Enhancing DevOps with Autonomous Monitors: A Proactive Approach to Failure Detection

Hrusto, Adha

2024

Document Version:
Publisher's PDF, also known as Version of record

Enhancing DevOps with Autonomous Monitors: A Proactive Approach to Failure Detection

Adha Hrusto

Doctoral Dissertation, 2024
Department of Computer Science
Lund University

This thesis is submitted to the Research Education Board of the Faculty of Engineering at Lund University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering.

Department of Computer Science
Faculty of Engineering
Lund University

Dissertation 76, 2024
LU-CS-DISS: 2024-03
ISSN: 1404-1219

ISBN 978-91-8104-209-2 (printed version)
ISBN 978-91-8104-210-8 (electronic version)

Printed in Sweden by Tryckeriet i E-huset, Lund, 2024

ABSTRACT

Software engineering practices, including continuous integration, continuous testing, and continuous deployment, aim to streamline and automate the software development process. A cultural and professional movement that builds upon continuous practices, *DevOps*, seeks to bridge the gap between development and operations. By fostering a collaborative environment, DevOps supports faster, more frequent, and reliable software releases, inherently promoting agile methodologies throughout the software development lifecycle.

By introducing *agility*, there is a higher risk of operational failures in cloud-based software systems. Recognizing this challenge, the objective of this thesis is to understand and present approaches for mitigating the cascading effects of operational failures across interconnected system components. In collaboration with two Swedish companies, we investigated how proactive monitoring strategies inspired by state-of-the-art machine learning (ML) solutions can prevent failure propagation and ensure seamless system operations.

The conducted research activities span from practice to theory and from problem to solution domain, including problem conceptualization, solution design, instantiation, and empirical validation. This complies with the main principles of the *design science* paradigm mainly used to frame problem-driven studies aiming to improve specific areas of practice.

The main contributions of this thesis are threefold. First, an in-depth overview of operational challenges and matching solutions in cloud-based software systems, focusing on alert management and monitoring data through two case studies and extensive literature reviews. Second, a *proactive alert strategy* called *autonomous monitors* to enhance early detection and prevention of operational *failures*. Finally, the practical applicability of these monitors is confirmed via empirical studies, highlighting their effectiveness in various industrial contexts.

We demonstrated the practical effectiveness of the proposed ML-based monitoring solution to pave the way for its widespread adoption for enhancing DevOps.

POPULAR SUMMARY

THE SECRET TO A SMOOTH DIGITAL EXPERIENCE

Adha Hrusto, Department of Computer Science, Lund University

In today's digital world, everyone expects websites and applications to work seamlessly without any glitches or downtime. But behind every click, swipe, and tap is a complex infrastructure that needs to run smoothly. However, ensuring this smooth digital experience is a complex challenge. Imagine this as a bustling city where data, like people, moves around seamlessly. Yet, just like in any city, things can go wrong – traffic jams (slowdowns), power outages (crashes), or roadblocks (errors). My research focuses on creating an intelligent system that acts as the world's best city planner and traffic monitor rolled into one, ensuring everything in our digital city flows perfectly.

Traditional methods wait for a problem to happen before fixing it, like waiting for a car to break down in the middle of the road before deciding to do regular maintenance checks. This reactive approach can be costly and frustrating for everyone relying on these digital services. In my work, I propose and implement a smarter, proactive strategy. It is a technical solution that predicts where and when the digital city's flow might get disrupted, allowing us to fix potential problems before they even occur and escalate. This is like having sensors and cameras all over the city that alert the maintenance crew to a potential issue, a flickering street light or a developing pothole, so they can address it during the night, preventing any disruption to the city's daily life.

This innovative approach brings several benefits. For users, it means fewer interruptions while shopping online, streaming videos, or scrolling through social media, creating a smoother and more enjoyable digital experience. For businesses, it translates into higher customer satisfaction and trust, including savings from avoiding the high costs associated with fixing emergencies and compensating disappointed customers.

Looking forward, the implications of this research extend beyond just keeping our digital experiences smooth. They lead to a future where technology self-

adjusts and improves continuously, making our reliance on digital services more seamless and integrated into our lives. Just as urban planners dream of cities where traffic flows without congestion and services are always available, my research envisions a digital world free from disruptions, where technology anticipates and meets our needs effortlessly.

As we move towards this future, my research work opens up exciting possibilities for how we design, manage, and interact with the digital landscapes that have become central to our daily lives. The secret to a smooth digital experience resides in anticipating and preventing problems before they arise, ensuring our digital world is as resilient, reliable, and responsive as the best-planned cities.

ACKNOWLEDGEMENTS

This thesis was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

I express my deepest gratitude and appreciation to Prof. Per Runeson, Dr. Emelie Engström, and Dr. Magnus C Ohlsson, whose expertise and mentorship have been invaluable throughout my Ph.D. journey. It has been a privilege to have you as my supervisors and to collaborate with you on the studies.

I would also like to thank my colleagues from the Department of Computer Science, especially the SERG members, for inspiring discussions and very pleasant lunches and fikas. Song, Sergio, Matthias, Konstantin, Momina, Ayesha, and Idriss, thank you for making this journey even more wonderful. Special thanks to my inspiring colleague and friend Alma Oručević-Alagić for her kindness and encouragement.

I extend my sincere gratitude to System Verification, the company where I have always been warmly welcomed, appreciated, and encouraged to pursue my personal and professional goals. In particular, I would like to thank Kadira Šubo and Henrik Sällman for initiating and encouraging my application for the doctoral program, as well as my immediate manager Andreas Axelsson and HR manager Joanna Doweyko for continuous support in both professional and personal matters.

I had an opportunity to collaborate with the DevOps, development, and test teams from System Verification and two anonymous case companies. I am thankful for their willingness to share insights and answer my questions. Special appreciation goes to Tobias Anderson for his exceptional guidance in the technical aspects of the project.

I am extremely grateful to my family and friends for their unconditional love, unwavering support, and belief in me. Their encouragement and faith have been my guiding light through challenging times.

Finally, all praises and thanks are due to God, the Almighty, for His blessings and knowledge granted to me.

Tack så mycket!
Adha Hrusto

LIST OF PUBLICATIONS

This thesis consists of an introduction and a compilation of five papers listed below and referred to by Roman numerals.

Publications Included in the Thesis

I **Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations**
Adha Hrusto, Per Runeson, Emelie Engström
Springer Nature Computer Science 2, 447 (2021)
doi: 10.1007/s42979-021-00826-y

II **Towards Optimization of Anomaly Detection in DevOps**
Adha Hrusto, Emelie Engström, Per Runeson
Information and Software Technology, 160 (2023)
doi: 10.1016/j.infsof.2023.107241

III **Autonomous Monitors for Detecting Failures Early and Reporting Interpretable Alerts in Cloud Operations**
Adha Hrusto, Per Runeson, Magnus C Ohlsson
In Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2024)
doi: 10.1145/3639477.3639712

IV **Advancing Software Monitoring: An Industry Survey on ML-Driven Alert Management Strategies**
Adha Hrusto, Per Runeson, Emelie Engström, Magnus C Ohlsson
In Proceedings of the 50th Euromicro Conference Series on Software Engineering and Advanced Applications:
Practical Aspects of Software Engineering (SEAA-KKIO 2024)
doi: 10.1109/SEAA64295.2024.00073

V **Monitoring Data for Anomaly Detection in Cloud-Based Systems: A Systematic Mapping Study**
Adha Hrusto, Nauman bin Ali, Emelie Engström, Yuqing Wang
Submitted to TOSEM.

Related Publications

These papers are referenced in the introduction chapter of this thesis.

VI **Optimization of Anomaly Detection in a Microservice System Through Continuous Feedback from Development**
Adha Hrusto, Emelie Engström, Per Runeson
In Proceedings of the 10th ACM/IEEE International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems (SESoS 2022).
doi: 10.1145/3528229.3529382

Contribution statement

All papers included in this thesis have been co-authored with other researchers. The authors' individual contributions to Papers I-V are as follows:

Paper I
Adha Hrusto, Prof. Per Runeson, and Dr. Emelie Engström initiated study and collaboration with the industry partner. All authors contributed in conducting interviews with the practitioners and formulating identified problem instances. Adha Hrusto has led the solution design and implementation of the solution prototype while the decisions and actions taken were critically assessed by Prof. Per Runeson and Dr. Emelie Engström. Adha Hrusto was mainly responsible for writing the paper, while Prof. Per Runeson and Dr. Emelie Engström have also contributed to writing, reviewing, and editing.

Paper II
Adha Hrusto designed the study and as a lead author, was responsible for writing the paper. Dr. Emelie Engström initiated very important discussions about the research approach, while Prof. Per Runeson assisted in refining and formulating design science concepts of the study. Adha Hrusto proposed reviewing deep learning approaches for advancing solution design and was responsible for its in-context implementation and evaluation. Dr. Emelie Engström and Prof. Per Runeson have also contributed to reviewing and editing the paper.

Paper III
Adha Hrusto initiated collaboration with a new industry partner and was solely included in all discussions with practitioners in the case company. Adha Hrusto has also led the design, implementation, and evaluation of the proposed solution. All questions and decisions were discussed in regular meetings with co-authors. The paper was written by Adha, while co-authors mainly contributed to reviewing and editing.

Paper IV
The survey study was initiated by Adha Hrusto, the main author, who has also entirely contributed to designing and distributing the questionnaire to the target sample. The questions included in a questionnaire and survey setup were evaluated by co-authors before publishing the final version of the survey instrument. Adha Hrusto analyzed the survey results and wrote the paper, while the co-authors were responsible for reviewing and editing.

Paper V
Adha Hrusto initiated and led the systematic mapping study, taking charge of its setup and progression. The most important steps regarding methodology, review process, and data extraction and analysis were collaboratively decided with co-authors Dr. Emelie Engström, Dr. Nauman bin Ali and Yuqing Wang. All authors were actively involved in reading selected papers and extracting data, with the responsibilities evenly shared. Adha Hrusto was in charge of summarizing the extracted data from the reviewed papers and drafting the manuscript, whereas the co-authors focused on reviewing and editing the content.

CONTENTS

Introduction **1**
1 Background and Related Work 3
2 Research Approach 12
3 Summary of Results 17
4 Discussion 28
5 Validity of research 29
6 Conclusions 32

Included papers **35**

I Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations **37**
1 Introduction 38
2 Background and Related work 39
3 Research Approach 42
4 Case Description 44
5 Problem Conceptualization 46
6 Solution Design 48
7 Prototype Implementation and Empirical Validation 51
8 Discussion and Conclusion 54

II Towards Optimization of Anomaly Detection in DevOps **59**
1 Introduction 60
2 Background and Related Work 62
3 Research Approach 63
4 Problem Context 66
5 Review of DL Methods for Anomaly Detection in MTS 67
6 Guidance for a Minimum Feasible DL Method 70
7 Implementation and evaluation of anomaly detection approaches . 73
8 Conclusion 85

III Autonomous Monitors for Detecting Failures Early and Reporting Interpretable Alerts in Cloud Operations **89**
1 Introduction . . . 90
2 Background and related work . . . 91
3 Research approach . . . 93
4 Problem context . . . 94
5 Autonomous monitors . . . 95
6 Discussion . . . 105
7 Conclusion . . . 108

IV Advancing Software Monitoring: An Industry Survey on ML-Driven Alert Management Strategies **111**
1 Introduction . . . 112
2 Background and Related Work . . . 113
3 Research methodology . . . 114
4 Results . . . 118
5 Discussion and conclusion . . . 127

V Monitoring Data for Anomaly Detection in Cloud-Based Systems: A Systematic Mapping Study **129**
1 Introduction . . . 130
2 Background and Related Work . . . 131
3 Research Methodology . . . 135
4 Results and Analysis . . . 142
5 Discussion and Conclusion . . . 160

Bibliography **165**
References . . . 165

INTRODUCTION

Continuous software engineering (CSE) embraces continuous practices such as continuous integration, continuous testing, continuous deployment, continuous run-time monitoring, to support and enhance the entire software life-cycle all the way from the business strategy to the final software product. Another term, closely related to continuous practices and more often used in industry, *DevOps*, stands for cross-functional collaboration between development (Dev) and operations (Ops) and complements continuous practices by stressing the importance of *collaboration* and *culture* in software development environments. *Automation* and *monitoring* are additional dimensions of DevOps [145], also highly important within CSE. Thus, automation infrastructure is a means of achieving *continuity* of CSE activities while run-time monitoring in operations monitors the health and performance of the deployed software through continuous *measurements*. Accumulated monitoring data may provide particular insights into development, e.g., timely exposing unusual discrepancies in performance metrics. Accordingly, analyzing and managing operational data is of high importance for providing fast and valuable feedback to development, which was the focus of the research we conducted in this thesis.

While continuous integration and continuous deployment (CI/CD) practices significantly reduce the time to market for new features by enabling more frequent releases, they can introduce challenges in maintaining consistent quality due to insufficient testing [189]. This consideration is even more critical in the context of complex, cloud-based software ecosystems with possible dependencies on third-party services. Here, a minor quality issue in one component can trigger a domino effect, potentially compromising the functionality and reliability of the system as a whole. Thus, development and operation teams mainly rely on various monitoring tools and dashboards to discover bugs, glitches, or performance degradation before they escalate and cause more severe consequences. However, monitoring highly complex software systems, e.g., consisting of dozens of microservices, might be challenging as the volume and complexity of monitoring data increase over time with the software system evolution. Therefore, collected monitoring data in such systems may burden operations and cause information overflow, e.g., alert flooding or storm [252] in the feedback loop to development.

The main objective of this thesis is to understand and address challenges on the borderline between development and operations related to monitoring data overload and alert management with an overarching goal of enhancing DevOps practices. The solution focus was on applying state-of-the-art machine learning (ML) approaches in order to achieve *proactive monitoring*. This included continuous monitoring of multidimensional cloud-based software systems to identify and resolve potential issues before they impact performance or availability. For this purpose, anomaly detection (AD) techniques were employed to discover deviations in high-volume data [50, 96] and to identify early signs of operational failures. This was complemented with the prediction for root cause analysis by the large language model (GPT-3.5) utilizing log data. All relevant information regarding detected anomalies in specific performance metrics, including their current values, thresholds, and suggestions on how to approach resolving raised issues, were reported in the form of the *smart alert* to development. In this way, by timely reporting interpretable alerts, cascading effects of operational failures could be prevented.

The research was designed to integrate insights from both the practical challenges faced by the industry and theoretical perspectives from existing literature, aiming to address these challenges with ML solutions for proactive monitoring. A valuable aspect of the research project was the collaboration with two Swedish companies, providing a unique opportunity to conduct the studies in real-world industrial settings. These companies are referred to as *the first case company* and *the second case company* throughout the introduction for clarity and confidentiality. The first case company operates in the public transportation sector, focusing on ticket management and sales. The system analyzed here exemplifies a microservice architecture with distributed monitoring capabilities, offering a specific context to explore monitoring challenges and data management in operational settings. The second case company specializes in developing Product Information Management (PIM) software solutions. Their system enables customers to manage product information efficiently, enhancing their online sales and marketing strategies. Despite differences in functionality, this system shares cloud architecture and environmental similarities with the first, allowing for a comparative analysis that extends the research's applicability across different operational contexts.

Through these collaborations, the research was able to bridge theoretical knowledge with practical industrial applications, leading to a comprehensive exploration of ML-based solutions tailored to proactive monitoring within these two distinct yet technically interconnected environments. It combines novel frameworks for advanced alert management with practical engineering implementations to address monitoring challenges and data overload in continuous software engineering. Contributions of this thesis are threefold:

- **C1:** A comprehensive understanding of operational challenges in the context of large-scale software systems through two case studies. These studies led to the definition of problem constructs focusing on alert management

strategies and monitoring data usage, which is crucial for designing matching solutions. Extensive literature reviews enabled a theoretical understanding of the problem domain and provided the foundation for developing corresponding solutions.

- **C2:** A novel proactive alert strategy, named *autonomous monitors*, inspired by detailed observations of cloud-based software systems and most recent academic findings. This strategy aims to detect operational failures early and ensures continuous monitoring, enhancing the ability to predict and prevent anomalies before they disrupt operations.

- **C3:** The effectiveness and applicability of *autonomous monitors* were validated through empirical evaluations in two case companies and a survey assessing their use in various industrial settings. This research confirms the utility and adaptability of the proposed solution, marking a significant contribution to the field by demonstrating its potential benefits in real-world industrial, cloud-based environments.

1 Background and Related Work

This section provides a comprehensive overview of foundational concepts shaping modern software engineering according to the state-of-the-art literature. The scope of presented concepts spans from problem to solution domain, as studied in this thesis. The problem domain includes continuous engineering activities, with a primary focus on continuous monitoring and the challenges associated with balancing fast releases and software quality while considering the complexity and dimensionality of software systems. The solution domain examines proactive alert strategies based on anomaly detection, emphasizing the significance of early identification and mitigation of potential failures. This is particularly critical for high-dimensional software systems, where a failure in one component may trigger a cascade of issues impacting other components and the system's overall health. Accordingly, we present the most recent and relevant solutions in this domain, which guided our research towards practical implementations and evaluations within the studied context of two case companies.

1.1 Continuous Software Engineering

Continuous software engineering provides a set of continuous activities needed for an iterative software development process with accelerated delivery. As shown in Figure 1 by Fitzgerald and Stol [55], CSE spans across three domains: business strategy, development, and operations.

The software product creation starts with business strategy and planning, followed by technical implementation, deployment, and continuous execution in op-

erations. This is accomplished in several cycles, where BizDev and DevOps, connecting the three domains, ensure a continuous flow of information between *business strategy* and *development*, and *development* and *operations*, respectively. In this way, any discontinuities between sales/marketing expectations, development decisions, and users' satisfaction are eliminated [55].

DevOps gains more attention than BizDev as it is more frequently used within both academic and industrial circles. A common interpretation of DevOps [207] is that it is about culture, tooling, and processes and aims to improve collaboration and integration between team members, including developers, managers, operational personnel, and everyone involved in product creation and deployment. This means that DevOps is defined by four principles: collaboration, automation, measurement, and monitoring [146].

