo Maciel, F Vieira de Souza, Rivalino Matias Jr, and Kishor S Trivedi. 2014. Software aging in the eucalyptus cloud computing infrastructure: characterization and rejuvenation. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 10, 1 (2014), 1–22.
[31] Mona Attariyan, Michael Chow, and Jason Flinn. 2012. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*. 307–320.
[32] Martin Atzmueller. 2015. Subgroup discovery. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 5, 1 (2015), 35–49.
[33] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. 2020. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3395–3404.
[34] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing* 1, 1 (2004), 11–33.
[35] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A Maltz, and Ming Zhang. 2007. Towards highly reliable enterprise network services via inference of multi-level dependencies. *ACM SIGCOMM Computer Communication Review* 37, 4 (2007), 13–24.

[36] Sean Banerjee, Bojan Cukic, and Donald Adjeroh. 2012. Automated duplicate bug report classification using subsequence matching. In *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*. IEEE, 74–81.

[37] June-ho Bang, Young-Jong Cho, and Kyungran Kang. 2017. Anomaly detection of network-initiated LTE signaling traffic in wireless sensor and actuator networks based on a Hidden semi-Markov Model. *Computers & Security* 65 (2017), 108–120.

[38] Abdul Ali Bangash, Hareem Sahar, Abram Hindle, and Karim Ali. 2020. On the time-based conclusion stability of cross-project defect prediction models. *Empirical Software Engineering* 25, 6 (2020), 5047–5083.

[39] Kevin Bartz, Jack W. Stokes, John C. Platt, Ryan Kivett, David Grant, Silviu Calinoiu, and Gretchen Loihle. 2008. Finding Similar Failures Using Callstack Similarity. In *SysML*.

[40] Mehdi Bateni and Ahmad Baraani. 2013. Time Window Management for Alert Correlation using Context Information and Classification. *International Journal of Computer Network & Information Security* 5, 11 (2013).

[41] Soeren Becker, Florian Schmidt, Anton Gulenko, Alexander Acker, and Odej Kao. 2020. Towards aiops in edge computing environments. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 3470–3475.

[42] Luca Bellani, Michele Compare, Piero Baraldi, and Enrico Zio. 2019. Towards developing a novel framework for practical phm: A sequential decision problem solved by reinforcement learning and artificial neural networks. *International Journal of Prognostics and Health Management* 10, 4 (2019).

[43] Anes Bendimerad, Youcef Remil, Romain Mathonat, and Mehdi Kaytoue. 2023. On-premise Infrastructure for AIOps in a Software Editor SME: An experience report. In *31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESECFSE*.

[44] Tarek Berghout and Mohamed Benbouzid. 2022. A systematic guide for predicting remaining useful life with machine learning. *Electronics* 11, 7 (2022), 1125.

[45] Pamela Bhattacharya and Iulian Neamtiu. 2010. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *2010 IEEE International Conference on Software Maintenance*. IEEE, 1–10.

[46] Jasmin Bogatinovski, Sasho Nedelkoski, Alexander Acker, Florian Schmidt, Thorsten Wittkopp, Soeren Becker, Jorge Cardoso, and Odej Kao. 2021. Artificial Intelligence for IT Operations (AIOPS) Workshop White Paper. *arXiv preprint arXiv:2101.06054* (2021).

[47] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the association for computational linguistics* 5 (2017), 135–146.

[48] Marcello Braglia, Gionata Carmignani, Marco Frosolini, and Francesco Zammori. 2012. Data classification and MTBF prediction with a multivariate analysis approach. *Reliability Engineering & System Safety* 97, 1 (2012), 27–35.

[49] Lionel C. Briand, John W. Daly, and Jurgen K Wust. 1999. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on software Engineering* 25, 1 (1999), 91–121.

[50] Mark Brodie, Sheng Ma, Guy M. Lohman, Laurent Mignet, Natwar Modani, Mark Wilding, Jon Champlin, and Peter Sohn. 2005. Quickly Finding Known Software Problems via Automated Symptom Matching. In *Second International Conference on Autonomic Computing (ICAC 2005)*. IEEE Computer Society, 101–110.

[51] Zaharah A Bukhsh, Hajo Molegraaf, and Nils Jansen. 2023. A maintenance planning framework using online and offline deep reinforcement learning. *Neural Computing and Applications* (2023), 1–12.

[52] D Cappuccio. 2013. Ensure cost balances out with risk in highavailability data centers. *Gartner, July* (2013).

[53] Saul Carliner. 2004. An overview of online learning. (2004).

[54] Vittorio Castelli, Richard E Harper, Philip Heidelberger, Steven W Hunter, Kishor S Trivedi, Kalyanaraman Vaidyanathan, and William P Zeggert. 2001. Proactive management of software aging. *IBM Journal of Research and Development* 45, 2 (2001), 311–332.

[55] Peggy Cellier, Mireille Ducassé, Sébastien Ferré, and Olivier Ridoux. 2008. Formal concept analysis enhances fault localization in software. In *Formal Concept Analysis: 6th International Conference, ICFCA 2008, Montreal, Canada, February 25-28, 2008. Proceedings 6*. Springer, 273–288.

[56] Thanyalak Chalermarrewong, Tiranee Achalakul, and Simon Chong Wee See. 2012. Failure prediction of data centers using time series and fault tree analysis. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*. IEEE, 794–799.

[57] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.

[58] Yen-Yu Chang, Pan Li, Rok Sosic, MH Afifi, Marco Schweighauser, and Jure Leskovec. 2021. F-fade: Frequency factorization for anomaly detection in edge streams. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 589–597.

[59] Surajit Chaudhuri, Ashish Kumar Gupta, and Vivek Narasayya. 2002. Compressing SQL workloads. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 488–499.

[60] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous incident triage for large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 364–375.

[61] Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, et al. 2020. How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 373–384.

[62] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. 2014. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 1887–1895.

[63] Tse-Hsun Chen, Weiyi Shang, Zhen Ming Jiang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. 2014. Detecting performance anti-patterns for applications developed using object-relational mapping. In *Proceedings of the 36th International Conference on Software Engineering*. 1001–1012.

[64] Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, et al. 2019. Outage prediction and diagnosis for cloud service systems. In *The World Wide Web Conference*. 2659–2665.

[65] Zhuangbin Chen, Yu Kang, Feng Gao, Li Yang, Jeffrey Sun, Zhangwei Xu, Pu Zhao, Bo Qiao, Liqun Li, Xu Zhang, et al. 2020. Aiops innovations of incident management for cloud services. (2020).