As shown in Figure 1, the three domains are represented with continuous practices required for planning, developing, and operating software. In the conducted research work, the focus was on the linkage between development and operations, thus, the corresponding continuous practices are further explained:

- **Continuous Integration (CI)** is an automatically triggered process of interconnected steps, including code compilation, execution of automated tests

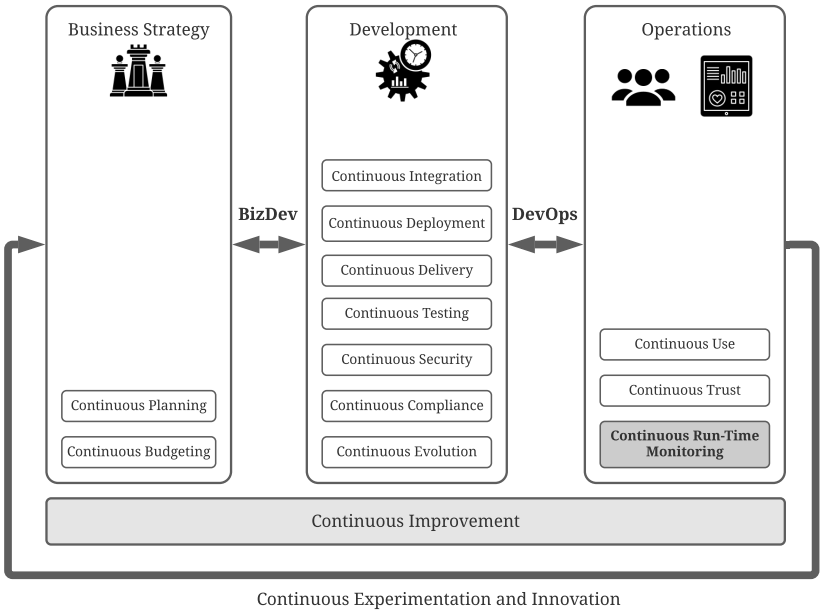


Figure 1: A holistic view on the continuous activities adapted from Fitzgerald and Stol [55]

on unit, integration, system and acceptance level, validating code coverage, and building deployment packages [208].

- **Continuous Testing (CT)** is an uninterrupted process of running automated tests on the latest version of the software. Its main purpose is to help reduce the time between the introduction of errors and their detection.
- **Continuous Deployment (CD)** is an automatic process of reliable delivering software to the customer environments as soon as new code is developed [208]. According to Rodríguez [177], the CD has roots in Agile Software Development (ASD) as agile methodologies are based on the same principles of delivering valuable software continuously while accelerating development processes.
- **Continuous Delivery** ensures that code is always in a deployable state after automated testing, but the decision to release is manual. Unlike continuous deployment, which automatically releases updates, continuous delivery allows teams to decide when to release updates [55].
- **Continuous Run-Time Monitoring (CM)** is a continuous activity of monitoring running software services, which may enable tracking their health status and timely detection of various malfunctions [55] while collected monitoring data may serve for improving development processes [30], generating or prioritizing test cases [170, 211], or detecting anomalies [96].
- **Continuous Improvement** is a data-driven activity for improving software quality and increasing customer satisfaction by utilizing data generated in the CI/CD pipeline to drive decision-making and eliminate waste.

Another term, mostly used among practitioners, that encompasses the aforementioned continuous activities is *continuous integration/continuous deployment (CI/CD) pipeline*. It is considered a set of automated steps designed to streamline and accelerate the delivery of software. The effectiveness of the CI/CD pipeline is further enhanced by DevOps principles [207], which promote a shared cultural mindset among development and operations teams. These principles are essential for maintaining a seamless and uninterrupted flow of responsiveness within the pipeline, enabling teams to rapidly address issues and implement changes without compromising quality or stability. By integrating CI/CD with DevOps, organizations can achieve a higher degree of efficiency, agility, and reliability in their software development and delivery processes [199].

Implementation of the continuous practices may bring various benefits to software development environments such as [177]: 1) *shorter time-to-market* due to the higher frequency of release cycles; 2) *improved release reliability* achieved with a narrower test focus since each deployment introduces only a limited amount of code while also relying on rollback mechanisms; 3) *instant feedback* from automated infrastructure that enables discovering, locating, and resolving issues more

rapidly, utilizing both pre-deployment extensive testing and post-deployment run-time monitoring; 4) *increased developers' productivity* as they can only be oriented on the development of new features while relying on the automation infrastructure and feedback from operations for discovering bugs and various anomalies.

In the next subsection, we briefly present various cloud deployment models and their significance in contemporary software architecture, setting the stage for understanding the systems analyzed within the case companies studied in this thesis. Further, we narrow our focus to continuous run-time monitoring since it has an important role in the feedback from operations to development and providing valuable alerting notifications to development in case of detected anomalies, which was recognized as crucial in addressing identified problem instances in case companies.

1.2 Cloud Deployment Models

In today's rapidly evolving technological landscape, the adoption of cloud computing has become increasingly common for modern software systems. Cloud computing offers various deployment models that provide flexibility, scalability, and cost-effectiveness for organizations. One widely used deployment model is the *public cloud*, where services and infrastructure are provided by third-party cloud providers, such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform [203]. With public clouds, services are available through a public cloud service provider that hosts the cloud infrastructure, and customers don't have any control over the located infrastructure. Another deployment model is the *private cloud*, which is dedicated to a specific organization and hosts applications that are relevant to its business operations. Private clouds offer organizations more control over their infrastructure and data, as it is not shared with other organizations. Community clouds are a shared deployment model between organizations with similar requirements and business objectives. Community clouds are maintained and managed by all participating members of the community [178].

In addition to these deployment models, there is also the concept of *hybrid clouds*, which combines elements of both public and private clouds. Hybrid clouds allow organizations to leverage the benefits of both public and private clouds, enabling them to utilize public cloud services for scalability and cost-efficiency while keeping sensitive or critical data on their private cloud for security and compliance purposes. These different cloud deployment models offer organizations the flexibility to choose the most suitable model based on their specific requirements, resources, and goals [178].

Organizations can utilize various deployment solutions to implement these cloud deployment models. Some widely used deployment solutions include [191]:

- **Infrastructure as a Service:** This solution provides virtualized computing resources, such as virtual machines, storage, and networks, enabling

organizations to deploy and manage their applications and data in the cloud infrastructure.

- **Platform as a Service:** This solution provides a platform for organizations to build, deploy, and manage their applications without the need to worry about underlying infrastructure details.
- **Software as a Service:** This solution allows organizations to access and use software applications over the internet without the need for installation or maintenance.
- **Serverless Computing:** This deployment solution allows organizations to run their applications without managing the infrastructure. The cloud provider takes care of all the infrastructure and resource management, allowing organizations to focus on developing and running their applications.

Overall, the choice of cloud deployment model depends on factors such as security requirements, resource needs, cost considerations, and organizational objectives. Additionally, the selection of the deployment model is significantly influenced by the system architecture of the software. Different system architectures have varying requirements that may align with specific cloud deployment models [77]. Several software system architectures are widely adopted today, with microservices, cloud-native, and layered (N-Tier) architectures being particularly prominent. Microservices architecture is favored for its scalability and flexibility, aligning well with the dynamic nature of cloud environments [45]. Cloud-native architecture leverages microservices, containers, and continuous delivery to maximize cloud benefits, such as elasticity and resilience. Layered architecture is popular for its modularity and maintainability.

The selection of architecture significantly impacts cloud deployment models. For instance, microservices and cloud-native architectures are ideal for distributed cloud environments, facilitating efficient resource use and high availability [10]. In contrast, a monolithic architecture may be better suited for a private cloud deployment model, providing the organization with full control over the entire stack [119]. Choosing the right architecture ensures optimal performance, cost efficiency, and scalability in cloud deployment, aligning with application and organizational needs. Furthermore, the system architecture also impacts the choice of deployment solutions within the selected cloud deployment model. For example, a system built on microservices architecture might benefit from a serverless computing deployment solution to offload infrastructure management, whereas a monolithic architecture may require Infrastructure as a Service for more control over virtualized computing resources.

It is essential for organizations to carefully analyze their system architecture and its requirements to make informed decisions about the most suitable cloud deployment model and deployment solutions. This alignment ensures efficient utilization of cloud resources and maximizes the benefits offered by cloud.

1.3 Continuous Monitoring

The main idea of the term *continuity* is to ensure an uninterrupted software development life cycle, even after deployment. When a software product has been deployed, it is important to continue with post-deployment activities, such as *continuous monitoring*. Suonsyrja et al. [202] studied the importance of post-deployment activities and how automatically collected data from operations could be used as feedback for requirement modifications and further development processes. They reviewed the literature that relates to the evolution of software development methods and created a questionnaire to examine if companies are interested in collecting usage data and what they think the data could be used for. The results show that practitioners are interested in adopting post-deployment activities and that these activities are important for gathering data about performance, time, and the way of using the software product but also for discovering system failures. Additionally, continuous monitoring ensures that software systems remain reliable and functional after deployment, complementing traditional testing, especially when time-to-market pressures reduce the scope of pre-deployment testing.

Monitoring data, also referred to as operational data, is a primary means of achieving full observability. This predominantly abundant data can vary significantly in structure and format and take various forms, including logs [101], metrics [159], and traces [24]. The use of monitoring tools is crucial for gathering and interpreting the diversity of data, allowing for the prompt identification of performance issues and anomalies. These tools offer powerful capabilities for aggregating and visualizing data, providing essential insights that support proactive data management [65]. Monitoring data has a wide range of applications and is crucial for improving various aspects of the software development life cycle. For example, using feedback based on monitoring data aggregation, integration, and mapping for determining analytical and predictive relationships to development processes has been reported by Cito et al. [29, 30]. The testing phase may also benefit from monitoring data, such as improving the reliability of testing through iterative estimation of the operational profile, as presented by Pietrantuono et al. [170], or for the automatic generation of test cases, as reported by Deepika et al. [211]. Moreover, security and privacy issues can be addressed utilizing monitoring data [163]. Thus, monitoring data is essential for identifying and addressing various performance, security, and testing concerns in software development.

Beyond improving performance and security, monitoring data plays a key role in decision-making and operational efficiency. Some operations data has been used for decision-making and detecting harmful release candidates [20], while Fu et al. presented how to use clusters of system logs to infer dependencies that are used for failure prediction and root cause analysis [59]. Additionally, the richness of log data was leveraged to detect abnormal behavior and performance issues, as emphasized by Candido et al. [36]. Their Contemporary Logging Framework categorizes essential aspects of logging and monitoring, emphasizing structured

approaches to manage the vast and heterogeneous data generated by modern software systems. Advanced monitoring solutions, such as IntelligentMonitor, integrate real-time data collection, intelligent analytics, and machine learning to enhance monitoring capabilities. According to Thantharate [209], these technologies effectively manage data overload, reduce alert fatigue, and improve system visibility, thereby enhancing overall system performance and reliability. Hence, continuous monitoring aids in identifying and resolving issues that arise post-deployment but also provides critical insights that drive ongoing software improvement. The integration of advanced analytics and machine learning in monitoring practices, as demonstrated by recent research, highlights the potential for continuous monitoring to significantly improve the resilience and efficiency of software development and operations.

One of the initial aims when collaborating with the DevOps and CloudOps teams from the case companies was to understand the current monitoring practices and specific needs of each company. By engaging closely with these teams, we wanted to gain a comprehensive overview of the existing monitoring frameworks, tools, and data collection methods. In this way, we could further explore solutions for advanced analysis of operational data in order to address identified problem instances and enhance the monitoring capabilities of the case companies.

Initial insights gained from this collaboration highlighted several critical areas for improvement. One of the key findings was the need to prioritize *anomaly detection* as a primary method for utilizing monitoring data. This focus emerged from our observations and discussions during the early stages of the project. It became evident that while the companies had established monitoring systems, there were significant shortcomings in the way *alerting events* were reported and managed. This hindered the effectiveness of the monitoring systems and often led to unnecessary disruptions and challenges in identifying relevant issues. As a result, there was a recognized need for a new and improved approach to discovering unusual and unexpected deviations in software behavior, specifically through the detection of *anomalies*.

1.4 Anomaly Detection

Anomaly detection is a research area with growing attention across numerous domains such as cybersecurity, healthcare, bioinformatics, industrial fault detection, genetics, and many more [181]. An observation that significantly and unexpectedly deviates from the normal pattern in the observation data is considered an *anomaly* [14]. However, the concept of anomaly detection may be context-dependent, and different approaches may be needed for different types of data. According to Hagemann et al. [75], the three most common approaches to anomaly detection entail classical machine learning, deep learning, and statistical methods. The selection of the approaches substantially depends on the available data, its type, and volume. Moreover, the data available for the learning of normal and

anomalous dependencies may define the selection between unsupervised and supervised methods. Labeled data for applications of anomaly detection are rarely available, and even when there are at least partial annotations, they might be insufficient and unreliable since anomalies in most of the industrial context do not appear often and periodically [181]. Thus, they are hardly detected and require special treatment. Moreover, a careful selection of suitable approaches for their analysis is needed.

In this context, semi-supervised and unsupervised learning methods are particularly valuable as they minimize the need for labeled data. For example, in microservice environments, integrating natural language processing (NLP) with distributed tracing data has proven effective [117]. This approach analyzes spans within traces, detecting anomalies based on deviations without requiring extensive labeled datasets. This method demonstrates the flexibility and adaptability of semi-supervised learning, leveraging the structure of the data to identify performance issues and differences introduced between software releases. Additionally, the use of large language models (LLMs) to automate the configuration and interpretation of anomaly detection systems exemplifies the power of unsupervised methods [241]. These models can analyze historical data, extract patterns, and identify anomalies without explicit labels, simplifying the monitoring process. This approach enhances the system's ability to adapt to new and evolving anomalies, facilitating a more efficient and robust detection process.

These examples illustrate how semi-supervised and unsupervised methods effectively address the challenge of limited and unreliable labeled data in anomaly detection. Our attention was also oriented towards such approaches due to the lack of annotated data. However, we initially focused on classical machine learning approaches for labeling the data and predicting anomalies, but due to data complexity and untrustworthiness of the labeling process, we shifted focus towards advanced unsupervised approaches for anomaly detection such as *deep learning*. More specific selection of anomaly detection approaches has been defined by the software system and environment in case companies. Thus, we focused on approaches for treating *multidimensional time series data* as the monitoring data, indicating the health of the software, consists of multiple numerical observations captured within a specific time range across multiple services or system components. This type of data, known as multivariate time series, is quite challenging for analysis, especially if there is no ground truth data. Consequently, we relied on the latest advancements and widely recognized techniques to ensure effective monitoring and anomaly detection in complex systems within case companies.

1.5 Alerts

Alerts are warning or error notifications reported based on detected software malfunctions or performance degradation. They are a convenient way of notifying development teams about the state and the health of the software running in oper-

ations. Alert rules can be configured through a heuristic method involving manual threshold setting for monitored data or by using advanced algorithms like anomaly detection to process the raw data. However, there might be some variations depending on the industry domain, such as a large-scale service system of a commercial bank [253], cyber-physical systems [134], process plants e.g., an offshore oil gas separation plant [144] or large-scale enterprise IT system [131], on how alerting events are triggered.

Alerts play a crucial role in the timely diagnosis of critical failures. However, without proper management, they can overwhelm development teams with excessive information, leading to confusion and inefficiency. Ensuring that alerts are well-calibrated and relevant helps maintain clarity and focus. For instance, Zhao et al. [253] report a case where the development team had wasted time debugging non-severe alerts while severe ones were missed. Thus, they propose an effective ML-based algorithm that utilizes both reported alerts and key performance indicators (metrics) to trigger high-priority alerts more accurately. Similarly, to support manual alert examination, Lin et al. [131] propose an unsupervised ML approach for clustering alerts based on their content.

To tackle the challenges posed by excessive and often irrelevant alerts, refining alert mechanisms in cloud monitoring systems is imperative for maintaining focus and operational efficiency. One approach, as discussed in research on mitigating alert fatigue [219], involves leveraging ML techniques to filter and prioritize alerts, ensuring that only critical notifications reach administrators, thereby alleviating the cognitive burden of handling irrelevant alerts. Furthermore, studies on alert filtering in cloud systems propose a binary classification method to distinguish between significant and non-essential alerts, utilizing data-driven models to enhance the relevance of the alerts presented [218]. Additionally, an investigation into the anti-patterns of alerts in industrial cloud environments reveals common issues like unclear descriptions and misleading severity levels, which can complicate diagnosis and delay resolutions [238]. To address these issues, the adoption of preventative guidelines and the development of personalized alert strategies are suggested, aiming to tailor alert systems to specific system conditions and user roles.

These aforementioned combined efforts strive to create a more efficient and responsive cloud monitoring environment, where practitioners can promptly and accurately address critical issues without being sidetracked by non-essential information. Similarly, we aimed to enhance monitoring and alert strategy in case companies in order to optimize the responsiveness of the DevOps teams and the reliability of their systems. This new strategy emphasizes a proactive approach, continuously monitoring the system's health and identifying anomalies in real-time monitoring data. These anomalies promptly trigger *alert notifications*, providing actionable suggestions for the practitioners to ensure timely awareness and response.

1.6 Operational Failures

A *failure* denotes the inability of a system or one or more of its components (services) to perform one of the functionalities [195]. This is an unwanted epilogue of any system perturbation we want to avoid. Early detectability of observations that significantly deviate from the normal pattern, namely *anomalies*, might be crucial for preventing the roll-out of severe system *failures*. Further, an anomaly detection service that provides early warning of impending failure can reduce the costs associated with reliability and downtime. Reflecting the need for early failure detection, Cotroneo et al. [34] present a practical solution by implementing non-intrusive event analysis and monitoring rules, which significantly improve the system's ability to detect and manage anomalies in a timely manner.

By addressing the complexities of monitoring and alerting in DevOps, this thesis aims to develop solutions that mitigate the risk of widespread system failures. The goal is to ensure that potential issues are identified early through anomaly detection, preventing them from escalating into significant failures that could disrupt multiple parts of the system.

2 Research Approach

This thesis aims to enhance DevOps practices and address the challenges inherent in fast-paced development environments by applying ML-based solutions. Central to our research approach is the collaboration with two case companies, providing a unique opportunity to ground our research in real-world industrial settings. This partnership enriches our research with practical insights and ensures the relevance and applicability of our findings to the software industry domain. Our research is organized into sequential goals, each designed to build on the insights and developments of the previous through multiple iterations. The research goals are defined as follows:

- **RG1:** Understand challenges in the feedback loop from operations to development concerning overflow of monitoring data and alert management.
- **RG2:** Define problem constructs and envision matching solution designs according to the state-of-the-art literature.
- **RG3:** Implement and evaluate proposed solution designs in multiple industrial contexts.
- **RG4:** Examine wider applicability of implemented solution instances and overall acceptance of ML-based solutions for enhancing DevOps practices.

Accordingly, the overarching goal of this thesis is to understand and improve certain aspects of the socio-technical industry practice, which aligns with the main

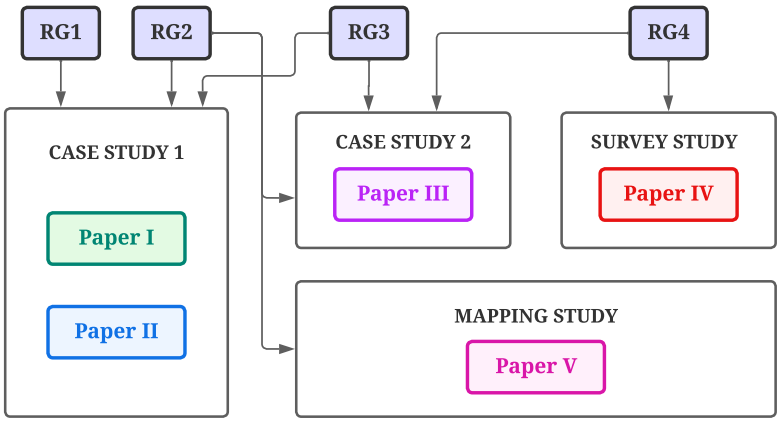


Figure 2: An overview of conducted research mapped to the research goals and outcomes

principles of the *design science* paradigm [82]. To achieve these goals, different research approaches were employed under the design science frame to understand problem instances and develop and evaluate design artifacts. The essence of the design science research contributions is encapsulated in a *technological rule* [49], typically expressed in the following format: *To achieve* ***<Effect>*** *in* ***<Context>***, *apply* ***<Intervention>***. This formulation is suitable for matching problem instances and corresponding solutions, offering prescriptive recommendations for practice [4] that can be applied to similar problems in different contexts. Thus, it transforms the knowledge gained from the research into a pragmatic, actionable format that can guide practice and contribute to the broader knowledge base in software engineering, in our case, the DevOps context. More detailed reflection on the contributions of this thesis expressed through technological rules, including their formulation, is given in Section 3.

The overview of conducted research in this thesis is shown in Figure 2, visualizing the relationship between the studies, papers, and research goals. The **RG1** was mainly investigated in the first case company, studying a microservice system developed and operated in the DevOps environment and gathering insights from experienced practitioners. In the subsequent studies, identified problem instances were further examined by reviewing the relevant literature and state-of-the-art ML-based solutions applicable to identified problem instances. Thus, the second research goal, **RG2**, was investigated in both case studies in order to discover matching solutions for addressing challenges concerning monitoring data overflow and alert management in the context of cloud-based software systems. The scarcity of publications that map corresponding problem contexts (e.g., multivariate metric data) with matching solutions (e.g., ML techniques for anomaly detection) moti-

vated conducting of the mapping study. Thus, this study also played a crucial role in achieving **RG2** by systematically aligning specific challenges in managing high volumes of different monitoring data types with effective ML solutions.

The second case study extended the validity of the technological rule defined in the first case study by instantiating an improved version of the solution instance in a different but similar industrial context. Therefore, the **RG3** was addressed in both case studies, while implementation and evaluation in a new cloud-based environment enabled defining more generalizable contributions (see Section 3.6). The outcomes of the second case study complemented with the survey study results, led to accomplishing the research goal **RG4**. This included further examination of the applicability of the proposed and implemented ML-based solution in diverse industrial contexts, including different types of cloud-based systems and tools used for their development and operations. The collective findings from Papers III and IV contributed to defining a more general technological rule grounded in empirical evidence and rigorous evaluation (Thesis TR, Section 3.6).

As shown in Figure 3, the *design science* (DS) frame entails four main activities: problem conceptualization, solution design, instantiation, and empirical validation [182]. *Problem conceptualization* is a fundamental constituent of design science research, but it is not necessarily the first step. However, it may be a starting point of problem-driven research as the problem first needs to be understood in order to envision potential solutions. *Solution design* is an activity of mapping identified problem to a matching solution [49]. *Instantiation* refers to the implementation of the solution design in a specific industrial context while *empirical validation* aims at evaluating the solution instance and examining how well it addresses identified problem instances [182].

The previously introduced research goals are achieved by conducting the aforementioned DS activities. The first case study resulted in three papers (Paper I, II, and VI), where Paper VI is not included in the thesis as its main findings were presented in Paper II. Each of the research approaches in **Paper I** and **Paper II** include activities shown in arrows (see Figure 3) that span from problem to solution and from practice to theory domain. Interviews and observations were used in Paper I to explore the problem domain in the first case company to identify problem instances and formulate a general problem construct considering related work in the field, which adds to the contribution **C1**. Further, the matching solution was designed considering previous research and available solutions to similar problems. It served as a proof of concept for further work since the complexity of the industrial context hindered a full-scale implementation. As presented in Paper I, we managed to execute a partial empirical validation of the initial solution instance, considering time and environment constraints. Thus, the cycle, shown in Figure 3, is not fully complete and only partially adds to **C3**.

The outcomes of the first study (Paper I) were complemented by conducting another cycle of DS activities within the same case, whose results were published in Paper II. As shown in Figure 3, the problem conceptualization step in the second

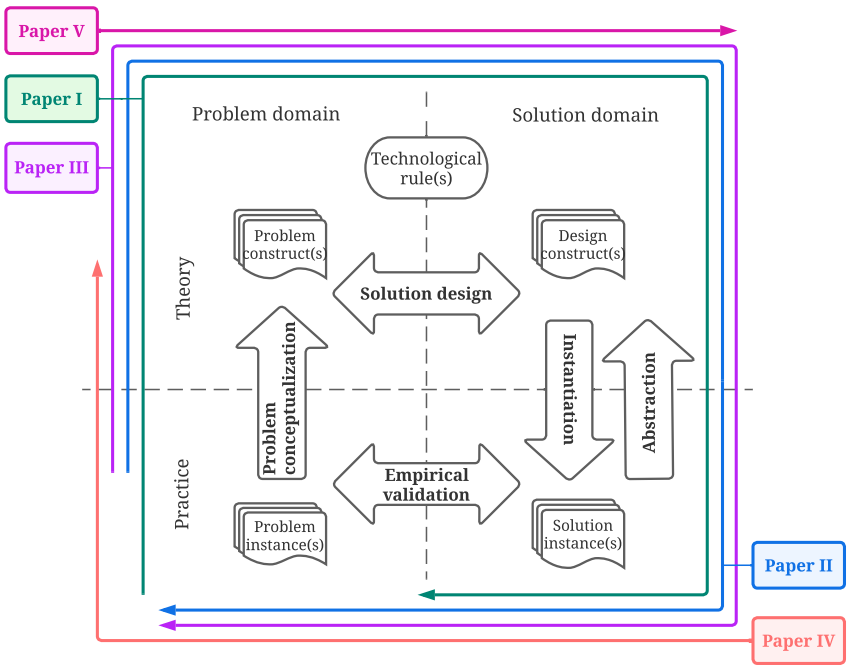


Figure 3: Research approach under the design science frame

study (**Paper II**) was less extensive, including brief discussions with practitioners and investigation of relevant solutions to formulate a more condensed problem description. Moreover, in Paper II, we advanced the proposed solution design, presented in Paper I, with respect to the state-of-the-art deep learning solutions in order to capture all notions of anomalousness in operations data (**C1**). The goal was to provide a more robust and reliable solution than the initial one, which complements contributions from Paper I and adds to **C2**. These contributions related to the problem conceptualization and solution design activities in the DS cycles were presented in both Paper II and Paper VI. However, in Paper VI, the focus was mainly on the exploration of approaches for advancing the solution design presented in Paper I. Additionally, actual implementation and empirical validation of the advanced solution instance in a cloud environment was presented in Paper II, adding to contribution **C3**.

Similarly, we employed the design research in **Paper III** to enhance alert management practices within a second case company, aiming to extend the validity of prescriptive recommendations for anomaly detection in multivariate data. The problem conceptualization phase was shorter according to Figure 3 as we only needed to confirm hypotheses from previous studies regarding the problem con-

text. However, it still required close collaboration with DevOps and CloudOps teams, including interviews and observations of monitoring and alerting practices to precisely define the scope of identified operational challenges. The next phase involved the creation of an automated workflow to enhance monitoring and alert reporting. The solution leveraged cutting-edge ML technologies, including deep transformer networks for anomaly detection and GPT-3 models for log interpretation, aimed at producing *interpretable alerts* from complex multivariate data streams. The instantiation of the proposed solution was realized through the development and deployment of autonomous monitors using Microsoft Azure services, ensuring compatibility and integration with the company's existing cloud infrastructure. This phase was crucial for transitioning from theoretical designs to practical applications within the operational environment of the case company. Finally, empirical validation was conducted to assess the effectiveness of the implemented solution, which contributes to both **C2** and **C3**. Feedback was collected from the DevOps and CloudOps teams through surveys and direct observations of alert notifications in the MS Teams channel.

An industrial survey study presented in **Paper IV** focused entirely on empirical validation and problem conceptualization, which perfectly fits within the DS frame as shown in Figure 3. The survey study was designed and conducted following the recommendations and guidelines presented by Molléri et al. [155], Kasunic [112], and Linåker et al. [136]. The research approach encompassed a comprehensive sequence of steps, beginning with clear documentation of the research objectives and questions, focusing on identifying the frequency, mechanisms, and types of operational failures encountered by software development companies. Sampling a diverse group from the industry allowed capturing a broad spectrum of experiences and data practices. As the primary tool, the questionnaire is thoroughly designed and validated to ensure accuracy and relevance. Following data collection, a detailed analysis enabled uncovering patterns and insights regarding the role of ML-based anomaly detection and smart alerts, elaborated in Paper III. This step was crucial in evaluating the practicality and effectiveness of such technologies across different organizational contexts. Finally, the findings were synthesized and reported, offering a comprehensive overview of current practices and the potential impact of AI/ML tools on software development and operations, adding to **C3**.

The final systematic mapping study reported in **Paper V** significantly contributes to the theoretical aspect of the problem-solution domain as visualized in the design science framework shown in Figure 3. The study employs a systematic approach based on the guidelines of Kitchenham and Charters [116], aiming to provide a comprehensive overview of the use of monitoring data for anomaly detection in cloud-based software systems. The methodology involves a rigorous search and selection process, using a search string optimized through a Quasi-Gold Standard approach, primarily utilizing the Scopus database for broad literature coverage. The study includes 104 papers, which are methodically reviewed to categorize monitoring data types and the tools employed for data collection.

The analysis extends to preprocessing techniques and anomaly detection methods, mapping them to different data categories and assessing their practical applicability in real-world scenarios, which overall contributes to **C1**.

3 Summary of Results

The overall contribution of this thesis is related to the goal of improving feedback from operations to development in cloud-based and large-scale software systems developed and operated in DevOps environments. Through detailed case studies, this research provides a comprehensive understanding of the operational challenges faced in large-scale software systems, specifically addressing alert management strategies and monitoring data usage. A novel proactive alert strategy, named *autonomous monitors*, has been developed, inspired by both detailed observations of cloud-based software systems and the latest academic findings in the field of anomaly detection. This strategy aims to detect operational *failures* early, ensuring continuous monitoring and enhancing the ability to predict and prevent anomalies before they disrupt operations. The effectiveness and applicability of these autonomous monitors have been validated through empirical evaluations in two case companies, complemented by a survey assessing their use across various industrial settings. A more detailed overview of specific results and contributions per paper is given below.

3.1 Closing the Feedback Loop in DevOps

The research in this thesis was initiated in collaboration with case companies, which directed the project towards addressing practical challenges and industry-specific needs. The goal of the first study, presented in Paper I, was to explore the problem context in the first case company and design a solution for a conceptualized problem, which is later instantiated and evaluated as a proof of concept for further work. Among the three problem constructs of the general alert flooding problem, we focused on *optimization problem*, which we identified as the most critical and impactful. This problem concerns the difficulty in distinguishing between high-priority alerts indicating significant system failures, and less critical alerts that point to temporary, non-disruptive glitches. The lack of clear differentiation often led to an overwhelming volume of alerts, causing unnecessary load and inefficient resource allocation.

To tackle the *optimization problem*, we proposed a conceptual design introducing an additional element, a *smart filter*, within the feedback loop from operations to development as shown in Figure 4. As part of our second contribution (**C2**), this solution design acts as *autonomous monitors* that detect performance anomalies in near-real-time. The smart filter analyzes metrics from multiple systems components, triggering alerts based on sophisticated rules created from multiple input

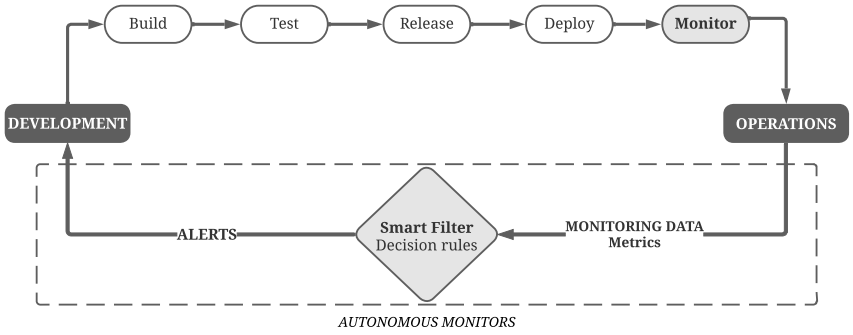


Figure 4: Overview of the solution in Paper I

variables. This approach allows for the identification of unusual system behaviors, facilitating rapid and accurate responses to the most relevant system issues.

Building on this design, we developed and tested a prototype, contributing to **C3**. The prototype utilized basic machine learning techniques, specifically tree-based methods [56, 61], to generate advanced decision rules. We used a custom labeling process based on exceeding thresholds for individual features and considering factors like service vulnerabilities and alert frequency. The evaluation of our prototype against the existing alert system in the case company, as well as an unsupervised multivariate anomaly detection (MAD) method [130], revealed significant improvements. While the MAD model maintained a similar level of noisy alerts as the existing system, our smart filter greatly reduced false positives. This enhancement simplified the root cause analysis and refined the alerting process, ensuring that only relevant and actionable alerts were communicated, thereby improving the efficiency of the system's response to critical failures.

Throughout this study, we identified several challenges that impacted our further decision-making about the future work in the case company. A key challenge was the ongoing uncertainty caused by the lack of a clear resolution path for fired alerts, which were communicated through both Slack and email without a clear resolution strategy. Additionally, it became evident that determining the criteria that trigger developers to respond to alerts was complex, making it difficult to define *a "real anomaly"* and evaluate new detection approaches accurately.

The previously presented solution design included a labeling process, which is, in general, a very thorny process and highly sensitive in case of anomaly detection problems. Hence, we decided to shift our focus towards *unsupervised deep learning approaches* for anomaly detection in multivariate data, governed by the existing solutions in the state-of-the-art literature. The final aim was to create a unique technical solution that would overcome all aforementioned challenges and enable timely reporting of the most relevant alerts to only one of the communication platforms.

3.2 Towards Optimization of Anomaly Detection

In the following study conducted within the same case company, we addressed shortcomings of the solution design shown in Figure 4 and explored state-of-the-art deep learning (DL) approaches for its improvement (**C1**). We were inspired by the results achieved using the unsupervised DL methods in addressing anomaly detection tasks [96, 138, 148]. Unsupervised learning does not require data labeling, which is preferable due to the ambiguity of fired alerts in the case company, while deep learning enables capturing highly complex and nonlinear dynamics in the multivariate data. Furthermore, we decided to explore approaches for the evaluation of the ML-based solutions without any ground truth data.

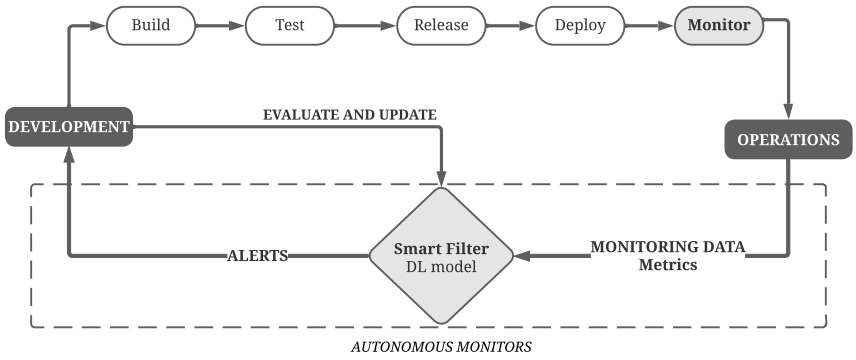


Figure 5: Overview of the solution in Paper II

Within the next design cycle, we implemented the smart filter using the reconstruction-based deep learning model (**C2**), specifically *LSTM autoencoder* (see Figure 5). This model proved effective in identifying anomalous behaviors in multivariate time series data without requiring prior data labeling. The key strength of using autoencoders lies in their ability to learn the distribution of normal observations and subsequently identify deviations from this norm. By reconstructing expected system behaviors, the autoencoder model highlighted instances where the actual system behavior deviated significantly from the reconstructed output, signaling potential anomalies. This technique was robust against parameter changes and demonstrated consistent performance in detecting both high and low-impact anomalies.

A significant aspect of our research was the evaluation of this technique (**C3**) without labeled data, a common challenge in real-world applications. To address this, we relied on feedback from development teams to assess the model's performance. During the implementation phase, the smart filter sent alerts to the development team, who then provided qualitative assessments of the alerts' relevance and impact. This feedback loop, shown in Figure 5, was critical in fine-tuning the

detection thresholds, refining the model, and improving its accuracy. We refer to this process as *optimization* because it aims to achieve the best-performing model through multiple iterations. Thus, the feedback supported by expert insights, can serve as a viable substitute for traditional ground-truth data in model evaluation.

Our findings indicated that the model was particularly effective in *reducing false positives*, which are a major source of noise in alert systems. Reduction of unnecessary alerts allowed development teams to concentrate on resolving relevant anomalies, enhancing the overall efficiency of the failure response process. The feedback mechanism also revealed that the smart filter occasionally identified low-impact anomalies that, while not immediately critical, provided early warnings of potential issues, offering a proactive approach to system maintenance.

Furthermore, the research recognized the importance of a flexible and scalable infrastructure for deploying machine learning models in a DevOps context. The cloud-based solution we developed facilitated the seamless integration of the smart filter into existing workflows, enabling real-time data processing and model updates. This infrastructure is crucial for maintaining the relevance and accuracy of the ML models as the operational environment evolves.

Contributions of this study expressed as the technological rule:

TR 1.1 To improve alert management in DevOps, integrate a DL-based solution for anomaly detection in operations and iteratively refine it using feedback-generated labeled data.

3.3 Detecting Early Signs of Operational Failures

By continuing further research in this thesis, the goal was to examine how identified problem-solution instances were common in other similar cloud-based industrial contexts. Therefore, we initiated collaboration with the *second case company* to confirm the hypothesis that identified challenges in monitoring and managing operational data are widespread. Additionally, this study aims to develop a more sophisticated solution that combines deep learning models for anomaly detection with a powerful language model to interpret logs and generate detailed, *actionable alerts*. This approach detects anomalies more accurately but also helps operations teams understand the root causes and implications of these anomalies, enabling faster and more effective resolutions. Expressed as the technological rule:

TR 2.1 To report highly relevant and clear alerts in DevOps, integrate a combination of the DL-based solution for anomaly detection with LLMs for generating interpretable notifications.