[66] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1487–1497.

[67] Shyam R Chidamber and Chris F Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering* 20, 6 (1994), 476–493.

[68] Michael Chow, David Meisner, Jason Flinn, Daniel Peek, and Thomas F Wenisch. 2014. The mystery machine: End-to-end performance analysis of large-scale internet services. In *11th {USENIX} symposium on operating systems design and implementation ({OSDI} 14)*. 217–231.

[69] Ira Cohen, Jeffrey S Chase, Moises Goldszmidt, Terence Kelly, and Julie Symons. 2004. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control.. In *OSDI*, Vol. 4. 16–16.

[70] Nick Craswell. 2009. *Mean Reciprocal Rank*. Springer US, 1703–1703.

[71] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient Log Parsing Using n n-Gram Dictionaries. *IEEE Transactions on Software Engineering* 48, 3 (2020), 879–892.

[72] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2010. An extensive comparison of bug prediction approaches. In *2010 7th IEEE working conference on mining software repositories (MSR 2010)*. IEEE, 31–41.

[73] Yingnong Dang, Qingwei Lin, and Peng Huang. 2019. AIOps: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 4–5.

[74] Yingnong Dang, Rongxin Wu, Hongyu Zhang, Dongmei Zhang, and Peter Nobel. 2012. ReBucket: A method for clustering duplicate crash reports based on call stack similarity. In *34th International Conference on Software Engineering, ICSE*. 1084–1093.

[75] Narjes Davari, Bruno Veloso, Gustavo de Assis Costa, Pedro Mota Pereira, Rita P Ribeiro, and João Gama. 2021. A survey on data-driven predictive maintenance for the railway industry. *Sensors* 21, 17 (2021), 5739.

[76] Nickolas Allen Davis, Abdelmounaam Rezgui, Hamdy Soliman, Skyler Manzanares, and Milagre Coates. 2017. Failuresim: a system for predicting hardware failures in cloud data centers using neural networks. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 544–551.

[77] Tijl De Bie. 2011. Maximum entropy models and subjective interestingness. *Data Mining and Knowledge Discovery* 23, 3 (2011), 407–446.

[78] Shaleen Deep, Anja Gruenheid, Paraschos Koutris, Jeffrey F. Naughton, and Stratis Viglas. 2020. Comprehensive and Efficient Workload Compression. *Proc. VLDB Endow.* 14, 3 (2020), 418–430.

[79] Karel Dejaeger, Thomas Verbraken, and Bart Baesens. 2012. Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Transactions on Software Engineering* 39, 2 (2012), 237–257.

[80] Ailin Deng and Bryan Hooi. 2021. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 4027–4035.

[81] Tristan Denmat, Mireille Ducassé, and Olivier Ridoux. 2005. Data mining and cross-checking of execution traces: a re-interpretation of jones, harrold and stasko test information. In *Proceedings of the 20th IEEE/ACM International Conference on Automated software engineering*. 396–399.

[82] Tejinder Dhaliwal, Foutse Khomh, and Ying Zou. 2011. Classifying field crash reports for fixing bugs: A case study of Mozilla Firefox. In *IEEE 27th International Conference on Software Maintenance, ICSM*. 333–342.

[83] Josu Díaz-de Arcaya, Ana I Torre-Bastida, Raúl Miñón, and Aitor Almeida. 2023. Orfeon: An AIOps framework for the goal-driven operationalization of distributed analytical pipelines. *Future Generation Computer Systems* 140 (2023), 18–35.

[84] Rui Ding, Qiang Fu, Jian-Guang Lou, Qingwei Lin, Dongmei Zhang, Jiajun Shen, and Tao Xie. 2012. Healing online service systems via mining historical issue repositories. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. 318–321.

[85] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[86] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2012. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering* 17 (2012), 531–577.

[87] Karim O Elish and Mahmoud O Elish. 2008. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software* 81, 5 (2008), 649–660.

[88] Stephen Elliot. 2014. DevOps and the cost of downtime: Fortune 1000 best practice metrics quantified. *International Data Corporation (IDC)* (2014).

[89] Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, and John Grundy. 2018. Metric selection and anomaly detection for cloud operations using log and metric correlation analysis. *Journal of Systems and Software* 137 (2018), 531–549.

[90] Olga Fink. 2020. Data-driven intelligent predictive maintenance of industrial assets. *Women in Industrial and Systems Engineering: Key Advances and Perspectives on Emerging Topics* (2020), 589–605.

[91] Damien Fourure, Muhammad Usama Javaid, Nicolas Posocco, and Simon Tihon. 2021. Anomaly detection: How to artificially increase your f1-score with a biased evaluation protocol. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part IV*. Springer, 3–18.

[92] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko. 2013. Failure prediction based on log files using random indexing and support vector machines. *Journal of Systems and Software* 86, 1 (2013), 2–11.

[93] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158.

[94] Zhiwei Gao, Carlo Cecati, and Steven X Ding. 2015. A survey of fault diagnosis and fault-tolerant techniques—Part I: Fault diagnosis with model-based and signal-based approaches. *IEEE transactions on industrial electronics* 62, 6 (2015), 3757–3767.

[95] Sachin Garg, Aad Van Moorsel, Kalyanaraman Vaidyanathan, and Kishor S Trivedi. 1998. A methodology for detection and estimation of software aging. In *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No. 98TB100257)*. IEEE, 283–292.

[96] Todd L Graves, Alan F Karr, James S Marron, and Harvey Siy. 2000. Predicting fault incidence using software change history. *IEEE Transactions on software engineering* 26, 7 (2000), 653–661.

[97] Michael Grottke, Rivalino Matias, and Kishor S Trivedi. 2008. The fundamentals of software aging. In *2008 IEEE International conference on software reliability engineering workshops (ISSRE Wksp)*. Ieee, 1–6.

[98] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2019. A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv.* 51, 5 (2019), 93:1–93:42.

[99] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.

[100] Chetna Gupta, Pedro RM Inacio, and Mário M Freire. 2022. Improving software maintenance with improved bug triaging. *Journal of King Saud University-Computer and Information Sciences* 34, 10 (2022), 8757–8764.

[101] Maurice H Halstead. 1977. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc.

[102] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *network and distributed systems security symposium*.

[103] Jianjun He, Ling Xu, Meng Yan, Xin Xia, and Yan Lei. 2020. Duplicate bug report detection using dual-channel convolutional neural networks. In *Proceedings of the 28th International Conference on Program Comprehension*. 117–127.

[104] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–37.

[105] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 60–70.

[106] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 207–218.