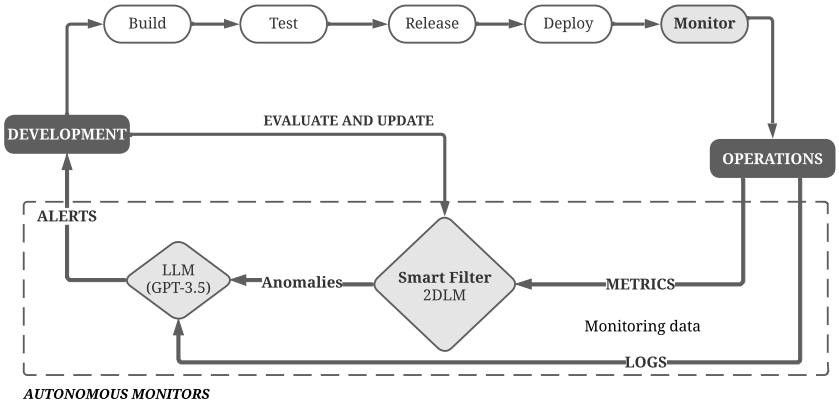


Figure 6: Overview of the solution in Paper III

As shown in Figure 6, the *smart filter* consists of two DL models, LSTM Autoencoder (*LSTMAE*) and Transformer-based Anomaly Detection (*TRANAE*). Thus, in addition to the *LSTMAE* model used in the first case study, there is an additional model with advanced architecture and the ability to capture complex temporal dependencies (**C2**). The *TRANAE* model demonstrated superior performance in detecting both short-term and long-term anomalies. However, it had not been previously evaluated in industrial contexts. The combination of these models ensured a comprehensive monitoring solution that could identify a wide range of anomaly types.

A significant innovation in this study was the integration of the *GPT-3.5 Turbo* language model to analyze logs and provide detailed, *interpretable alerts*. This integration transformed raw anomaly detection outputs into actionable instructions. The generated alerts included information about detected anomalies as well as contextual information about the potential root causes and suggested resolution steps. This feature greatly assisted operations teams in understanding the nature of the detected anomalies, reducing the cognitive load required to interpret the alerts, and facilitating quicker and more effective responses. The ability to deliver clear, actionable insights significantly improved the operational workflow, making it easier for teams to prioritize and address early signs of *operational failures*. Outlined as the technological rule:

> **TR 2.2** To increase developers' responsiveness to fired alerts, implement intuitive alert interfaces, allowing for immediate interpretation and action.

The evaluation of the *autonomous monitors* involved both quantitative and qualitative assessments (**C3**). The quantitative assessment of the DL models demon-

strated their effectiveness in detecting anomalies within multivariate time series data. The *TRANAE* model's higher precision, with a notable 11.5% improvement over *LSTMAE*, highlighted its robustness and effectiveness in dynamic thresholding and anomaly detection. Overall, the evaluation confirmed the effectiveness of both models, with *TRANAE* showing a considerable advantage in precision and reliability for real-time monitoring in cloud-based environments.

In parallel, a qualitative evaluation was conducted through feedback collected from the operations teams who actively assessed reported alerts. The teams provided insights into the usability of the generated alerts, focusing on their relevance, clarity, and actionability. While the feedback was predominantly positive, noting the system's effectiveness in early detection and response, it also highlighted areas for improvement, such as further refining threshold settings to enhance the accuracy of critical alerts. Overall, the qualitative feedback confirmed the system's value in operational contexts and provided a roadmap for future enhancements.

This study demonstrated that the developed solution could be adapted to different industrial contexts, provided the monitoring data types and operational requirements are similar. This innovative solution represents a significant step forward in *proactive monitoring* and management of complex cloud environments. Future research will focus on further refining proposed and implemented techniques and improving the interpretability of detected anomalies, potentially incorporating more advanced LLM models.

3.4 ML-Driven Alert Management Strategies in Industry

To be able to examine the wider applicability of ML-based solutions for alert management beyond the scope of the two case studies, we initiated an industrial quantitative survey study (**C3**). We collected the data using a structured questionnaire as shown in Figure 7. The survey gathered insights from *25 respondents across 11 software companies*, aiming to understand current monitoring practices, challenges, and the future trajectory of ML adoption in this domain. Moreover, the goals of this study were to assess the current challenges in handling *operational failures*, understand the utilization of monitoring data and tools, and evaluate the readiness of different roles to adopt and rely on AI/ML technologies. This approach allowed us to gather empirical data that reflects the varied experiences and insights of practitioners working in different industrial environments.

The study identified a range of operational failures dominant across various companies, highlighting a critical area of concern in the software development domain. These failures include *system outages, performance degradation, data integrity issues, and security vulnerabilities*. These failures often result from complex interactions within different system components due to integration or compatibility issues. The impact of these operational failures extends beyond immediate technical setbacks, affecting development time, costs, product quality, and cus-

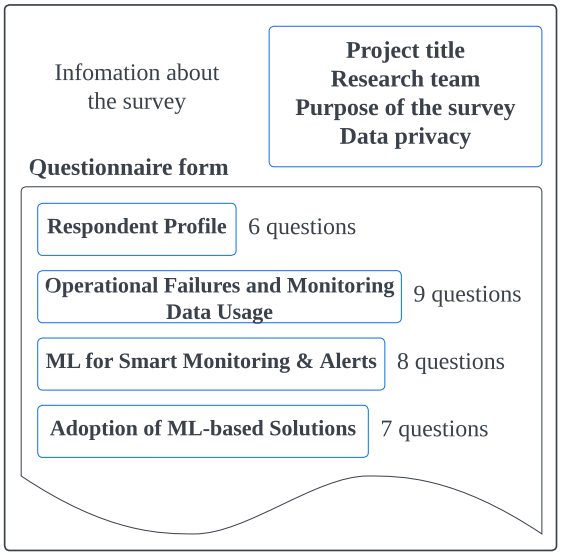


Figure 7: Structure of the questionnaire

tomer satisfaction. This diversity and frequency of failures emphasize the critical importance of developing advanced monitoring solutions.

The survey results revealed that monitoring data is extensively utilized across various professional roles to ensure system health and diagnose operational issues. Key data types, *such as performance metrics and log files*, are important in tracking system performance, identifying vulnerabilities, and resolving issues. DevOps engineers and quality assurance (QA) teams, in particular, rely heavily on this data to manage continuous integration processes and maintain high software quality standards. In contrast, managers rely on monitoring data for decision-making, while CloudOps engineers and software developers (DEV) use it to track performance trends and identify vulnerabilities. Among the monitoring tools, *Amazon CloudWatch, Azure Monitor, and Grafana* are widely accepted due to their robust features and seamless integration. The survey also indicated a consistent distribution of different types of monitoring data across various tools, suggesting that *performance metrics and log files* are always prioritized regardless of the tool used.

Despite the availability of advanced monitoring tools, many companies still employ *basic alert strategies* (threshold-based). Moreover, the survey findings highlighted varied perspectives among roles regarding the need for advanced ML-driven monitoring solutions. Some of them are eager to adopt such solutions (DEV and QA) while others prefer to stick with existing basic monitoring strategies (DevOps and CloudOps), possibly due to concerns about implementation complexities. However, there is a recognized need for more advanced analysis of

monitoring data where the main benefits anticipated from adopting ML include *faster detection of anomalies*, *reduced manual monitoring efforts*, and *more accurate root cause analysis*. In contrast, challenges such as the integration of ML solutions with existing systems, the cost of implementation, and concerns over data privacy and security were emphasized as significant barriers.

Finally, the readiness to adopt AI/ML solutions as well varies across different roles and companies. The study found that while there is general optimism about the future impact of these technologies, current adoption levels are relatively modest. For instance, roles such as software developers and quality assurance practitioners showed a higher degree of openness towards these technologies, recognizing their potential to streamline processes and *improve accuracy in failure detection*. On the other hand, roles more closely tied to infrastructure management, like DevOps and CloudOps engineers, exhibited a more cautious stance, possibly due to concerns about the operational complexities involved in integrating these technologies. The survey revealed a gap between the minimal current impact of AI/ML tools on development and testing and the anticipated future benefits, such as faster development cycles and better operational efficiency. This indicates a *transitional phase* where companies are beginning to explore AI/ML capabilities but have yet to fully integrate them into their workflows.

3.5 A Mapping Study of Monitoring Data for Anomaly Detection

While working on case studies and exploring anomaly detection techniques for the types of data identified in case companies, we encountered an unexpected challenge in identifying techniques that had been evaluated in real-world operational contexts. Moreover, many studies did not provide sufficiently detailed information on the data used for training and evaluating the implemented techniques. This lack of transparency and practical applicability motivated the initiation of this mapping study (**C1**), which aims to bridge the gap in existing research. By focusing on the use of actual operational data and its characteristics, this study seeks to provide practitioners and researchers with comprehensive insights into the practical application of anomaly detection techniques, ensuring their relevance and effectiveness in real-world cloud environments.

The analysis of 104 papers included in a review revealed that monitoring data is crucial for maintaining system observability and reliability, with significant emphasis placed on the *structure*, *types*, and *origins* of this data. The study identified the importance of both structured and unstructured data, with a predominant focus on structured data due to its ease of analysis. Monitoring data encompasses three primary categories: 1) *metrics*, providing quantitative performance measurements; 2) *logs*, capturing a record of events and transactions; 3) *traces*, illustrating the path of requests through the system. In terms of data origin, the study showed a strong preference for primary data from operational environments, utilized by

85% of the reviewed studies. This focus ensures that the findings are grounded in practical, real-world contexts. Secondary data, found in 27% of the papers, contributes to the research by providing additional validation through publicly accessible datasets and standardized benchmarks.

In examining the monitoring tools employed in cloud-based systems, we discovered several key tools, such as Prometheus, Sysstat/Perf, and Amazon CloudWatch, each serving vital roles in ensuring system observability. These tools are widely recognized for their ability to provide real-time data monitoring, system diagnostics, and alerting, which are essential for proactive system maintenance and timely failure resolution. In addition, according to the findings, there is a growing trend towards the adoption of *custom monitoring solutions*, which are developed to meet the specific needs of different cloud environment setups. These custom tools offer more flexibility for implementing specialized and adaptable monitoring infrastructures that can handle unique system requirements and complexities.

The findings include various data preprocessing and anomaly detection techniques that are thoroughly cataloged, specifying their application in cloud-based systems. Preprocessing techniques like *normalization*, *data formatting*, *feature selection*, and *dimensionality reduction* were identified as crucial for preparing data for analysis, ensuring consistency, and improving the performance of machine learning models. Additionally, the study acknowledges the variability in data preprocessing needs, specifying that extensive preprocessing is not required in some cases, particularly when data is already in a suitable format for analysis.

Anomaly detection techniques are systematically mapped and categorized into statistical, machine learning, and deep learning approaches. The mapping of these techniques revealed their applicability across different types of data, providing a comprehensive overview for practitioners and researchers to select appropriate techniques tailored to specific data characteristics and operational requirements. This study recognized the importance of aligning preprocessing and anomaly detection techniques with the nature of the data, ensuring effective anomaly detection and enhancing the reliability and performance of cloud-based systems.

The final results of the mapping study focused on the representativeness of monitoring data for real-world cloud environments. The findings showed the importance of using data that accurately reflects real-world conditions for evaluating anomaly detection techniques. According to the results, 71% of the papers utilized data sources that were more representative of actual operational contexts, coming from deployed production systems and benchmarks. However, only 27% of the papers reported usage of labels that were considered more representative, often lacking expert validation and including artificially generated data or labels resulting from common issues like infrastructure problems. This discrepancy shows that the data itself may be realistic, but the labels used to evaluate anomalies may not fully capture the complexities of real-world scenarios.

The results of this study also highlighted various anomaly detection techniques and their application to either more or less representative data. Techniques like

custom statistical methods and sequential models (using LSTM, GRU, and RNN) were more frequently evaluated using representative data, emphasizing their robustness in practical settings. In contrast, techniques like decision trees and random forests were often tested in controlled environments, potentially limiting their generalizability to real-world conditions. Thus, validating AD techniques with data and conditions that closely mirror actual operational environments is crucial for confirming their robustness and applicability in real-world scenarios.

3.6 Synthesis

The research synthesized from the five papers indicates that the challenges related to monitoring and managing operational failures are common across various cloud-based industrial contexts. The problem instance of alert flooding, coupled with the challenge of prioritizing critical alerts over non-critical ones, is the most significant concern that hinders operational efficiency. This was consistently observed in both case studies and was confirmed by the survey findings. This synthesis explores the generalizability of the proposed solution, *autonomous monitors*, and outlines the necessary adaptations and efforts required for their implementation in diverse operational contexts.

Autonomous monitors are designed to automatically detect and alert on anomalies within complex operational environments, primarily in cloud-based systems. The main functionalities of autonomous monitors include near real-time data monitoring, anomaly detection, and generating actionable alerts that provide clear insights and recommended actions. While the overall concept of autonomous monitors is broadly applicable across various industrial contexts, their successful implementation largely depends on an organization's readiness to adopt cutting-edge technologies and its ability to allocate the necessary resources.

In addition, the successful generalization of *autonomous monitors* requires consideration of several factors in collaboration with organizations willing to integrate *autonomous monitors* into their workflows.

Data. When implementing *autonomous monitors* across different industrial contexts, the adaptation of data handling processes is crucial. The types and sources of data can vary significantly between organizations, requiring customized approaches to data collection, preprocessing, and analysis. Additionally, it is essential to tailor data preprocessing steps, such as formatting, normalization, feature selection, and dimensionality reduction, to suit the specific nature of data (format, volume, frequency). Furthermore, organizations may need to address issues related to data quality and completeness, which are important for the accurate functioning of machine learning models.

Model Adaptability. The adaptability of ML models used for anomaly detection is key to wider industrial adoption. Required efforts include extensive model training and validation using representative data sets from the target environment. Additionally, the parameters and thresholds of anomaly detection models must

be fine-tuned to align with the operational patterns and behaviors. This involves adapting the models to recognize what constitutes normal versus anomalous activity within the context of that particular environment. Moreover, ongoing monitoring and iterative refinement are necessary to maintain model performance as the system evolves.

Integration with existing systems. Autonomous monitors must be capable of interfacing with current monitoring tools, databases, and alerting mechanisms, such as Amazon CloudWatch, Azure Monitor, or on-premise solutions, to collect and analyze data effectively. This often requires developing custom connectors or adapters that facilitate smooth data flow and communication between systems. In addition, organizations need a scalable and flexible technical infrastructure capable of supporting real-time data processing and ML model deployment. This includes investing in cloud-based solutions that can handle dynamic data loads and enable continuous model updates.

Feedback Loop. The feedback loop for iteratively refining ML models without traditional labeled data has proven to be an effective strategy. This mechanism leverages the expertise of operations and development teams to fine-tune the ML models used within *autonomous monitors*. Implementing such a feedback-driven optimization process in different organizations would require establishing clear communication channels and protocols for collecting and utilizing feedback. This collaborative approach also helps address the unique regulatory, privacy, and security concerns that may arise in different operational contexts. For example, feedback loops allow teams to monitor how personal data is being used and recommend changes to limit exposure, such as anonymizing or minimizing sensitive data used in training or predictions.

Therefore, the implementation of the *autonomous monitors* requires careful consideration of the unique aspects of each operational context even though the core functionalities are broadly applicable. By addressing these factors, *autonomous monitors* can provide significant improvements in operational efficiency and reliability across a wide range of cloud-based environments, making them a versatile addition to the modern DevOps toolkit.

The summary of presented insights and approaches is encapsulated in the thesis's technological rule:

> **Thesis TR** To enhance operational efficiency in DevOps, employ autonomous monitors to proactively monitor, detect, and alert initial signs of critical failures.

This rule synthesizes the overarching goal of the research — to create a resilient and efficient operational environment through proactive monitoring systems. By leveraging *autonomous monitors*, software development companies can streamline their alerting processes, reduce noise from non-critical alerts, and focus resources on addressing the most significant anomalies in monitoring data.

4 Discussion

In the domain of software engineering research, particularly when engaging with industrial partners, a critical challenge is balancing the need for realism with the demands for *control* and *precision* [200]. This trade-off is especially relevant in empirical studies that aim to validate advanced technological solutions in real-world environments. Our research prioritizes *realism* by instantiating our studies within the operational environments of case companies, capturing the complexities and challenges faced by industry practitioners. For instance, our implementation and evaluation of the *autonomous monitors* were carried out in a live production environment, where the complexity of cloud-based systems and the variability of operational conditions provided a robust test bed. This approach provided valuable insights into the operational effectiveness of the proposed monitoring solution, enabling a wide range of scenarios that would be difficult to achieve in a controlled laboratory setting.

However, this emphasis on realism introduced challenges related to the variability of industrial environments, such as fluctuations in system load, diverse data distributions, and configuration changes, which introduced noise and complexity. This variability made it challenging to control all conditions and isolate the direct impact of the implemented monitoring solution. Moreover, the noisy and dynamic nature of data from real-world environments might have impacted the precision of the results. Despite these challenges, we successfully balanced the trade-off by collaborating closely with case companies, refining our solutions based on continuous feedback, and integrating state-of-the-art solutions for anomaly detection and log analysis. While the variability in real-world settings may have led to some inconsistencies, it also provided valuable insights into the performance of our solutions across different operational contexts.

The aspect of realism differentiates our research from many recent publications in the same domain. Some of them use controlled environments [67, 234] or simulated operational failures [149, 212] for evaluation purposes. In contrast, our work exposed the tough realities of conducting empirical evaluation in real-world settings, where we experienced and addressed challenges related to the shortage of ground truth data and the interpretability of detected anomalies in monitoring data. Compared to the most similar solutions, such as *MonitorAssistant* [241] and *IntelligentMonitor* [209], our contributions still stand out with unique findings. *MonitorAssistant* [241] uses historical failures and their impact on metrics to guide anomaly detection and provide recommendations based on past patterns. As such information was not available within our case companies, we developed the *autonomous monitors* capable of detecting unforeseen anomalies and reporting actionable alerts that are tailored to the current operational context. Although the *IntelligentMonitor* [209] successfully addresses data overload and visibility of system health, our solution goes a step further by addressing the challenge of alert fatigue through the use of interpretable alerts generated by using LLMs. This en-

sures that the alerts are issued timely with clear and meaningful guidance, making them more practical and useful in real-world cloud operations.

Autonomous monitors offer significant advantages over the built-in anomaly detectors available in commercial monitoring tools [65] by providing a higher level of customization and precision tailored to the specific needs of a system. While commercial tools often rely on generic thresholds or basic algorithms, *autonomous monitors* use advanced ML models specifically trained on the system's monitoring data. Additionally, industry practitioners may be uncertain about how well these generic anomaly detectors will perform in their specific environments. Therefore, this thesis offers valuable insights into how to build custom solutions that integrate near-real-time data collection, anomaly detection, and the reporting of interpretable alerts, thereby providing evidence on how some state-of-the-art ML models perform in real cloud environments. This may help practitioners and researchers develop more effective and reliable monitoring systems that are adjusted to their operational contexts.

Building on the findings of this thesis, several future research directions can be pursued to further enhance and refine the proposed solutions. First, expanding the evaluation of the *autonomous monitors* across a wider range of industrial contexts would provide deeper insights into their adaptability and robustness. Additionally, exploring the integration of different and more advanced machine learning models could enhance anomaly detection capabilities. Investigating the use of real-time feedback loops with *automated adjustments* could improve the accuracy and efficiency of the monitors over time. Moreover, a deeper exploration of what constitutes *an anomaly* in different cloud-based contexts is essential, as operational norms and thresholds can vary significantly across industries and systems. Finally, addressing the scalability and performance of *autonomous monitors* in highly dynamic cloud environments, as well as ensuring compliance with regulatory and security standards, will be crucial for their widespread adoption and success in diverse operational settings.

5 Validity of research

In this section, we discuss the overall limitations of the design research in Papers I-III, using the assessment criteria of *relevance*, *rigor*, and *novelty* as outlined by Engström et al. [49]. We focused on balancing all three aspects to produce research that is both credible and practically valuable. The discussion on rigor will be enriched with an examination of potential threats to validity according to the definitions used by Feldt and Magazinius [54]. Both Paper IV and Paper V independently elaborate on their respective validity threats. Moreover, this section provides a brief reflection on the ethical considerations taken into account throughout the thesis.

5.1 Relevance

The relevance of the research conducted in Papers I-III is evaluated based on its applicability and benefit to both practitioners and the research community [182]. From the practitioners' perspective, the contributions offer practical solutions to real-world problems related to alert management in cloud-based systems. By addressing specific challenges identified through case studies, the research provides insights and solutions that can be adapted or directly applied in similar industrial contexts. The solutions proposed are designed to enhance operational efficiency, making them highly relevant to organizations facing similar issues. From an academic standpoint, the relevance is highlighted by the generalizability (see Section 3.6) of the findings, which contribute to the broader body of knowledge on machine learning applications in software engineering.

Additionally, in each of Papers I-III, the related work section critically analyzes previous studies that have explored similar research phenomena, identifying the research gaps. In this way, we demonstrate the significance and relevance of conducted research by showing how its main contributions build upon and extend the existing body of knowledge. Moreover, the survey study (Paper IV) conducted as part of this thesis provides a broad perspective on current monitoring practices, challenges, and the adoption of ML-based solutions. This study validates the relevance of the *autonomous monitors* but also highlights the industry's readiness and barriers to adopting such innovations, making the findings highly relevant to practitioners seeking to enhance their cloud-based systems.

5.2 Rigor

The rigor of the design science studies presented in this thesis is determined by assessing the knowledge-creating activities: problem conceptualization, solution design, and empirical validation [182]. Regarding the problem conceptualization, we used interviews, observations, documentation analysis, and informal discussions with practitioners to collect qualitative data and identify the most challenging problem instances of monitoring data flow and overflow in operations. These methods were systematically applied, following best practices in qualitative research, to ensure the reliability and validity of the data. Moreover, they provide direct insights into the studied phenomenon as they require quite close collaboration with the practitioners.

We assess the rigor of the design activity by estimating to which extent the proposed design builds on the prior designs from state-of-the-art solutions [49]. In Paper II, we provide an overview of the most relevant deep learning approaches that we considered for improving the initial version of the smart filter from Paper I. This involved a comprehensive review of a variety of anomaly detection techniques and their ability to capture complex, nonlinear relationships in multivariate time series data in real operational contexts.

Furthermore, the empirical validation of the proposed solution, the *autonomous monitors*, was conducted in a real-world environment using operational data to verify their effectiveness in a practical context. The feedback loop established with industry practitioners, who provided qualitative assessments of the *smart filter* performance, further enhances the rigor by incorporating real-world expertise and practical insights into the validation process. In addition, the survey findings confirmed the practical benefits of this ML-based monitoring solution in real-world settings.

In addition, we address several key aspects of validity threats to ensure the credibility of its findings. *Internal validity* is achieved by employing structured and systematic research approaches across the thesis, ensuring that observed outcomes are directly linked to the interventions. This is accomplished through iterative design cycles and empirical validations that rigorously test the interventions in real operational environments. However, researcher bias during interviews and observations poses potential threats, which were mitigated by using triangulation and incorporating expert (practitioner) feedback.

Construct validity is approached by ensuring that the concepts being investigated and measured, such as "*an anomaly*" are grounded in theory and practice. The use of diverse data sources, including interviews, observations, and literature reviews, ensures that they are accurately captured and assessed. Furthermore, the design and evaluation of the solutions were iteratively refined based on qualitative feedback from practitioners. Despite these efforts, we acknowledge that certain limitations exist, such as the potential for varying interpretations of what constitutes a "*real anomaly*" across different industrial contexts.

External validity is supported by diverse case studies (Papers I-III) and an industrial survey (Paper IV), enhancing the generalizability of findings across different contexts. Despite potential limitations due to specific company characteristics, the inclusion of various company sizes and sectors in the survey study aids in broadening the applicability of the results.

Conclusion validity is ensured through a robust data collection process, empirical validation, and careful documentation of the research methods. The iterative nature of the design science cycles, particularly in developing and refining the *autonomous monitors*, strengthens the reliability of the conclusions drawn. However, threats such as potential biases in the interpretation of qualitative data and the subjective nature of feedback mechanisms are acknowledged. We mitigate these by employing a systematic approach to data analysis and involving multiple stakeholders in the feedback process.

5.3 Novelty

The novelty of this research is evident in its innovative approaches and contributions to the field of cloud-based system monitoring. It particularly stands out for its fusion of machine learning, natural language processing, and software engineer-

ing principles to develop an advanced framework for system monitoring. Unlike many theoretical studies, this research goes beyond simulations and controlled experiments by deploying the proposed solutions in operational environments. This practical application allowed a thorough assessment of the models' performance in detecting anomalies within actual industrial data, providing concrete evidence of their utility and robustness. The integration of state-of-the-art solutions into operational workflows represents a significant innovation in how these models perform in real-world scenarios.

Additionally, the contributions of design research in the thesis were captured in *technological rules*. They provide guidelines that synthesize theoretical insights with practical applications supported by empirical evidence from case studies. These rules are designed with an emphasis on contextual relevance, making them applicable to a wide array of settings by allowing practitioners to adjust the solutions to fit the unique challenges and characteristics of each environment (see Section 3.6).

5.4 Ethical Considerations of the Thesis

Throughout the research conducted for this thesis, several ethical considerations were addressed to ensure integrity and ethical rigor. Confidentiality agreements were established to protect sensitive operational data and proprietary information, ensuring strict adherence to data privacy and intellectual property regulations. These agreements were important for maintaining the confidentiality of the case companies involved, and that all research activities adhered to strict corporate and academic standards. This included rigorous data protection protocols, ensuring that all collected monitoring data was securely stored and exclusively used for research purposes. For the qualitative components, including interviews and observations, informed consent was obtained from all participants. They were assured of confidentiality and anonymity, reinforcing the trust and privacy of those involved. In the survey component of the research, participants were informed about data privacy policies directly on the questionnaire form. This included detailed information on how their responses would be used, stored, and protected, ensuring transparency and securing informed consent. The ethical guidelines followed in this thesis reflect a dedication to responsible and respectful collaboration with all stakeholders.

6 Conclusions

In this thesis, we addressed one of the main challenges identified in DevOps, *alert flooding* (**RG1**). By leveraging available monitoring data and advanced machine learning solutions, this research has developed *smarter alert mechanisms* that provide critical insights into system operations (**RG2**). The introduction of

autonomous monitors, which employ both advanced ML and natural language processing (NLP) techniques, has significantly improved early *failure detection*. These autonomous monitors effectively reduce the volume of false positives, thereby enhancing the efficiency of the system's response to true anomalies. Through detailed case studies and empirical evaluations, the practical applicability and adaptability of these solutions have been validated across various industrial settings, showcasing their potential to transform and improve operational workflows in cloud-based environments (**RG3**).

The results also revealed a growing yet cautious interest among industry practitioners in adopting AI/ML technologies to enhance DevOps processes (**RG4**). While there is a clear recognition of the benefits, such as enhanced detection of operational failures and reduced manual efforts, concerns around implementation complexities and integration with existing systems persist. Moreover, the findings from this thesis highlight the importance of continued exploration and development in this field, particularly in refining anomaly detection models and expanding their application across different cloud-based environments. As the industry evolves, the adoption of advanced monitoring and alerting systems will be crucial for maintaining robust and reliable software systems, ensuring that DevOps teams can effectively manage and prevent potential failures. This research provides a foundational framework for these advancements, offering valuable insights into the integration of AI/ML in modern system operations.

INCLUDED PAPERS

PAPER I

CLOSING THE FEEDBACK LOOP IN DEVOPS THROUGH AUTONOMOUS MONITORS IN OPERATIONS

Adha Hrusto, Per Runeson, Emelie Engström

Abstract

DevOps represent the tight connection between development and operations. To address challenges that arise on the borderline between development and operations, we conducted a study in collaboration with a Swedish company responsible for ticket management and sales in public transportation. The aim of our study is to explore and describe the existing DevOps environment, as well as to identify how the feedback from operations can be improved, specifically with respect to the alerts sent from system operations. Our study complies with the basic principles of the design science paradigm, such as understanding and improving design solutions in the specific areas of practice. Our diagnosis, based on qualitative data collected through interviews and observations, shows that alert flooding is a challenge in the feedback loop, i.e., too many signals from operations create noise in the feedback loop. Therefore, we design a solution to improve alert management by optimizing when to raise alerts and accordingly introducing a new element in the feedback loop, a smart filter. Moreover, we implemented a prototype of the proposed solution design and showed that a tighter relation between operations and development can be achieved using a hybrid method that combines rule-based and unsupervised machine learning for operations data analysis.

Springer Nature Computer Science (2021)

1 Introduction

The software industry has gone through several revolutionary changes over the last decades. A major change is that software is no longer delivered as a box product. Technological advancements and availability of cloud computing platforms have enabled continuous delivery of *software systems* leveraging the flexibility and reliability of various *cloud* delivery solutions [171]. Moreover, cloud providers offer an infrastructure for developing and operating large-scale software systems empowered by continuous practices and DevOps, the latest industry concept based on principles of collaboration, automation, measurements, and monitoring [199]. However, it also comes with an abundance of data to be managed as it is considered to be the *fuel* of the DevOps process [20].

The software life cycle includes continuous integration, continuous testing, and continuous deployment practices [55]. During deployment, software systems are transitioned from development to operations, to be continuously used by end-users. The connection between development (Dev) and operations (Ops), known as DevOps, ensures faster development cycles and frequent releases. However, keeping the same level of software quality becomes challenging due to shorter testing cycles. Run-time monitoring of services in operations [53], which is the focus of this study, is of high importance for gaining confidence in a software system and providing feedback to the development.

Through the run-time monitoring system, a vast amount of data is continuously collected and saved for manual or automatic analysis. The data analysis serves as feedback to development teams and provides deep and quick insight into the status of the software system during operational execution [20]. Consequently, developers and project managers can act as soon as they are notified about anomalies. The notification is typically implemented as *alerts* sent through a messaging platform, like Slack, triggered by alert rules, which are defined as functions of the operational data. However, the abundance of data and particularly alerts from minor or major malfunctions in system components, tend to flood over the developers and create noise that drowns the important alerts.

In the literature, there are examples of various methods for the analysis of operations data but only a few are addressing real industrial needs and challenges companies are facing in relation to the feedback from operations to development [252]. Consequently, there is a limited choice of potential solutions available in the literature for designing more context-specific solution designs based on the identified industrial needs. *Thus, with our research, we aim to fill this gap by addressing challenges related to the flow – and overflow – of data from operations to development.* We intend to explore and improve existing solution designs in the context of the case company's feedback loop from operations to development. Thus our study complies with the principles of a design science paradigm [182].

We conducted a study in collaboration with a Swedish company responsible for ticket management and sales in public transportation. Their main product is

the back-end system for ticketing and payments, developed and operated in a DevOps environment using Microsoft services and tools. Following design science principles, we *explore and describe* the existing DevOps environment and identify the main challenges on the borderline between operations and development, using qualitative data collected through interviews and observations. To address the identified challenges, we *design* a solution for more effective processing of data available through the monitoring system in operations by introducing a smart filter in the feedback loop. Thus our research adds to the new research and innovation discipline called AIOps, artificial intelligence for IT operations [37]. Moreover, we present a *prototype implementation and validation* of the proposed design. It includes a description of the labeling process of unlabeled operations data, using unsupervised anomaly detection and considering the service vulnerabilities, as well as learning new advanced alert rules using a supervised, decision tree-based Python module.

The contributions of our paper are threefold:

C1. **Problem conceptualization.** We identified alert *targeting*, signal to noise *optimization*, and system *interoperability* as being three important problem instances of the general alert flooding problem in the feedback from operations to development.

C2. **Solution design.** We present a unique technical solution that combines various systems' and applications' metrics for learning advanced alert rules within the new element in the feedback loop, a smart filter.

C3. **Prototype implementation.** We performed a pilot implementation of the proposed solution in the case environment as a proof of concept for further work.

The rest of the paper is structured as follows. In Section 2 we present the background and previous work in this field. In Section 3, we elaborate on the research approach, while in Section 4, we describe the case company. Identified problem instances are introduced in Section 5. The solution proposal is presented in Section 6. Prototype implementation of the proposed solution and empirical validation are given in Section 7, while Section 8 discusses the contributions and concludes the paper.

2 Background and Related work

Ståhl et al. [199] conclude in their systematic mapping study on continuous practices and DevOps, that the concepts of continuous software engineering practices and DevOps are ambiguous in the literature. We adhere to their proposed definition that "**Continuous deployment** is an *operations practice* where release candidates evaluated in continuous delivery are frequently and rapidly placed in a production

environment". In contrast, "**Continuous release** is a *business practice* where release candidates evaluated in continuous delivery are frequently and rapidly made generally available to users/customers". Depending on the environment, a release may be achieved through deployment, for example in most SaaS (Software as a Service) environments. On the contrary, for user-installed software, continuous deployment is not an applicable concept, as the user must take actions to install a new version. However, continuous releases may still be offered to the users.

Ståhl et al. [199] find DevOps be a broader term, including culture and mindset. It also comprises tools, processes, and practices. We adhere to this broad definition of DevOps, as we want to investigate "the interplay between specific continuous practices and DevOps principles, processes and methods" [199], which aligns well with Fitzgerald and Stol's scoping of continuous software engineering [55].

Despite the observed ambiguity, there are additional research summaries. Laukkanen et al. [122] presented a literature review of problems, causes and solutions, when adopting continuous delivery. They build on a previous literature review by Rodriguez et al. [177], and summarize topics related to build design, system design, integration, testing, release, human and organizations, and resources. However, the operational aspects are not included. Similarly, Shahin et al. [189] do not cover practices beyond continuous deployment in their review and Mishra and Otaiwi [153] only briefly mention operational feedback as contributing to software quality in DevOps, in their systematic mapping study.

There is, however, research related to post-deployment activities. Suonsyrjä et al. [202] studied how automatically collected data from operations could be used as feedback to the development. They reviewed the literature and surveyed practitioners' interest in such activities. They conclude that topics related to post-deployment monitoring appeared in the scientific literature during the 20th century but, not during the last two decades [202]. As an exception, Orso et al. [164] presented the GAMMA system 2002, as an approach to support monitoring software's behavior during its lifetime.

Monitoring is not only focused on the software. According to Pietrantuono et al. [170], monitoring of the software product in operation can be used for collecting usage data. The data is afterward analyzed and reused for selecting the most representative test cases, based on usage profiles, which are used in their approach to "continuous software reliability testing".

Moreover, monitoring has also been part of alarm systems used for triggering warning signals in case of unusual rises in systems' metrics. Xu et al. [237] proposed a Process-Oriented Dependability (POD)-Monitor for reducing the number of false alarms focusing on sporadic and infrequent operations. Their approach utilizes process-context information and the Support Vector Machines (SVM) algorithm for learning when to suppress alarms and reduce the overload on operators.

Alerts is another term used for denoting the same or similar events as *alarms* and according to Zhao et al. [252], they represent a key source of anomalous events

in operations. Zhao et al. [252] reported an approach for handling alert storms consisting of alert storm detection using Extreme Value Theory (EVT), alert filtering using the ML Isolation Forest method, alert clustering using Similarity Matrix Construction, and representative alert selection. Furthermore, Zhao et al. [253] published another study on enhancing the quality of services by utilizing the monitoring data. Similarly, they analyzed alerts but with the aim of identifying the severity level. They proposed a framework AlertRank for extracting severe alerts based on textual and temporal alert features as well as features extracted from monitoring metrics. Since there are two different terms in the literature, in the rest of the paper we use *alerts* to denote signals of unexpected systems' behaviors in operations.

Monitoring in operations can be utilized even without alert rules, thus considering raw operations data. Cito et al. [30] identified three main categories of operations data: system metrics, application metrics, and application system metrics [30]. Recently, researchers and practitioners have devoted significant effort to the analysis of aforementioned operations data considering, among others, machine learning techniques and the development of various applications. Anomaly detection is one of the available applications for early detection of a system's abnormal behavior. It has been used for detecting deviations in software releases based on the data generated by a DevOps toolchain [19]. Further, Du et al. [46] presented DeepLog, a model based on deep learning for natural language processing, which is used for learning patterns in logs and detecting anomalies in log data. More thorough research on anomaly detection has been undertaken by He et al. [80] where they provide an overview of supervised and unsupervised machine learning techniques used for log analysis. In addition, logs have been studied for several other applications. Clustering log sequences into groups, identifying causal dependencies, and creating failure rules are the main steps in the root cause analysis and failure prediction approach proposed by Fu et. al. [59].

More attempts at problem identification by log analysis can be found in papers by He et al. [79] and Lin et al. [133] where KPI (Key Performance Indicators) are used in combination with logs. In both papers, the authors deal with clustering-based techniques, but their solutions differ in the second phase of the proposed approaches. In the solution by He et al. [79], the second phase consists of correlation analysis of identified clusters with system KPIs, while the second phase by Lin et al. [133] includes extracting most representative logs from clusters and comparison of clusters created in test and production environment for simpler problem identification. Furthermore, feedback from operations has been used for decision making and improving feature planning [30] as well as for feedback-driven development where monitoring data has been used for improving developer's tools [29].

In summary, operations data has been studied and analyzed for different purposes but still, there is more to be explored in DevOps contexts, to improve the feedback from operations to development. State-of-the-art solutions [19, 96, 252] address relevant challenges in managing operations data. However, situations of

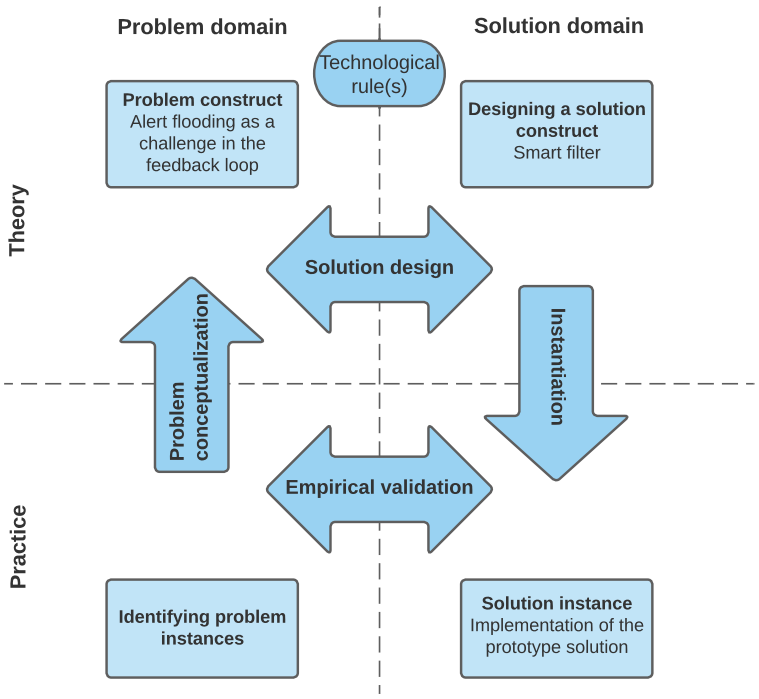


Figure 1: Overview of the design science approach

alert flooding in DevOps environments are not extensively explored. Thus, we aim to contribute to the design of solutions that better manage alerts in DevOps.