[107] Lyndon Hiew. 2006. *Assisted detection of duplicate bug reports.* Ph. D. Dissertation. University of British Columbia.

[108] Aaron Huff. 2015. Breaking down the cost. *Commercial Carrier Journal* (2015).

[109] Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. 2015. Performance anomaly detection and bottleneck identification. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 1–35.

[110] Tariqul Islam and Dakshnamoorthy Manivannan. 2017. Predicting application failure in cloud: A machine learning approach. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*. IEEE, 24–31.

[111] Shrainik Jain, Bill Howe, Jiaqi Yan, and Thierry Cruanes. 2018. Query2Vec: An Evaluation of NLP Techniques for Generalized Workload Analytics. *arXiv preprint arXiv:1801.05613* (2018).

[112] Nicholas Jalbert and Westley Weimer. 2008. Automated duplicate detection for bug tracking systems. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 52–61.

[113] David Jauk, Dai Yang, and Martin Schulz. 2019. Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[114] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 111–120.

[115] Vimalkumar Jeyakumar, Omid Madani, Ali Parandeh, Ashutosh Kulshreshtha, Weifei Zeng, and Navindra Yadav. 2019. ExplainIt!–A declarative root-cause analysis engine for time series data. In *Proceedings of the 2019 International Conference on Management of Data*. 333–348.

[116] James A Jones and Mary Jean Harrold. 2005. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. 273–282.

[117] James A Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on Software engineering*. 467–477.

[118] Xiaolin Ju, Shujuan Jiang, Xiang Chen, Xingya Wang, Yanmei Zhang, and Heling Cao. 2014. HSFal: Effective fault localization using hybrid spectrum of full slices and execution slices. *Journal of Systems and Software* 90 (2014), 3–17.

[119] Changhee Jung, Sangho Lee, Easwaran Raman, and Santosh Pande. 2014. Automated memory leak detection for production use. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, Pankaj Jalote, Lionel C. Briand, and André van der Hoek (Eds.). ACM, 825–836.

[120] Amin Karami and Manel Guerrero-Zapata. 2015. A fuzzy anomaly detection system based on hybrid PSO-Kmeans algorithm in content-centric networks. *Neurocomputing* 149 (2015), 1253–1269.

[121] Syaeful Karim, Harco Leslie Hendric Spits Warnars, Ford Lumban Gaol, Edi Abdurachman, Benfano Soewito, et al. 2017. Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset. In *2017 IEEE international conference on cybernetics and computational intelligence (CyberneticsCom)*. IEEE, 19–23.

[122] Mehdi Kaytoue, Sergei O. Kuznetsov, and Amedeo Napoli. 2011. Revisiting Numerical Pattern Mining with Formal Concept Analysis. In *IJCAI*. IJCAI/AAAI, 1342–1347.

[123] Kasem Khalil, Omar Eldash, Ashok Kumar, and Magdy Bayoumi. 2020. Machine learning-based approach for hardware faults prediction. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 11 (2020), 3880–3892.

[124] Taghi M Khoshgoftaar and David L Lanning. 1995. A neural network approach for early detection of program modules having high risk in the maintenance phase. *Journal of Systems and Software* 29, 1 (1995), 85–91.

[125] Aleksandr Khvorov, Roman Vasiliev, George A. Chernishev, Irving Muller Rodrigues, Dmitrij V. Koznov, and Nikita Povarov. 2021. S3M: Siamese Stack (Trace) Similarity Measure. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR*. 266–270.

[126] Myunghwan Kim, Roshan Sumbaly, and Sam Shah. 2013. Root cause detection in a service-oriented architecture. *ACM SIGMETRICS Performance Evaluation Review* 41, 1 (2013), 93–104.

[127] Sunghun Kim, Thomas Zimmermann, and Nachiappan Nagappan. 2011. Crash graphs: An aggregated view of multiple crashes to improve crash triage. In *Proceedings of the 2011 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN*. 486–493.

[128] Gokhan Kul, Duc Luong, Ting Xie, Patrick Coonan, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. 2016. Ettu: Analyzing query intents in corporate databases. In *Proceedings of the 25th international conference companion on world wide web*. 463–466.

[129] Anukool Lakhina, Mark Crovella, and Christophe Diot. 2004. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM computer communication review* 34, 4 (2004), 219–230.

[130] Anukool Lakhina, Mark Crovella, and Christophe Diot. 2005. Mining anomalies using traffic feature distributions. *ACM SIGCOMM computer communication review* 35, 4 (2005), 217–228.

[131] Ahmed Lamkanfi, Serge Demeyer, Quinten David Soetens, and Tim Verdonck. 2011. Comparing mining algorithms for predicting the severity of a reported bug. In *2011 15th European Conference on Software Maintenance and Reengineering*. IEEE, 249–258.

[132] Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. 2017. Applying deep learning based automatic bug triager to industrial projects. In *ESEC/FSE 2017*. ACM, 926–931.

[133] Sangho Lee, Changhee Jung, and Santosh Pande. 2014. Detecting memory leaks through introspective dynamic behavior modelling using machine learning. In *Proceedings of the 36th International Conference on Software Engineering*. 814–824.

[134] Wan-Jui Lee. 2017. Anomaly detection and severity prediction of air leakage in train braking pipes. *International Journal of Prognostics and Health Management* 8, 3 (2017).

[135] Johannes Lerch and Mira Mezini. 2013. Finding Duplicates of Your Yet Unwritten Bug Report. In *17th European Conference on Software Maintenance and Reengineering, CSMR*. 69–78.

[136] Anna Levin, Shelly Garion, Elliot K Kolodner, Dean H Lorenz, Katherine Barabash, Mike Kugler, and Niall McShane. 2019. AIOps for a cloud object storage service. In *2019 IEEE International Congress on Big Data (BigDataCongress)*. IEEE, 165–169.

[137] Jian Li, Pinjia He, Jieming Zhu, and Michael R Lyu. 2017. Software defect prediction via convolutional neural network. In *2017 IEEE international conference on software quality, reliability and security (QRS)*. IEEE, 318–328.

[138] Jing Li, Rebecca J Stones, Gang Wang, Zhongwei Li, Xiaoguang Liu, and Kang Xiao. 2016. Being accurate is not enough: New metrics for disk failure prediction. In *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 71–80.

[139] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022. Causal Inference-Based Root Cause Analysis for Online Service Systems with Intervention Recognition. *arXiv preprint arXiv:2206.05871* (2022).