3 Research Approach

Our study, as shown in Figure 1, is a problem-driven design science approach [182]. Thus our starting point was to gain deeper insights into the specific challenges of our case company. As a first step, we explored how the general problem of incorporating feedback from operations in the development manifests as a *problem instance* in the industrial context under study. For that purpose, we conducted six interviews and performed observations in the case company to identify and articulate the main problem instances on which to focus further improvements.

To obtain a comprehensive overview of the issues, we selected interviewees in senior positions with different responsibilities within the team, including a product owner, a test manager, a test developer, a system architect, and two developers. During the interviews, we asked general as well as more specific questions related

to the DevOps cycle. The interviews were semi-structured since we wanted to flexibly explore the interviewee's opinions and let them speak about their main issues. Focus areas and examples of questions used in the interviews are shown in Table 1. All collected qualitative data, notes, and video records were analyzed using the NVivo tool. Furthermore, we observed their processes in operations and the way they were handling operations data. This enabled uncovering insights and defining problem instances.

In the *problem conceptualization* step, we described three identified problem instances (Section 5) through the lens of envisioned matching solutions, i.e., we formulated three high-level technological rules. However, in this paper, we refined only one of them in the conceptual *solution design*. Hence, we improve the feedback loop from operations to development by introducing a new element, a smart filter, for optimization of alert to noise ratio. In the design process, we considered the insights gained through interviews, results of the intensive discussions with the development team, and state-of-the-art solutions for alert management [252, 253].

Table 1: Topic areas and examples of questions used in the semi-structured interviews

Focus area	**Examples of questions**
CI/CD pipeline	- Could you describe the CI/CD pipeline? - What are the shortcomings and how can they be addressed?
Continuous monitoring	- Which parts of the system are monitored? - Which signals are the most critical and good candidates for monitoring?
Alerts	- How does the current alert system look like? - In which periods do you experience the highest number of alerts?
Accessibility of operations data	- Which types of operations data are available for analysis? - Which types of operations data are used for setting the alert rules?
Potential improvements	- How/what would you improve in your current monitoring system?

Moreover, alongside the proposed solution design, we implemented a prototype *instance* to get a better understanding of the opportunities of the available operations data, its type and characteristics as well as the constraints of the context. In the implementation of the prototype solution, we used unsupervised anomaly detection throughout the labeling process of unlabeled operations data while also considering the service vulnerability and observed metrics frequency. Further, for generating new advanced alert rules, a supervised tree-based machine learning technique was used. Regarding the *empirical validation*, there were time and environment constraints that hindered a full evaluation of the implemented solution. However, we were able to perform a partial evaluation using a limited data set for the implementation of the multivariate anomaly detection in a prototype environment. In this way, we were able to compare the results obtained by using the

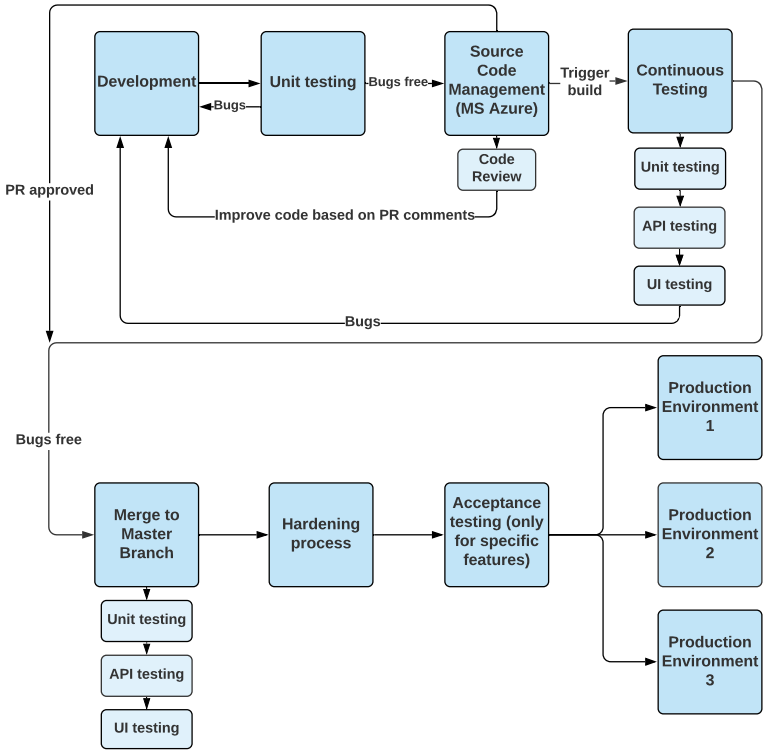


Figure 2: CI/CD pipeline

smart filter in the feedback loop with the results of using the pure unsupervised ML technique for predicting alerts based on the multivariate unlabeled data set.

4 Case Description

The system under study is a backend system of an application for ticketing and payments used in public transportation. It is a cloud-based system developed and operated in a DevOps environment using Microsoft tools and services. The system architecture is leaning towards a microservice architecture, which consists of 20 services that are highly maintainable, testable, and independently deployable.

Throughout the entire CI/CD cycle, shown in Figure 2, new features or updates of each service are tested on: 1) unit level, every time the build process of the system under test with its dependencies is triggered; 2) API and UI level, every time the master branch is updated as well as every night on the latest build version

Table 2: Types of operations data mapped with configured alerts

Operations data		Configured alerts
Logs	Exceptions	/
	Traces	/
	Requests	/
Application metrics	Dependency Failures	An unusual rise in the rate of dependency failures
	Exceptions	/
	Failed Requests	/
	Server Exceptions	/
System Metrics	CPU Time	/
	Errors Http 4xx	An unusual rise in the rate of failed HTTP requests
	Server Errors 5xx	Whenever there is a server error 500
	Response Time	/
	Requests	/

from the master branch. Moreover, the candidate version for the release is used as a reference version by other teams in the company for a week, which is called the "hardening process". If necessary, the latest version is tested in the acceptance-test environment, which serves as a production-like environment. The release cycle is weekly and ends by deploying to three production environments. Hence, the existence of several independent environments enables smooth development, testing, and deployment activities but also multiplies the complexity of the entire system.

The health status of each service is monitored using the Microsoft data platform, Azure Monitor. Azure Monitor collects the data from several sources, such as applications or Azure resources, into a common platform to be used for analysis, alerting, and visualization. Within this data platform, two types of data are available, metrics and logs. Metrics are numerical values denoting specific system's observations captured within a defined timestamp. Logs are represented by both numerical and textual values, and they describe specific events that happened at a particular moment in time. Both metrics and logs can be used for setting alert rules that signalize that something unexpected is detected in the observations of the targeted resources. The case company has implemented simple rules for detecting failed requests with error 500 and unexpected raises of dependency calls and failed Http requests, as shown in Table 2. When these rules are satisfied, alerts are triggered, and alert notifications are sent either to a dedicated Slack channel or via email.

Operations data shown in Table 2 represent only a small portion of all available data in Azure Monitor, but in this paper, we focus on the selected logs and

metrics. Among all accessible observations of different system components, we chose metrics and logs related to the data types used for setting current alert rules and the ones used in debugging in case of detected anomalies. Alert rules, shown in Table 2, are configured for all 20 services, and notifications about raised alerts are sent on two different platforms. Alerts that detect internal server error 500 are sent to the Slack channel, while unusual rises in the rate of dependency failures and failed requests are sent via email.

The development team has already reported various challenges in managing and responding to fired alerts with this configuration. Moreover, their everyday development tasks are filled with the uncertainty that every alert brings into their development environment due to an overload of non-relevant alerts. Consequently, this might cause a bottleneck in the information flow from operations to development. The flaws identified within the monitoring and alert system are elaborated in the next section.

5 Problem Conceptualization

In this section, we present three main problem instances, identified in the problem conceptualization step, with respect to the general goal of better incorporating feedback from operations into development. Based on observations made in the case company, *alert flooding* is identified as the main cause of all three problems. Alert flooding is a phenomenon that appears in the case of a high number of alerts that are not properly managed. In this paper, we focus on the specific aspects of this phenomenon namely, *targeting, optimization*, and *interoperability* problems.

5.1 Alert flooding as targeting problem

The first problem is defined as a targeting problem. This means that the distribution of alerts to target recipients, between the teams and individual assignment of a single or group of alerts within the team, is not fully transparent. Moreover, a lot of time is spent on discussions on how to resolve alerts and who is going to take the responsibility. Currently, three teams can be assigned when an alert is fired. Each team consists of four or five members, mainly developers, and every team is responsible for one of the domains, which consist of multiple services. Alert notifications are sent to a dedicated Slack channel, but no one is tagged or directly assigned to the raised alerts. Individual responsibilities within the team are not clear and team members usually discuss specific alerts in the same Slack channel. Sometimes they tag each other and ask if that person has already looked into raised alerts. As acknowledgment, they usually write that they will look at it right away or later. If they agree that an action should be taken, a ticket is created and added to a backlog of the board in Azure DevOps. Hence, two different platforms for communicating alerts are used but the information is not synchronized.

While observing the team and their current practices, we noticed that some team members showed more interest than others in resolving alerts and that some look into alerts that are related only to services they are developing or they are familiar with. Consequently, there is an increasing number of alert notifications because no one takes full responsibility for looking into alerts that frequently appear every day. After talking to some team members, it was clear that they would like to see some structured way of alert management and assignment but they also pointed out that acting on every alert would take too much time since their main focus is development. Because of that, designing a solution for the targeting problem becomes even more challenging.

5.2 Alert flooding as optimization problem

The second problem instance represents an optimization problem, which addresses optimization of a signal to noise ratio. In this case, the signal consists of high priority alerts while the noise represents low priority alerts, which frequently appear every day. Hence, the main question is how to differentiate between alerts that cause failures and alerts that cause temporary glitches that don't affect the system's performance.

While observing the current practices in alert management, we noticed that all alert notifications come to the Slack channel with the same priority. Over time, developers learned which alerts are reoccurring occasionally, and they consider them as "normal alerts". Normal alerts are mostly caused by glitches in an external or internal service or represent a consequence of a failure related to the central service. The central service represents the heart of the system and all alerts related to this service have the highest priority. This priority is not specified as a part of an alert notification, but is something that developers know since they developed the system and they know how vulnerable each of the services is. "Normal alerts" are not normal since they signalize that something might be wrong in the specific service, but they are normal as they occur frequently, and the team got used to them. They also produce noise in the channel used for communicating alerts and because of that some critical things may pass unnoticed. The team raised concerns about this and agreed that addressing and solving this particular problem might help in faster and better response to other more important alerts. One more reason to do so is because they currently do not act upon normal alerts unless there is a high number of occurrences.

The majority of current alert rules aim at discovering internal server errors with error code 500 while a significantly higher number of logs still remain unexplored, Table 2. Hence, there is a need for adding more alert rules. However, the team decided to stick with the existing alert rules since the current ones are not successfully managed. Recently, the team reported that they missed over 20 000 failed Http requests with error code 400. They did not notice this anomaly because they were overwhelmed with other alert notifications but also due to the fact that

they do not usually analyze logs or fix issues before they cause severe problems. Hence, designing new or redesigning existing alert rules to optimize the signal to noise ratio, is another challenge that they are facing while at the same time it is important that the number of non-relevant alerts is not increased and that the most critical alerts are prioritized.

5.3 Interoperability flaws between developed system and external systems

Many large-scale software systems depend on external services developed by third parties. In this way, the original system can offer more features to their end customers. This seems to be a huge benefit but may also increase the vulnerability of the entire system since even the smallest glitches in an external service might cause serious deviations in the original system. Similar issues are experienced in the case company as their backend system also depends on external payment providers, Azure databases, and other software projects developed in their company. There is a special Slack channel where RSS (Really Simple Syndication) feeds and emails from external services are forwarded. However, many problems are still discovered through customer service and user complaints. So, they get notified when something has already failed and is visible to end-users instead of in advance. Moreover, the uncertainty of potential disruptions makes developers even more confused. It is their responsibility to decide if a raised issue is something temporary or it really represents an issue they should look into and report. They usually make a decision based on the alert frequency and side effect appearance. There are no statistics that can prove developers' claims, but a huge number of alerts are caused due to interoperability flaws with external services. The existence of failed Http responses with unknown and unexpected error codes complicates root cause analysis even more. It is important to address this problem, otherwise the system stability will be degraded.

6 Solution Design

As stated in Section 3, we provide a conceptual design for the second problem instance, *alert flooding as an optimization problem*. This problem causes the highest information overflow in the feedback loop. By addressing this specific instance, the scope of the first and the third problem instances will be reduced, and individual solutions simplified. The first and the third problem instances will not be individually treated in this paper but will be considered in our future work.

Hence, we propose one solution design and focus on the following challenges related to the second problem instance: 1) reduce the number of noisy alerts without missing the critical ones; 2) increase the number of alert rules without causing an overload of alert notifications; 3) improve developer's responses to the fired

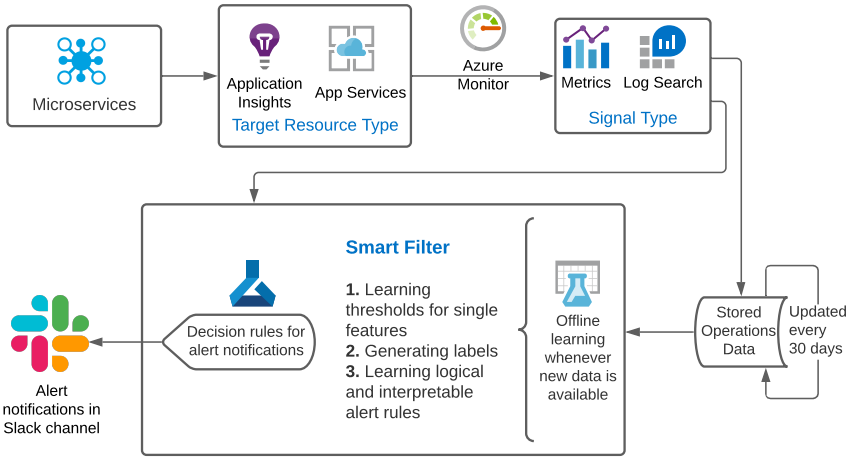


Figure 3: Overview of the proposed solution for the second problem instance

alerts while minimizing interference with their development related tasks. Accordingly, we present the overview of the proposed solution for the second problem instance in Figure 3.

The upper part of Figure 3, illustrates the previously explained architecture of the software system, consisting of 20 microservices and Azure Monitor, that monitors real-time application performance (Application Insights) and performance of Http-based services for hosting applications (App Services). The lower part of Figure 3, visualizes the enhanced alert system with a new addition, representing the bridge between MS Azure Monitor and Slack, the platform where alert notifications are sent. The new box, the smart filter, serves as a middle-ware and provides additional features to the existing alert management.

The main task of the introduced box is to generate alert rules for sending alert notifications to the messaging platform. Hence, we temporally disregard current alert notifications and instead focus directly on the most important data, specifically metrics shown in Table 3, holding information about the system's performance. The reason for such an approach is that the current alert rules only catch a limited number of system glitches and failures while at the same time not being able to differentiate noisy alerts from important ones. The smart filter will analyze more data and learn over time to identify new dependencies that may generate new and better decision rules. In this way, we will reduce the risk of omitting important alert notifications while keeping the Slack channel clean from noisy information. Therefore, in our proposed solution design, new decision rules are learnt based on the features representing the systems' and applications' performance metrics

Table 3: Overview of the selected data, service vulnerabilities, and desired decision rules

Selected application and system metrics	- CPU Time - Number of failed requests - Number of exceptions - Number of dependency failures - Http 4xx errors - Internal server errors - Total number of requests - Response time
Services with known vulnerabilities	- Service B –>buying tickets on vending machines - Service G –>service for validating selected locations - Service M –>main service for ticketing - Service P –>bridge to an external payment service
Example of a decision rule	IF num_of_failed_requests_SG >threshold_1 AND response_time_SB >threshold_2 AND num_of_Http500_SB >threshold_3 THEN send_notification

of the mostly affected services. The output of the smart filter is binary, meaning that new decision rules are able to determine when to send and when not to send alert notifications. As shown in Figure 3, the smart filter involves preprocessing and labeling of the data required for the learning process. The exact procedure is presented in Section 7.

All things considered, the proposed approach of generating new decision rules aims at filtering the incoming performance data and sending only relevant alert notifications to the Slack channel. Newly learnt alert rules should not increase the number of alert notifications in the Slack channel since the learning process also involves learning about the noisy data.

Therefore, the proposed solution design addresses the aforementioned challenge regarding the insufficient alert rules. The purpose of the enhanced alert management is to provide more insights into correlations between alerts and operations data and at the same time enable forwarding more details about potential failures within the alert notifications. In this way, the development team could have all information needed to discover the root causes of potential failures. Moreover, it is expected that developer's awareness of raised alerts will increase and that they will need less time for resolving critical systems behaviors. Therefore, the proposed solution design intends to resolve the previously listed challenges related to the second problem instance.

7 Prototype Implementation and Empirical Validation

In this section, we present technical details of the prototype implementation[1] as well as the effects of the implemented solution prototype in the identified problem context. Prototype implementation includes *data selection*, *tools and methods selection*, *threshold detection for each of the features*, *labeling process*, *training process* and *testing*. While working on the implementation of a solution prototype, we have decided to stick with basic machine learning techniques since we primarily wanted to examine the limitations of the suggested design. Hence, using deep learning or reinforcement learning for identified problem instances is beyond the scope of this paper.

Data selection. For the prototype implementation, we have chosen to only work with numerical values representing the various systems' and applications' performance metrics, to keep the simplicity. Logs are not included in the preliminary data selection due to their complex structure and due to the fact that the observed logs including traces, types of exceptions, or failed requests could only help with the explainability of potential failures. The metrics and services selected to be part of the training data (see Table 3) are chosen based on the observations made in the messaging and monitoring platform focusing on metrics frequency and service vulnerability. Therefore, we selected 8 metrics for each of the 11 services, which makes in total 88 features. Every feature vector has 8623 samples collected during a period of one month with a time granularity of 5 minutes, which was selected based on the current practice within the project.

Tools and method selection. The presented solution design involves learning new decision rules in the form of logical expressions "IF conditions THEN response" and for such an approach the first choice of ML methods are tree based methods, such as bagging and random forest. Therefore, for implementation, we use Skope-rules [61], a Python machine learning module for extracting rules from the tree ensemble as suggested by Friedman and Popescu [56]. The classification is binary, thus, if an instance representing the combination of multiple features satisfies conditions of the rule, then it is assigned to one of two output classes, "send_notification" or "dont_send_notification". Using this Python module requires labeled data for the learning process, thus making this approach even more challenging since the monitoring data platform collects only raw data and the knowledge about the expected outcomes is unknown.

Identifying thresholds. Therefore, we decided to generate labels based on the known service vulnerabilities and desired level of contamination. The first step of the labeling process is to identify thresholds for single features using machine learning for anomaly detection (see Figure 3, step 1). For that purpose, we used a Python toolkit PyOD [255] consisting of 30 different detection algorithms. Hence,

[1] `https://github.com/adha7/smart-alert-filter`, available upon request

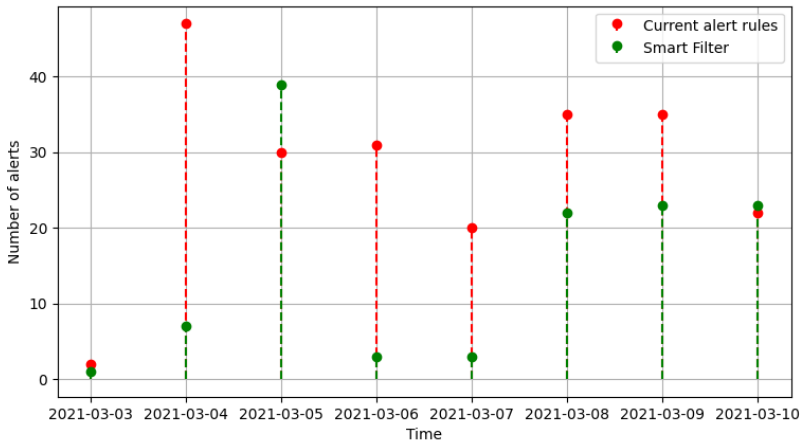
the thresholds are predicted for each of the 88 features where the outliers are expected to be extremely high values. By applying one of the algorithms from the PyOD module on a feature vector, we get anomaly scores for each of the values within a feature vector. Larger anomaly scores are assigned to outliers and the threshold is simply determined by picking a value from a sorted feature vector with a large enough score. The score value on the borderline between inliners and outliers is chosen so that the level of contamination of the entire training data equals 0.05. The contamination is determined by the number of outlying objects in the data set, in our case alert notifications that need to be sent to the messaging platform. Selected level of contamination corresponds to the 13 alert notifications per day and represents three times less of the current number of alert notifications. Since there is no optimal number of alert notifications per day we consider this decrease significant and at the same time large enough to not miss the important system failures.

Labeling process. After determining the thresholds for each of the features, the warnings are raised in the cases where the features reach values above these border values. Based on these warnings, we generate labels (see Figure 3, step 2) considering a fixed number of raised warnings in a time slot of 5 minutes as well as capturing for which services warnings are raised, targeting services shown in Table 3. Accordingly, the output class is labeled as 1, if there are more than 8 raised warnings in the same time slot, which means that there are at least two services affected considering that 8 warnings can be related to one service. Further, the output is also denoted as anomalous or 1, if there are warnings raised for the most vulnerable services, as shown in Table 3, no matter the number of raised warnings. When the labeling process is completed, learning logical and interpretable alert rules can be activated (see Figure 3, step 3).

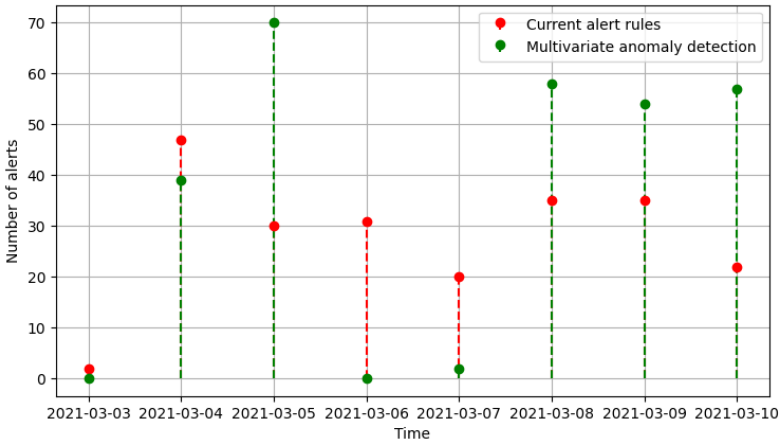
Training process. Through the training process, Skope-rules generated 120 rules for the class "dont_send_notification" and 43 rules for the class "send_notification". The rules are generated by fitting single estimators, decision trees, with predefined precision and recall as input parameters. The precision and recall reached during the training phase are between 0.92 and 0.99 for the output class "dont_send_notification". The precision score for the output class "send_notification" is evenly high as for the opposite class but the recall was significantly lower due to very low contamination, the number of outliers, in the training data set. A low recall score makes the algorithm "picky" when selecting outlying samples which might be good for filtering the noise but on the other hand, it might miss single and isolated outliers.

Testing. On this account, we analyze how the implemented prototype scales the number of predicted alert notifications per day to the actual number of raised alerts. We use test data collected within the 7 days (March 3, 20:35 – March 10, 19:40) for predicting outlying objects, alerts, and present the results in Figure 4.

We conclude that the smart filter produces half the number of alerts in a period of 7 days, 108 compared to 211. Regarding the distribution of alert notifications



(a) Smart filter



(b) Multivariate anomaly detection

Figure 4: Number of alerts per day in the test data. RED color: alerts raised with current alert rules; GREEN color: alerts raised with a) the smart filter and b) multivariate anomaly detection

per day, the number of predicted alerts during the weekend (March 6 and 7) is very low which is expected due to lower stress on the ticketing and payments system. During the workdays, the number of predicted alerts is less than actual except when there are issues in the system that the current alert system is not able to capture. This was the case on March 5, when there was a problem with buying tickets on the vending machines. The smart filter raised an alert 30 minutes earlier than it

was reported by customers, which means that this specific failure could have been caught before it was noticed by users.

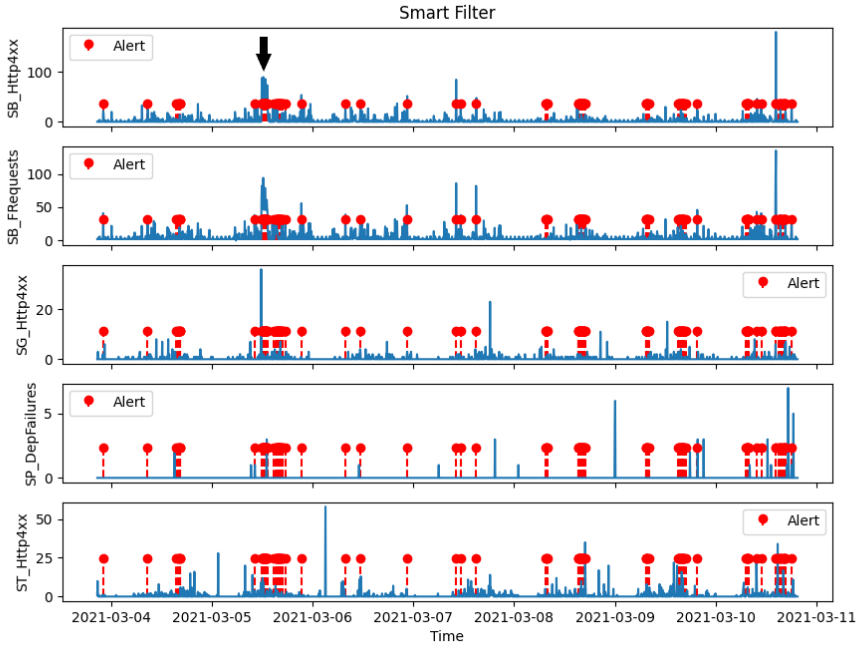
The implemented prototype reduces the overall overload on the development team but also gives space for further improvement by introducing prioritization of alerts and sending the alerts on different Slack channels based on their priority for even better and clearer differentiation.

Empirical validation. In addition to the smart filter implementation, we also implemented multivariate anomaly detection (MAD) to validate our prototype by comparing it with the pure unsupervised ML technique for detecting outliers, representing alerts, in multivariate unlabeled data set. We used the same Python toolkit PyOD [255] for the MAD implementation and selected the COPOD model, copula-based outlier detection introduced by Li et al. [130]. The COPOD model was trained using the same training data but without labels. The predictions, shown in Figure 4 (b), using the same test data set, revealed that the MAD trained model does not scale very well the number of predicted alerts. It predicts almost the same number of alerts as the actual alert system, making the same level of noise. Both models, trained using the smart filter and MAD respectively, reach the F1-score, a harmonic mean of precision and recall denoting a model's accuracy, above 0.9. However, the pure unsupervised ML might not be able to capture the imbalance between the target classes and the importance of specific services and their metrics. To clarify this, we look at the alert distribution over the metrics of highly affected services shown in Figure 5 (a) and (b). We noticed that the smart filter produces less noise around the actual failures, such as the one marked with the black arrow from March 5. This means that the actual failure can be more easily identified among the alerts that appear close to the selected alert on the graph. The predicted alerts using multivariate anomaly detection are grouped and based on the graph, they produce several alert floods which is the opposite to what we want to achieve. On the other hand, the smart filter predicts isolated alerts in case of short system's glitches and smaller groups of alerts when there is a larger issue rolling out.

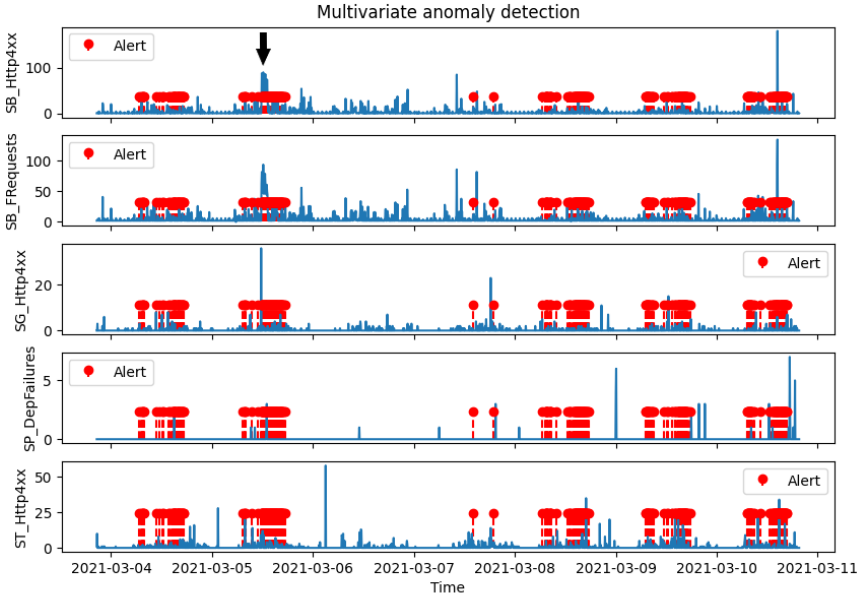
There are still some individual events that passed unnoticed but since this is only a prototype version, imperfections and shortcomings are expected. Furthermore, we used a limited data set collected within one month, which could have also affected the training process and learning when to send alert notifications due to a low number of outlying objects. We aim to address this in our future work by considering the larger data set.

8 Discussion and Conclusion

The synergy between development and operations in DevOps is important for developing and releasing high-quality software systems, but even more for gaining insights into the system's behavior in the production environment. In order to ensure the latter, raw operations data, collected through runtime monitoring tools, is



(a)



(b)

Figure 5: Distribution of raised alerts in the test data using a) the smart filter and b) multivariate anomaly detection. BLUE color: selected performance metric; RED color: raised alerts

analyzed to discover valuable feedback information. Our results have shown that monitoring and utilizing data available in the production may help developer teams to more easily identify, understand and communicate issues in the operations. Further, it helps present the valuable information in an actionable manner and reduces the pressure and overload.

The results obtained, following design science principles, directly relate to three main contributions mentioned in the introduction section, problem conceptualization (C1), solution design (C2), and prototype implementation (C3). We started with the problem conceptualization since the first step in solving a particular problem is understanding its causes and effects. Before our attempt to identify the main challenges on the borderline between development and operations, the everyday routine work at the case company obscured shortcomings in the information flow between operations and development. During the initial stage of interviews and observations, we managed to identify *targeting, optimization* and *interoperability* problem instances related to *alert flooding*. The problem conceptualization (C1) helped both the development team in acknowledging existing issues and the research team, in creating a solution design, which is our second contribution. After presenting our findings, the development team seemed relieved since they finally understood what was hindering them from making full use of operational data and how data overload in operations could be prevented.

The solution design (C2), as previously mentioned, addresses the problem of alert flooding with the emphasis on reducing the number of noisy alerts. The presented conceptual model includes a new element in the feedback loop, responsible for learning new advanced alert rules capable of reducing the total number of alerts and increasing their relevance. The smart filter addresses challenges in the alert management such as insufficient number of alert rules, noisy alert notifications, and slow developer's response on fired alerts. Therefore, this addition in the feedback loop improves the information flow from operations to development by introducing alert rules which combine various systems' and applications' metrics and services with the aim of capturing unexpected and faulty system's behaviors and providing more detailed insights to the development team.

The third contribution (C3) includes implementation of the solution prototype and validation in a specific context, i.e. our case, the ticketing and payment system operated in the DevOps environment. We successfully implemented a prototype version of the smart filter using a hybrid method consisting of unsupervised anomaly detection and supervised decision tree-based Python toolkit while also considering the importance of highly vulnerable services in the labeling process. The prototype was validated using a limited test data set collected through the monitoring system in the production environment. Accordingly, we demonstrated that a severe failure could have been caught if the smart filter was integrated in the feedback loop instead of the current alert system. Furthermore, we compared the implementation of our prototype with the pure unsupervised ML technique for multivariate anomaly detection. We showed that the customized hybrid method

better captures the systems' unbalanced operations data and system-specific characteristics needed for catching both systems' glitches and severe failures. Hence, the feedback information obtained as a final result has tightened the connection between operations and development. There have been several attempts at addressing similar challenges using state of the art solutions based on deep learning [46,96,252], while our solution proposal reach promising results while keeping simplicity of the ML approach.

The smart filter in the feedback loop improves the connection between operations and development but at the same time raises more challenges that need to be addressed in the future. Even though it reduces the total number of alerts, it could still be improved by increasing the level of differentiation between the raised alerts by introducing several levels of priorities and target recipients. We plan for further work to address the raised challenges by considering deep learning and other machine learning techniques as well as implementing the smart filter in the production environment. Consequently, the smart filter will be fully integrated and automated in the feedback loop and will require minimum human assistance. In this way, we would be able to get immediate feedback and insights from developers involved in the alert management, which is needed for obtaining a complete evaluation of the smart filter. Moreover, since our study provides prescriptions for problems in a very specific industrial context, in the future we aim to validate our solution in other similar contexts.

Acknowledgments

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. We thank the DevOps teams for their willingness to share insights and respond to our questions.

PAPER II

TOWARDS OPTIMIZATION OF ANOMALY DETECTION IN DEVOPS

Adha Hrusto, Emelie Engström, Per Runeson

Abstract

DevOps has recently become a mainstream solution for bridging the gaps between development (Dev) and operations (Ops) enabling cross-functional collaboration. The DevOps concept of continuous monitoring may bring a lot of benefits to development teams, such as early detection of run-time errors and various performance anomalies. We aim to explore deep learning (DL) solutions for detection of anomalous systems behavior based on collected monitoring data that consists of applications' and systems' performance metrics. Moreover, we specifically address a shortage of approaches for evaluating DL models without any ground truth data. We perform a case study in a real DevOps environment, following the principles of the design science paradigm. The research activities span from practice to theory and from problem to solution domain, including problem conceptualization, solution design, instantiation, and empirical validation. We proposed and implemented a cloud solution for DL model deployment and evaluation empowered by feedback from the development team. The labeled data generated through the feedback was used for evaluation of current and training of new DL models in several iterations. The overall results showed that reconstruction-based models, such as autoencoders, are quite robust to any parameter modification and are among the preferred for anomaly detection in multivariate monitoring data. Leveraging raw monitoring data and DL-inspired solutions, DevOps teams may get critical insights into the software and its operation. In our case, this proved to be an efficient way of discovering early signs of production failures.

Information and Software Technology (2023)

1 Introduction

The complexity and dimensionality of software systems are continuously increasing to meet higher market demands and customer expectations. Developing such systems is even more empowering in DevOps environments where collaboration, automation, measurement, and monitoring [145] are the main principles for eliminating the barriers between development and operations teams. DevOps is an emerging cultural and organizational practice that enables adoption of new and modern software architectures for developing scalable and reliable applications. The recent and popular architectural style entails microservices as an approach towards developing applications as a suite of small services [6]. The main advantage of those services is that they are independently developed, tested, deployed, and maintained [210].

In this paper, we report a case study of a microservice system for ticket and payment management in public transportation, in a DevOps environment. We specifically focus on infrastructure automation for building, deploying, and operating microservices, known as Continuous Integration / Continuous Deployment (CI/CD) pipelines [210]. The overall popularity of microservices has increased with the availability of various cloud platforms and solutions that enable building such infrastructures, required for flexible and frequent releases. Fast releases and their quality are the priority in DevOps environments. The DevOps concept of continuous monitoring in operations assists in revealing unexpected and unwanted system behavior. In our study, we focus on the detection of anomalous system behavior in monitoring data and timely reporting alerts to development, providing all details needed for fast response.

We perform the case study, adhering to the principles of the design science paradigm [182], as a continuation of our previous work [89], which conceptualized the problem and proposed a solution, expressed as technological rules, i.e., grounded recommendations for practice [182]. The initial results of the case study were presented at the workshop SESoS 2022 [87], which we in this paper extend by refining the technological rule and implementing and evaluating proposed solutions. Specifically, we aim to dig deeper into performance vulnerabilities of a microservice system and how they may be discovered early by raising alerts before they turn into a problem with severe consequences. For that purpose, monitoring in operations provides direct insights into the status of the running software system, while all monitoring data is continuously saved for potential further analysis. According to experiences from the industry and reported challenges in managing monitoring data [171], many software companies are still struggling to select the right set of tools for storing, visualizing, and processing the data. Having a unique solution for monitoring data management would enable timely identification of any potential deviations in data and reporting them as *alert notifications*.