[140] Tao Li, Chunqiu Zeng, Yexi Jiang, Wubai Zhou, Liang Tang, Zheng Liu, and Yue Huang. 2017. Data-driven techniques in computing system management. *ACM Computing Surveys (CSUR)* 50, 3 (2017), 1–43.

[141] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*. 169–180.

[142] Xiang Li, Wei Zhang, Hui Ma, Zhong Luo, and Xu Li. 2020. Data alignments in machinery remaining useful life prediction using deep adversarial neural networks. *Knowledge-Based Systems* 197 (2020), 105843.

[143] Yangguang Li, Zhen Ming Jiang, Heng Li, Ahmed E Hassan, Cheng He, Ruirui Huang, Zhengda Zeng, Mian Wang, and Pinan Chen. 2020. Predicting node failures in an ultra-large-scale cloud computing platform: an aiops solution. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 2 (2020), 1–24.

[144] Yi Li, Shaohua Wang, and Tien Nguyen. 2021. Fault localization with code coverage representation learning. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 661–673.

[145] Ze Li and Yingnong Dang. 2019. Aiops: Challenges and experiences in azure. *USENIX Association, Santa Clara* (2019), 51–53.

[146] Zhiguo Li and Qing He. 2015. Prediction of railcar remaining useful life by multiple data source fusion. *IEEE Transactions on Intelligent Transportation Systems* 16, 4 (2015), 2226–2235.

[147] Zeyan Li, Chengyang Luo, Yiwei Zhao, Yongqian Sun, Kaixin Sui, Xiping Wang, Dapeng Liu, Xing Jin, Qi Wang, and Dan Pei. 2019. Generic and robust localization of multi-dimensional root causes. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 47–57.

[148] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, et al. 2022. Actionable and interpretable fault localization for recurring failures in online service systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 996–1008.

[149] Zhihan Li, Youjian Zhao, Jiaqi Han, Ya Su, Rui Jiao, Xidao Wen, and Dan Pei. 2021. Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 3220–3230.

[150] Zhihan Li, Youjian Zhao, Rong Liu, and Dan Pei. 2018. Robust and rapid clustering of kpis for large-scale anomaly detection. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[151] Meng-Hui Lim, Jian-Guang Lou, Hongyu Zhang, Qiang Fu, Andrew Beng Jin Teoh, Qingwei Lin, Rui Ding, and Dongmei Zhang. 2014. Identifying recurrent and unknown performance issues. In *2014 IEEE International Conference on Data Mining*. IEEE, 320–329.

[152] Fan Lin, Matt Beadon, Harish Dattatraya Dixit, Gautham Vunnam, Amol Desai, and Sriram Sankar. 2018. Hardware remediation at scale. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 14–17.

[153] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. 2018. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 480–490.

[154] Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, and Dongmei Zhang. 2016. iDice: problem identification for emerging issues. In *Proceedings of the 38th International Conference on Software Engineering*. 214–224.

[155] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. 102–111.

[156] Ying Lin, Zhengzhang Chen, Cheng Cao, Lu-An Tang, Kai Zhang, Wei Cheng, and Zhichun Li. 2018. Collaborative alert ranking for anomaly detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1987–1995.

[157] Chao Liu, Long Fei, Xifeng Yan, Jiawei Han, and Samuel P Midkiff. 2006. Statistical debugging: A hypothesis testing-based approach. *IEEE Transactions on software engineering* 32, 10 (2006), 831–848.

[158] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. 2015. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proceedings of the 2015 internet measurement conference*. 211–224.

[159] Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. 2019. What bugs cause production cloud incidents?. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 155–162.

[160] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection.. In *USENIX annual technical conference*. 1–14.

[161] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software analytics for incident management of online services: An experience report. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 475–485.

[162] Sidi Lu, Bing Luo, Tirthak Patel, Yongtao Yao, Devesh Tiwari, and Weisong Shi. 2020. Making disk failure predictions smarter!. In *FAST*. 151–167.

[163] Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. 2014. Correlating events with time series for incident diagnosis. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1583–1592.

[164] Yingzhe Lyu, Heng Li, Mohammed Sayagh, Zhen Ming Jiang, and Ahmed E Hassan. 2021. An empirical study of the impact of data splitting decisions on the performance of AIOps solutions. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 4 (2021), 1–38.

[165] Yingzhe Lyu, Gopi Krishnan Rajbahadur, Dayi Lin, Boyuan Chen, and Zhen Ming Jiang. 2021. Towards a consistent interpretation of aiops models. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 1 (2021), 1–38.

[166] Meng Ma and Zhu Mao. 2020. Deep-convolution-based LSTM network for remaining useful life prediction. *IEEE Transactions on Industrial Informatics* 17, 3 (2020), 1658–1667.

[167] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1176–1189.

[168] Amirabbas Majd, Mojtaba Vahidi-Asl, Alireza Khalilian, Ahmad Baraani-Dastjerdi, and Bahman Zamani. 2019. Code4Bench: A multidimensional benchmark of Codeforces data for different program analysis techniques. *Journal of Computer Languages* 53 (2019), 38–52.

[169] Amirabbas Majd, Mojtaba Vahidi-Asl, Alireza Khalilian, Pooria Poorsarvi-Tehrani, and Hassan Haghighi. 2020. SLDeep: Statement-level software defect prediction using deep-learning model on static code features. *Expert Systems with Applications* 147 (2020), 113156.

[170] Adetokunbo Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2011. A lightweight algorithm for message type extraction in system application logs. *IEEE Transactions on Knowledge and Data Engineering* 24, 11 (2011), 1921–1936.

[171] Evan K Maxwell, Godmar Back, and Naren Ramakrishnan. 2010. Diagnosing memory leaks using graph mining on heap dumps. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 115–124.

[172] Thomas J McCabe. 1976. A complexity measure. *IEEE Transactions on software Engineering* 4 (1976), 308–320.

[173] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.. In *IJCAI*, Vol. 19. 4739–4745.

[174] Tim Menzies. 2019. The five laws of SE for AI. *IEEE Software* 37, 1 (2019), 81–85.

[175] Tim Menzies, Jeremy Greenwald, and Art Frank. 2006. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering* 33, 1 (2006), 2–13.

[176] Montassar Ben Messaoud, Asma Miladi, Ilyes Jenhani, Mohamed Wiem Mkaouer, and Lobna Ghadhab. 2022. Duplicate bug report detection using an attention-based neural language model. *IEEE Transactions on Reliability* (2022).

[177] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th Conference on Program Comprehension*. 167–177.

[178] Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. 2015. A large-scale study of flash memory failures in the field. *ACM SIGMETRICS Performance Evaluation Review* 43, 1 (2015), 177–190.

[179] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[180] Seyed Ali Mirheidari, Sajjad Arshad, and Rasool Jalili. 2013. Alert correlation algorithms: A survey and taxonomy. In *Cyberspace Safety and Security: 5th International Symposium, CSS 2013, Zhangjiajie, China, November 13-15, 2013, Proceedings 5*. Springer, 183–197.

[181] Natwar Modani, Rajeev Gupta, Guy M. Lohman, Tanveer Fathima Syeda-Mahmood, and Laurent Mignet. 2007. Automatically Identifying Known Software Problems. In *Proceedings of the 23rd International Conference on Data Engineering Workshops*. IEEE Computer Society, 433–441.

[182] Christoph Molnar. 2019. *Interpretable Machine Learning*.

[183] Akira Moroo, Akiko Aizawa, and Takayuki Hamamoto. 2017. Reranking-based Crash Report Deduplication. In *The 29th International Conference on Software Engineering and Knowledge Engineering*, Xudong He (Ed.). KSI Research Inc. and Knowledge Systems Institute Graduate School, 507–510.

[184] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering*. 181–190.

[185] Mukosi Abraham Mukwevho and Turgay Celik. 2018. Toward a smart cloud: A review of fault-tolerance methods in cloud systems. *IEEE Transactions on Services Computing* 14, 2 (2018), 589–605.

[186] G Murphy and Davor Cubranic. 2004. Automatic bug triage using text categorization. In *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. Citeseer, 1–6.

[187] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. 2006. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*. 452–461.

[188] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. 2013. Transfer defect learning. In *2013 35th international conference on software engineering (ICSE)*. IEEE, 382–391.

[189] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B Dasgupta, and Subhrajit Bhattacharya. 2016. Anomaly detection using program control flow graph mining from execution logs. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 215–224.

[190] Roberto Natella, Domenico Cotroneo, Joao A Duraes, and Henrique S Madeira. 2012. On fault representativeness of software fault injection. *IEEE Transactions on Software Engineering* 39, 1 (2012), 80–96.

[191] Sasho Nedelkoski, Jasmin Bogatinovski, Ajay Kumar Mandapati, Soeren Becker, Jorge Cardoso, and Odej Kao. 2020. Multi-source distributed system data for ai-powered analytics. In *Service-Oriented and Cloud Computing: 8th IFIP WG 2.14 European Conference, ESOCC 2020, Heraklion, Crete, Greece, September 28–30, 2020, Proceedings 8*. Springer, 161–176.

[192] Saul B Needleman and Christian D Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology* 48, 3 (1970), 443–453.

[193] S Nessa, M Abedin, W Eric Wong, L Khan, and Y Qi. 2009. Fault localization using N-gram analysis. In *Proceedings of the 3rd International Conference on Wireless Algorithms, Systems, and Applications*. 548–559.

[194] Hiep Nguyen, Zhiming Shen, Yongmin Tan, and Xiaohui Gu. 2013. FChain: Toward black-box online fault localization for cloud systems. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 21–30.

[195] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. 2020. A Systematic Mapping Study in AIOps. *arXiv preprint arXiv:2012.09108* (2020).

[196] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. 2021. A survey of AIOps methods for failure management. *ACM Transactions on Intelligent Systems and Technology (TIST)* 12, 6 (2021), 1–45.

[197] Celestino Ordóñez, Fernando Sánchez Lasheras, Javier Roca-Pardiñas, and Francisco Javier de Cos Juez. 2019. A hybrid ARIMA–SVM model for the study of the remaining useful life of aircraft engines. *J. Comput. Appl. Math.* 346 (2019), 184–191.

[198] Thomas J Ostrand, Elaine J Weyuker, and Robert M Bell. 2005. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering* 31, 4 (2005), 340–355.

[199] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[200] Cláudia Pascoal, M Rosario De Oliveira, Rui Valadas, Peter Filzmoser, Paulo Salvador, and António Pacheco. 2012. Robust feature selection and robust PCA for internet traffic anomaly detection. In *2012 Proceedings Ieee Infocom*. IEEE, 1755–1763.

[201] Eduardo HM Pena, Luiz F Carvalho, Sylvio Barbon Jr, Joel JPC Rodrigues, and Mario Lemes Proença Jr. 2017. Anomaly detection using the correlational paraconsistent machine with digital signatures of network segment. *Information Sciences* 420 (2017), 313–328.

[202] Phuong Pham, Vivek Jain, Lukas Dauterman, Justin Ormont, and Navendu Jain. 2020. DeepTriage: Automated Transfer Assistance for Incidents in Cloud Services. In *ACM SIGKDD*. 3281–3289.

[203] Natthakul Pingclasai, Hideaki Hata, and Ken-ichi Matsumoto. 2013. Classifying bug reports to bugs and other requests using topic modeling. In *2013 20Th asia-pacific software engineering conference (APSEC)*, Vol. 2. IEEE, 13–18.

[204] Teerat Pitakrat, Dušan Okanović, André van Hoorn, and Lars Grunske. 2018. Hora: Architecture-aware online failure prediction. *Journal of Systems and Software* 137 (2018), 669–685.

[205] Pankaj Prasad and Charley Rich. 2018. Market Guide for AIOps Platforms. *Retrieved March* 12 (2018), 2020.

[206] Xianping Qu and Jingjing Ha. 2017. Next generation of devops: Aiops in practice@ baidu. *SREcon17* (2017).

[207] Baishakhi Ray, Vincent Hellendoorn, Saheel Godhane, Zhaopeng Tu, Alberto Bacchelli, and Premkumar Devanbu. 2016. On the" naturalness" of buggy code. In *Proceedings of the 38th International Conference on Software Engineering*. 428–439.

[208] Lena Reiter and FH Wedel. 2021. AIOps–A Systematic Literature Review. (2021).

[209] Youcef Remil, Anes Bendimerad, Mathieu Chambard, Romain Mathonat, Marc Plantevit, and Mehdi Kaytoue. 2023. Subjectively Interesting Subgroups with Hierarchical Targets: Application to Java Memory Analysis. In *International Conference on Data Mining Workshops, ICDMW*. IEEE.

[210] Youcef Remil, Anes Bendimerad, Romain Mathonat, Philippe Chaleat, and Mehdi Kaytoue. 2021. "What makes my queries slow?": Subgroup Discovery for SQL Workload Analysis. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 642–652.

[211] Youcef Remil, Anes Bendimerad, Romain Mathonat, Chedy Raissi, and Mehdi Kaytoue. 2024. DeepLSH: Deep Locality-Sensitive Hash Learning for Fast and Efficient Near-Duplicate Crash Report Detection. In *46th International Conference on Software Engineering, ICSE*.

[212] Youcef Remil, Anes Bendimerad, Marc Plantevit, Céline Robardet, and Mehdi Kaytoue. 2021. Interpretable Summaries of Black Box Incident Triaging with Subgroup Discovery. In *8th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2021, Porto, Portugal, October 6-9, 2021*. IEEE, 1–10.

[213] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3009–3017.

[214] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B. Gupta, Xiaojiang Chen, and Xin Wang. 2022. A Survey of Deep Active Learning. *ACM Comput. Surv.* 54, 9 (2022), 180:1–180:40.

[215] Manos Renieres and Steven P Reiss. 2003. Fault localization with nearest neighbor queries. In *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.* IEEE, 30–39.

[216] Laxmi Rijal, Ricardo Colomo-Palacios, and Mary Sánchez-Gordón. 2022. Aiops: A multivocal literature review. *Artificial Intelligence for Cloud and Edge Computing* (2022), 31–50.

[217] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. 2007. Sensitivity of PCA for traffic anomaly detection. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. 109–120.

[218] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. 2007. Detection of duplicate defect reports using natural language processing. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 499–510.

[219] Korosh Koochekian Sabor, Abdelwahab Hamou-Lhadj, and Alf Larsson. 2017. DURFEX: A Feature Extraction Technique for Efficient Detection of Duplicate Bug Reports. In *2017 IEEE International Conference on Software Quality, Reliability and Security, QRS 2017*. 240–250.

[220] Felix Salfner, Maren Lenk, and Miroslaw Malek. 2010. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)* 42, 3 (2010), 1–42.

[221] Areeg Samir and Claus Pahl. 2019. A controller architecture for anomaly detection, root cause analysis and self-adaptation for cluster architectures. In *Intl Conf Adaptive and Self-Adaptive Systems and Applications*.

[222] Mark Sanderson. 2010. Test Collection Based Evaluation of Information Retrieval Systems. *Found. Trends Inf. Retr.* 4, 4 (2010), 247–375.

[223] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. 2019. e-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice

platforms. In *The World Wide Web Conference*. 3215–3222.

[224] Qihong Shao, Yi Chen, Shu Tao, Xifeng Yan, and Nikos Anerousis. 2008. Efficient ticket routing by resolution sequence mining. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 605–613.

[225] Bikash Sharma, Praveen Jayachandran, Akshat Verma, and Chita R Das. 2013. CloudPD: Problem determination and diagnosis in shared dynamic clouds. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 1–12.

[226] Shijun Shen, Jiuling Zhang, Daochao Huang, and Jun Xiao. 2020. Evolving from Traditional Systems to AIOps: Design, Implementation and Measurements. In *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*. IEEE, 276–280.

[227] Akbar Siami Namin, James H Andrews, and Duncan J Murdoch. 2008. Sufficient mutation operators for measuring test effectiveness. In *Proceedings of the 30th international conference on Software engineering*. 351–360.

[228] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. 2017. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 1067–1075.

[229] Jeongju Sohn and Shin Yoo. 2017. Fluccs: Using code and change metrics to improve fault localization. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 273–283.

[230] Marc Solé, Victor Muntés-Mulero, Annie Ibrahim Rana, and Giovani Estrada. 2017. Survey on models and techniques for root-cause analysis. *arXiv preprint arXiv:1701.08546* (2017).

[231] Vladimir Sor and Satish Narayana Srirama. 2014. Memory leak detection in Java: Taxonomy and classification of approaches. *J. Syst. Softw.* 96 (2014), 139–151.

[232] Charles Spearman. 1961. The proof and measurement of association between two things. (1961).

[233] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2828–2837.

[234] Ya Su, Youjian Zhao, Wentao Xia, Rong Liu, Jiahao Bu, Jing Zhu, Yuanpu Cao, Haibin Li, Chenhao Niu, Yiyin Zhang, et al. 2019. CoFlux: robustly correlating KPIs by fluctuations for service troubleshooting. In *Proceedings of the International Symposium on Quality of Service*. 1–10.

[235] Chapram Sudhakar, Ishan Shah, and T Ramesh. 2014. Software rejuvenation in cloud systems using neural networks. In *2014 International Conference on Parallel, Distributed and Grid Computing*. IEEE, 230–233.

[236] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 253–262.

[237] Xiaoyi Sun, Krishnendu Chakrabarty, Ruirui Huang, Yiquan Chen, Bing Zhao, Hai Cao, Yinhe Han, Xiaoyao Liang, and Li Jiang. 2019. System-level hardware failure prediction using deep learning. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[238] Yongqian Sun, Youjian Zhao, Ya Su, Dapeng Liu, Xiaohui Nie, Yuan Meng, Shiwen Cheng, Dan Pei, Shenglin Zhang, Xianping Qu, et al. 2018. Hotspot: Anomaly localization for additive kpis with multi-dimensional attributes. *IEEE Access* 6 (2018), 10909–10923.

[239] Ashish Sureka and Pankaj Jalote. 2010. Detecting duplicate bug report using character n-gram-based features. In *2010 Asia Pacific software engineering conference*. IEEE, 366–374.

[240] Zhiyuan Tan, Aruna Jamdagni, Xiangjian He, Priyadarsi Nanda, and Ren Ping Liu. 2013. A system for denial-of-service attack detection based on multivariate correlation analysis. *IEEE transactions on parallel and distributed systems* 25, 2 (2013), 447–456.

[241] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 785–794.

[242] Yuan Tian, David Lo, and Chengnian Sun. 2012. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In *2012 19th Working Conference on Reverse Engineering*. IEEE, 215–224.

[243] Yuan Tian, David Lo, and Chengnian Sun. 2013. Drone: Predicting priority of reported bugs by multi-factor analysis. In *2013 IEEE International Conference on Software Maintenance*. IEEE, 200–209.

[244] Md Nasir Uddin, Bixin Li, Zafar Ali, Pavlos Kefalas, Inayat Khan, and Islam Zada. 2022. Software defect prediction employing BiLSTM and BERT-based semantic feature. *Soft Computing* 26, 16 (2022), 7877–7891.

[245] Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*. Ieee, 119–126.

[246] Kalyanaraman Vaidyanathan and Kishor S Trivedi. 1999. A measurement-based model for estimation of resource exhaustion in operational software systems. In *Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No. PR00443)*. IEEE, 84–93.

[247] Owen Vallis, Jordan Hochenbaum, and Arun Kejariwal. 2014. A Novel Technique for {Long-Term} Anomaly Detection in the Cloud. In *6th USENIX workshop on hot topics in cloud computing (HotCloud 14).*

[248] Roman Vasiliev, Dmitrij V. Koznov, George A. Chernishev, Aleksandr Khvorov, Dmitry V. Luciv, and Nikita Povarov. 2020. TraceSim: a method for calculating stack trace similarity. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, FSE 2020.* 25–30.

[249] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. *Advances in neural information processing systems* 29 (2016).

[250] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. 2010. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*. 193–204.

[251] Jing Wang, Daniel Rossell, Christos G Cassandras, and Ioannis Ch Paschalidis. 2013. Network anomaly detection: A survey and comparative analysis of stochastic and deterministic methods. In *52nd IEEE Conference on Decision and Control*. IEEE, 182–187.

[252] Qing Wang, Wubai Zhou, Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. 2017. Constructing the knowledge base for cognitive it service management. In *2017 IEEE International Conference on Services Computing (SCC)*. IEEE, 410–417.

[253] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering*. 297–308.

[254] Song Wang, Wen Zhang, and Qing Wang. 2014. FixerCache: Unsupervised caching active developers for diverse bug triage. In *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement*. 1–10.

[255] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. 2017. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE international conference on intelligence and security informatics (ISI)*. IEEE, 43–48.

[256] Xuanrun Wang, Kanglin Yin, Qianyu Ouyang, Xidao Wen, Shenglin Zhang, Wenchi Zhang, Li Cao, Jiuxue Han, Xing Jin, and Dan Pei. 2022. Identifying Erroneous Software Changes through Self-Supervised Contrastive Learning on Time Series Data. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 366–377.

[257] Yu Wang, Qiang Miao, Eden WM Ma, Kwok-Leung Tsui, and Michael G Pecht. 2013. Online anomaly detection for hard disk drives based on mahalanobis distance. *IEEE Transactions on Reliability* 62, 1 (2013), 136–145.

[258] Frank Wilcoxon. 1992. *Individual comparisons by ranking methods*. Springer.

[259] W Eric Wong, Vidroha Debroy, Ruizhi Gao, and Yihao Li. 2013. The DStar method for effective software fault localization. *IEEE Transactions on Reliability* 63, 1 (2013), 290–308.

[260] W Eric Wong, Vidroha Debroy, and Dianxiang Xu. 2011. Towards better fault localization: A crosstab-based statistical approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 3 (2011), 378–396.

[261] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740.

[262] Stefan Wrobel. 1997. An Algorithm for Multi-relational Discovery of Subgroups. In *Principles of Data Mining and Knowledge Discovery, First European Symposium, PKDD '97, Trondheim, Norway, June 24-27, 1997, Proceedings*, Vol. 1263. Springer, 78–87.

[263] Di Wu, Yiping Ke, Jeffrey Xu Yu, Philip S Yu, and Lei Chen. 2010. Detecting leaders from correlated time series. In *Database Systems for Advanced Applications: 15th International Conference, DASFAA 2010, Tsukuba, Japan, April 1-4, 2010, Proceedings, Part I 15*. Springer, 352–367.

[264] Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, and Sunghun Kim. 2014. CrashLocator: locating crashing faults based on crash stacks. In *International Symposium on Software Testing and Analysis, ISSTA*. 204–214.

[265] Yuting Wu, Mei Yuan, Shaopeng Dong, Li Lin, and Yingqi Liu. 2018. Remaining useful life estimation of engineered systems using vanilla LSTM neural networks. *Neurocomputing* 275 (2018), 167–179.

[266] Sheng-Qu Xi, Yuan Yao, Xu-Sheng Xiao, Feng Xu, and Jian Lv. 2019. Bug triaging based on tossing sequence modeling. *Journal of Computer Science and Technology* 34 (2019), 942–956.

[267] Bin Xia, Yuxuan Bai, Junjie Yin, Yun Li, and Jian Xu. 2021. Loggan: a log-level generative adversarial network for anomaly detection using permutation event modeling. *Information Systems Frontiers* 23 (2021), 285–298.

[268] Xin Xia, David Lo, Ying Ding, Jafar M Al-Kofahi, Tien N Nguyen, and Xinyu Wang. 2016. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering* 43, 3 (2016), 272–297.

[269] Jiang Xiao, Zhuang Xiong, Song Wu, Yusheng Yi, Hai Jin, and Kan Hu. 2018. Disk failure prediction in data centers via online learning. In *Proceedings of the 47th International Conference on Parallel Processing*. 1–10.

[270] Miao Xie, Jiankun Hu, Song Guo, and Albert Y Zomaya. 2016. Distributed segment-based anomaly detection with Kullback–Leibler divergence in wireless sensor networks. *IEEE Transactions on Information Forensics and Security* 12, 1 (2016), 101–110.

[271] Qi Xie, Zhiyuan Wen, Jieming Zhu, Cuiyun Gao, and Zibin Zheng. 2018. Detecting duplicate bug reports with convolutional neural networks. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 416–425.

[272] Chang Xu, Gang Wang, Xiaoguang Liu, Dongdong Guo, and Tie-Yan Liu. 2016. Health status assessment and failure prediction for hard drives with recurrent neural networks. *IEEE Trans. Comput.* 65, 11 (2016), 3502–3508.

[273] Guoqing Xu and Atanas Rountev. 2013. Precise memory leak detection for java software using container profiling. *ACM Trans. Softw. Eng. Methodol.* 22, 3 (2013), 17:1–17:28.

[274] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. 2018. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 world wide web conference*. 187–196.

[275] Jiaxi Xu, Fei Wang, and Jun Ai. 2020. Defect prediction with semantics and context features of codes based on graph representation learning. *IEEE Transactions on Reliability* 70, 2 (2020), 613–625.

[276] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 117–132.

[277] Jifeng Xuan, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou, Zhongxuan Luo, and Xindong Wu. 2014. Towards effective bug triage with software data reduction techniques. *IEEE transactions on knowledge and data engineering* 27, 1 (2014), 264–280.

[278] Jifeng Xuan, He Jiang, Zhilei Ren, Jun Yan, and Zhongxuan Luo. 2010. Automatic Bug Triage using Semi-Supervised Text Classification.. In *SEKE*. 209–214.

[279] Jifeng Xuan and Martin Monperrus. 2014. Learning to combine multiple ranking metrics for fault localization. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 191–200.

[280] Geunseok Yang, Tao Zhang, and Byungjeong Lee. 2014. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *2014 IEEE 38th Annual Computer Software and Applications Conference*. IEEE, 97–106.

[281] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. 2019. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 647–657.

[282] Zunwen You, Zengchang Qin, and Zheng Zheng. 2012. Statistical fault localization using execution sequence. In *2012 International Conference on Machine Learning and Cybernetics*, Vol. 3. IEEE, 899–905.

[283] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2672–2681.

[284] Chunqiu Zeng, Wubai Zhou, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. 2017. Knowledge guided hierarchical multi-label classification over ticket data. *IEEE Transactions on Network and Service Management* 14, 2 (2017), 246–260.

[285] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V Chawla. 2019. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 1409–1416.

[286] Jie Zhang, Xiaoyin Wang, Dan Hao, Bing Xie, Lu Zhang, and Hong Mei. 2015. A survey on bug-report analysis. *Sci. China Inf. Sci.* 58, 2 (2015), 1–24.

[287] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. 2016. Automated IT system failure prediction: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1291–1300.

[288] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, et al. 2017. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[289] Sai Zhang and Congle Zhang. 2014. Software bug localization with markov logic. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 424–427.

[290] Tao Zhang, He Jiang, Xiapu Luo, and Alvin TS Chan. 2016. A literature review of research in bug resolution: Tasks, challenges and future directions. *Comput. J.* 59, 5 (2016), 741–773.

[291] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[292] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, et al. 2020. Understanding and handling alert storm for online service systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. 162–171.

[293] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, et al. 2020. Real-time incident prediction for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 315–326.

[294] Nengwen Zhao, Junjie Chen, Zhaoyang Yu, Honglin Wang, Jiesong Li, Bin Qiu, Hongyu Xu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2021. Identifying bad software changes via multimodal anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 527–539.

[295] Nengwen Zhao, Panshi Jin, Lixin Wang, Xiaoqin Yang, Rong Liu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Automatically and adaptively identifying severe alerts for online service systems. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2420–2429.

[296] Nengwen Zhao, Honglin Wang, Zeyan Li, Xiao Peng, Gang Wang, Zhu Pan, Yong Wu, Zhen Feng, Xidao Wen, Wenchi Zhang, et al. 2021. An empirical investigation of practical log anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1404–1415.

[297] Ying Zhao, Xiang Liu, Siqing Gan, and Weimin Zheng. 2010. Predicting disk failures with HMM-and HSMM-based approaches. In *Advances in Data Mining. Applications and Theoretical Aspects: 10th Industrial Conference, ICDM 2010, Berlin, Germany, July 12-14, 2010. Proceedings 10*. Springer, 390–404.

[298] Shuai Zheng, Kosta Ristovski, Ahmed Farahat, and Chetan Gupta. 2017. Long short-term memory network for remaining useful life estimation. In *2017 IEEE international conference on prognostics and health management (ICPHM)*. IEEE, 88–95.

[299] Jian Zhou and Hongyu Zhang. 2012. Learning to rank duplicate bug reports. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. 852–861.

[300] Wubai Zhou, Liang Tang, Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. 2016. Resolution recommendation for event tickets in service management. *IEEE Transactions on Network and Service Management* 13, 4 (2016), 954–967.

A LIST OF ABBREVIATIONS

Abbreviation	Meaning
ACID	Atomicity, Consistency, Isolation, Durability
ACODS	Ant Colony Optimization for Digital Signature
AE	Auto Encoder
AIOPS	Artificial Intelligence for IT Operations
ARIMA	Auto Regressive Integrated Moving Average
ASH	Active Session History
AST	Abstract Syntax Tree
AUC	Area Under the Curve
BERT	Bidirectional Encoder Representations from Transformers
BFOA	Bacterial Foraging Optimization Algorithm
CAR	Collaborative Alert Ranking
CART	Classification and Regression Trees
CBN	Causal Bayesian Network
CBO	Coupling Between Objects
CNN	Convolutional Neural Network
CPM	Correlational Paraconsistent Machine
ConvLSTM	Convolutional Long Short-Term Memory
DBMS	Database Management System
DBN	Deep Belief Network
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DIFT	Dynamic Information Flow Tracking
DIT	Depth of Inheritance Tree
DRAM	Dynamic Random Access Memory
DSL	Domain-Specific Language
DSNSF	Digital Signatures of Network Segment using Flow Analysis
DTW	Dynamic Time Warping
DoS	Denial of Service
EM	Expectation-Maximization
ETL	Extract, Transform, Load
FCA	Formal Concept Analysis
FF	Frequency-Factorization
FFT	Fast Fourier Transform
FPM	Frequent Pattern Mining
FSA	Finite State Automaton
FSM	Finite State Machine
FT-Tree	Frequent Template Tree
GLM	Generalized Linear Model
GNN	Graph Neural Network
GP	Genetic Programming
GRU	Gated Recurrent Unit
HDFS	Hadoop Distributed File System
HMM	Hidden Markov Model
HMRF	Hidden Markov Random Field
HsMM	Hierarchical state-based Markov Model
IDC	Internet Data Center
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
ITOA	Information Technology Operations Analytics

Abbreviation	**Meaning**
K-NN	k-Nearest Neighbors
KPI	Key Performance Indicator
LCOM	Lack of Cohesion in Methods
LDA	Latent Dirichlet Allocation
LIME	Local Interpretable Model-Agnostic Explanations
LOC	Lines of Code
LSTM	Long Short-Term Memory
MLP	Multi-Layer Perceptron
MRMR	Minimum Redundancy Maximum Relevance
MTM	Multi-Feature Topic Model
MTTx	Mean Time To X
NOM	Number of Methods
OLAP	Online Analytical Processing
PCA	Principal Component Analysis
PDF	Probability Density Functions
PL	Paraconsistent Logic
PRC	Precision-Recall Curve
PSO	Particle Swarm Optimization
Qos	Quality of Service
RCA	Root Cause Analysis
RFC	Response for a Class
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
RUL	Remaining Useful Life
SDP	Software Defect Prediction
SLO	Service Level Objective
SMART	Self-Monitoring, Analysis and Reporting Technology
SMM	Semi Markov Model
SR	Spectral Residual
TAN	Tree Augmented Naive Bayes
TCNN	Temporal Convolutional Neural Network
TF-IDF	Term Frequency-Inverse Document Frequency
TL	Transfer Learning
TOPSIS	Technique for Order of Preference by Similarity to Ideal Solution
WMC	Weighted Methods per Class
WSN	Wireless Sensor Network
XAI	Explainable Artificial Intelligence