Additionally, the health status of the entire system depends not only on the health of individual services but also on their relation and coexistence in a software

system as a whole. This means that the entire software system is based on the interaction between the services, which may not only be dependent on each other but even on another external service, hosted on a different cloud platform. To address this complexity, we treat the data from different services as multivariate time series (MTS) data represented by an ordered set of multidimensional vectors, recorded at a specific timestamp [27]. There have been several attempts at dealing with multivariate time series data with the aim of identifying anomalous systems behavior [96,123,148]. However, only a few discuss the evaluation of the proposed methods in contexts without known ground truth data. Evaluation of the learning model is a required step to achieve a reliable AI solution [120]. Thus, we aim to fill this gap and explore approaches for evaluation of deep learning solutions for filtering out anomalies in unlabeled multivariate time series. The novelty of the conducted research is in *the adaptation to the real industrial context*. More specifically, we defined and implemented guidelines for selecting and evaluating DL solutions in the aforementioned context. Expressed as a technological rule:

> **TR**: To improve alert management in DevOps environments, integrate a smart filter based on DL for anomaly detection in operations, and iteratively evaluate and update it utilizing generated labeled data through feedback from development.

The technological rule (TR) above is a refinement of the one identified in our initial exploratory study [89]. The initial solution design included an additional element in the feedback loop for anomaly detection, a smart filter, implemented using a basic ML approach based on decision rules [89]. The continuation of this work was published in our most recent paper [87], where we considered advancing the design of the proposed solution using unsupervised deep learning methods to meet the complexity of the studied system and the unavailability of annotated anomalous data. Contributions of the most recent work [87], (**C0** and **C1** listed bellow) are also included in this paper to keep the coherent structure. Thus, this study is an extension of the previous work [87] with an additional contribution **C2**, where we investigate six different variations of the unsupervised DL methods for anomaly detection in multivariate time series. Furthermore, we iteratively evaluate and update the best-performing DL models in the same DevOps environment (see Section 4), which we refer to as *optimization*.

The contributions of our paper are threefold, which strengthen the grounding of our technological rule by being field tested in our case study:

- **C0**: **A conceptualization of the problem** of applying and evaluating deep learning methods in a challenging industrial context that includes multidimensional time series data with unknown ground truth anomalies.
- **C1**: **A brief overview of unsupervised deep learning methods** for anomaly detection in multivariate time series representing performance metrics of a

microservice system as well as guidelines for selecting minimum feasible DL methods for anomaly detection in the same or similar contexts.

- **C2**: A **cloud solution** for **deployment** of the DL method for anomaly detection in multivariate time series and its in-context **implementation**, **evaluation**, and **optimization** through continuous feedback from development.

The rest of the paper is structured as follows. In Section 2 we present background and related work in the field of monitoring large-scale cloud applications and anomaly detection. Next, in Section 3 we give an overview of the research approach while in Section 4 we describe the case company and the system under study. We present an overview of unsupervised deep learning methods for multivariate time series in Section 5 and give guidance on selection minimum feasible DL methods in Section 6. An overview of the cloud solution used for deployment and integration of the DL model into the feedback loop with the implementation details and results is given in Section 7. Finally, we discuss limitations of proposed solution design, conclude and briefly discuss future work in Section 8.

2 Background and Related Work

Continuous monitoring is a very important post-deployment activity that provides insights into software usage and health. Collecting and inspecting the raw monitoring data is highly significant for diagnosing potential failures. Identifying and resolving prospective issues on time, improves overall quality and reliability, and reduces potential technical debt [6]. Surveying state-of-the-art literature, Gall and Pigni [60] identified the benefits of using monitoring such as quick incident response, transparency, and quality of services. These goals are achievable through analysis of fine-grained metrics and end-to-end correlation, highly significant for microservice architectures and capturing the multidimensional view of the software [60].

Regarding the monitoring of microservices, the general background may be found in publications that explicitly treat this type of systems [28, 104] as well as in papers that rather focus on cloud monitoring while implicitly introducing challenges on monitoring microservice architectures [75, 171], due to their very strong synergy. Ying et al. [104] claim that monitoring has an important role in a microservice system and that it can be conducted on several levels: hardware, network, system, application, and service access level. They propose a monitoring scheme that consists of the following platforms: 1) Logstash for data collection and filtering; 2) Elasticsearch for storage analysis; 3) Kibana for visualization. In a broader overview on monitoring of microservices, Waseem et al. [229] introduce monitoring practices such as exception tracking, health check API, and log management instead of monitoring levels. Moreover, they bring reasons for having a monitoring infrastructure for microservices, including diagnosing and reporting errors, failures, and performance issues, which is also the aim of our study.

From a slightly different perspective, focusing on monitoring cloud systems, Pourmajidi et al. [171] present a similar monitoring framework as Ying et al. [104] with an additional suggestion that includes a stack of InfluxDB and Grafana, open-source time series database, and analytics and interactive visualization web application, respectively. Moreover, one of the challenges they are concerned with is that the monitoring frameworks provide the health status of individual components instead of the overall system [171], which is a concern we are partially addressing in this study.

Further, there is a need for a unified monitoring framework to replace a stack of different tools as in the aforementioned examples. Larger companies providing cloud services, such as Microsoft and IBM [89,96], have already started offering a unified platform that enables version control, building, deploying, and monitoring of applications.

Data collected through the monitoring system may take different forms, but we specifically focus on multivariate time series, mainly denoting various applications' and systems' performance metrics. Mining the time series data aims at extracting meaningful knowledge using tasks such as classification, clustering, forecasting, and anomaly detection [27]. Anomaly detection is the process of identifying unusual and unexpected events across time [14]. Hagemann and Katsarou [75] summarize anomaly detection methods for cloud computing environments into three categories: machine learning, deep learning, and statistical approaches. Recent studies [96, 148] show that the researchers were mainly interested in deep learning methods due to their ability to learn highly complex and non-linear correlations with no prior assumptions about the data. For instance, Islam et al. [96] argued that Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) autoencoders have the strongest predictive power. Among other researchers, there is the same tendency when selecting anomaly detection methods for multivariate time series [3, 96, 148]. Another method attracting growing attention is Generative Adversarial Networks (GANs) [12, 123] that belongs to reconstruction-based methods like the aforementioned autoencoders. Similar to autoencoders, GANs aim at reconstructing the normal behavior of time series and detecting anomalies based on how much the reconstructed data points deviate from the normal behavior. In Section 5, we provide an in-depth review of deep learning methods for the studied context.

3 Research Approach

We conducted a case study in the context of a larger research project using the conceptual framework of the design science paradigm [182], as shown in Figure 1. Our previous work [87], (in *blue*, Figure 1) and current work (in *green*, Figure 1) are part of the same case study. The starting point of this case study is the bottom left quadrant (I), with an identified problem instance of *alert flooding* and

outcomes from the design science cycle conducted in our initial study [89]. In this study, we aim to further investigate one of the technological rules, that maps the problem of *alert flooding* to a solution that optimizes alert to noise ratio, by adding a smart filter to the feedback loop from operations to development. We revisited the previously published initial, basic version of the smart filter [89] by gathering observations of its limitations from the DevOps practitioners, and surveying relevant literature. In a design science cycle [182], this step corresponds to the problem conceptualization activity and constitutes contribution **C0** listed in Section 1. As shown in Figure 1 (quadrant II), we reduced our challenging industrial context to a problem of *learning from multidimensional time series data with unknown ground truth anomalies*.

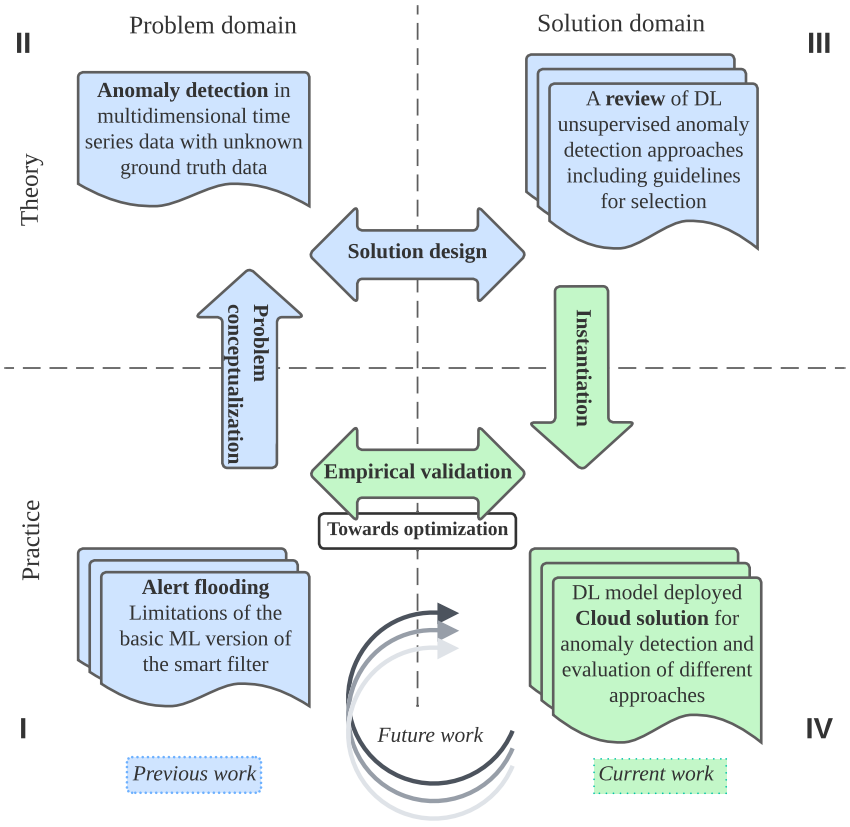


Figure 1: Overview of the research approach in the design science framework (*blue*: previous work; *green*: current work)

Next, we investigate solutions for an upgrade since the initial implementation of the smart filter [89] relied on a supervised anomaly detection method and manually labeled data, which was insufficient to fully characterize all notions of anomalousness [181]. Hence, we wanted to explore unsupervised deep learning approaches that are capable of learning the complex dynamics with no prior assumptions about the anomalous and non-anomalous data distribution. Thus, in this step we design an upgraded solution of the smart filter, inspired by available solutions in the literature, which is stated in contribution **C1** in Section 1 (see Figure 1, quadrant III). Hence, we approach the literature with the following research questions:

- **RQ1**: What unsupervised deep learning methods are mostly used for anomaly detection in multivariate time series?
- **RQ2**: What unsupervised deep learning methods for anomaly detection in MTS may be used in the studied context?

Eighteen techniques were compared through a 3D lens proposed by Choi et al. [27]. The details of this analysis are reported in Section 5 and summarized as *generic guidelines* in Section 6. As shown in Figure 1, these two research questions were already addressed in our previous work [87]. Unchanged findings are included in this paper.

The next step is towards optimization, and we continue from the cloud solution plan that is also presented in our most recent work [87]. It presents the parts of a cloud infrastructure needed for integration of the upgraded deep learning version of the smart filter in the feedback loop from operations to development. In this paper, we expand this section and provide details of the *in-context implementation*, including deployment of DL models, generating labels, and *evaluation of several approaches* (see Figure 1, bottom right quadrant). This was mainly empowered by the first author's practical experience and through the support of the practitioners in the case company. Still, reaching the best-performing ML solution requires a few more iterations to confirm the robustness of the selected ML model, which is marked as *future work* in Figure 1. Furthermore, we plan future steps for addressing limitations identified throughout the optimization process. Thus, we intend to answer the following research questions related to contribution **C2** in Section 1:

- **RQ3**: What capabilities of the cloud infrastructure are required to implement the proposed solution design for optimization of anomaly detection in a microservice system?
- **RQ4**: Which unsupervised deep learning method for anomaly detection performed the best in the studied context?
- **RQ5**: What are the limitations of the proposed solution and how to address them?

In Section 7 we present the proposed solution design, in-context evaluation, and results from the optimization cycles. While mentioned design steps enabled experimentation and evaluation of state-of-the-art DL approaches, the final selection of the DL approach with the highest precision in detecting anomalies will be carried out in future work.

4 Problem Context

This study is conducted in collaboration with a Swedish company, developing and operating the backend system of an application for ticket and payment management in public transportation. It is an example of a system that utilized benefits from both a cloud platform and a microservice architecture. The entire system is developed using Microsoft tools and cloud solutions, following the aforementioned DevOps principles. The health status of each of the 20 microservices is monitored using Microsoft Azure Monitor that collects data, including performance metrics and logs from various sources, such as applications and Azure resources. Collected data is mainly used for analysis, visualization, and alerting. For further analysis of anomaly detection, we consider only metrics – numerical representations of specific systems' observations taken within a predefined timestamp. Logs are disregarded as their analysis is out of scope of this study.

As stated in Section 1, it is important to take into account performance metrics of all monitored microservices and analyze dependencies over a time axis but even more importantly dependencies on other variables' observations from other microservices. Hence, we consider *application metrics* (dependency failures, exceptions, failed requests) across *14* services and *system metrics* (status codes Http 2xx, errors Http 4xx, errors Http 5xx, number of requests, response time) across *13* different services. We selected those services based on their vulnerability and significance identified during the initial study [89]. In total, we get a 120-dimensional vector consisting of 120 real-valued observations denoting specific performance metrics, which is an instance of a multivariate times series (see Appendix 1). The training data set is composed of samples recorded at a specific time during a period of 12 months. Moreover, the training data set is contaminated with approximately 1% of undetected anomalies which makes the two classes of anomalous and non-anomalous data highly imbalanced. This is an estimated contamination level based on observations from the DevOps team and the number of alerts they were receiving before we started this case study.

We have already attempted at analyzing smaller data set consisting of the aforementioned performance metrics in order to address reported challenges in managing alerts, that notify the developer team about strange and unusual system's behavior. However, creating new advanced decision rules [89] that combine metrics across different services is a prototype solution used for exploring the limits of this very complex industrial context. Therefore, we aim to explore deep learning meth-

ods, since they can learn highly complex dynamics with no assumptions about the data distributions and underlying patterns. A detailed analysis of deep learning methods applicable for the multivariate time series is presented in the following section.

5 Review of DL Methods for Anomaly Detection in MTS

We reviewed eighteen recent studies proposing deep learning methods for anomaly detection in time series. Inspired by a review by Choi et al. [27], we analyze and present a condensed overview of unsupervised DL methods across three dimensions: 1) inter-correlation between variables, 2) temporal context modeling, and 3) anomaly score criteria. The first dimension covers the various methods that may be employed for calculating the correlation between multiple variables, such as dimensional reduction, 2D matrix, or graphs [114, 125, 249]. Thus, high-dimensional monitoring data may be represented with fewer feature representations in order to reduce the dimensionality problem and the number of computing resources needed for the analysis of the raw data. The second dimension considers the temporal context of the time series and it is defined by the selection of the neural network architectures, such as Recurrent Neural Networks (RNN) [115], Long Short Term Memory (LSTM) [249], Gated Recurrent Unit (GRU) [251], or Convolutional Neural Networks (CNN) [103]. The third dimension relates to the calculation of the anomaly score that indicates the levels of anomalousness. The greater the score is, the more likely it is that the observed time series sequence is abnormal. The anomaly score can be calculated based on the reconstruction error [96], prediction error [249] or dissimilarity [103]. The mapping of DL methods for anomaly detection in multivariate time series across the aforementioned dimensions is shown in Table 1.

The first group of columns in Table 1 classifies DL methods based on the reconstruction error along the inter-correlation between variables and different neural networks for modeling temporal context. The reconstruction-based anomaly detection methods leverage the encoder-decoder architecture, namely autoencoders (AE), to reconstruct input time series and measure how much the reconstructed time series deviate from the original samples. An encoder is a fully connected neural network that encodes a time series sequence into a lower dimension to obtain the most representative features. A decoder uses the output from the encoder to reconstruct the encoded time series into the original dimensions. An autoencoder aims to reproduce the input time series with noise and anomalies removed, since the hidden layers learn to generalize the distribution of the normal data [115]. This architecture has been used as a single solution for addressing anomaly detection problems [91] as well as in combination with Generative Adversarial Networks (GANs) [64] or anomaly likelihood functions, as introduced by Ahmad et al. [3].

Table 1: Review of unsupervised deep learning methods for multivariate time series across three dimensions: 1) Anomaly criteria (reconstruction error, prediction error, dissimilarity); 2) Inter-correlation (dimensional reduction, 2D matrix, graph); 3) Modeling temporal context (LSTM, GRU, CNN, RNN)

	ANOMALY CRITERIA									
	Reconstruction error					Prediction error			Dissimilarity	
INTER-CORRELATION	LSTM	GRU	CNN	RNN	CNN+LSTM	LSTM	GRU	CNN	RNN	CNN
Dimensional reduction	[125], [26]		[249]			[249]			[190]	[138]
2D matrix	[114]		[103]		[243], [254]					
Graph		[251]					[251]			
Other	[64], [91]	[96]		[115]	[109], [242]	[44]		[159]		

Autoencoders have been widely used in various setups. Kieu et al. [115] proposed autoencoder ensembles by building a set of RNN autoencoders that may be trained in an independent framework or in a shared framework, where all autoencoders in an ensemble interact through a shared layer. In this example, the anomaly score is calculated as the median of all reconstruction errors obtained by applying the Euclidean norm to the original and reconstructed time series. This approach successfully deals with the problem of overfitting to the original time series [115]. Zhang et al. [243] and Zhao et al. [254] presented similar solutions that include constructing 2D feature matrices needed for representing the inter-correlations between the pairs of time series before applying convolutional encoders and decoders for reconstructing the input time series. Benefits of combining CNN and LSTM layers in autoencoders for learning spatial and temporal features have inspired many researchers [109, 242]. Unlike the aforementioned solutions, Chevrot et al. [26] explored using several decoders, one per mini batch. The mini batches were created by separating the original batch of data based on the discrimination feature, which simplifies the solution but provides better accuracy and timely anomaly detection.

As already stated, the encoder-decoder structure has been used in GANs solutions, mainly to model a *Generator*. Hence, the generator also aims at reconstructing time series data and generating samples that possibly could have been drawn from the original data set. In addition to the Generator, GANs consist of another sub-model, a *Discriminator*. The Discriminator learns to distinguish the real samples from the original data set and fake samples obtained from the Generator. The two GAN sub-models are trained together with the objective of minimizing the adversarial loss to match the distribution of the generated time series to the data distribution of original samples [64]. Geiger et al. [64] applied the GAN approach on normalized raw data while calculating the anomaly score as a linear combination of reconstruction errors and outputs from the Discriminator.

Differently from the aforementioned approach, Khoshnevisan et al. [114] and Wenqian et al. [103] used a 2D correlation matrix to capture inter-correlation between multiple time series and explore the most representative features before applying the GAN method. Both approaches use the same Generator structure, encoder-decoder-encoder, to optimize the input reconstruction in original and latent space. The approaches differ in modeling the temporal context and calculating the anomaly score. Khoshnevisan et al. [114] built the corresponding neural networks out of the convolutional-LSTM layer in order to learn spatial and temporal dependencies while Wenqian et al. [103] relied only on the convolutional layers. In the testing phase, Wenqian et al. [103] used both the apparent and latent loss, a L_1 distance between the real and generated time series, and Euclidean distance between latent representations of the original samples and encoded generated samples, respectively. Differently, Khoshnevisan et al. [114] determined the anomalous samples based on the reconstruction errors as a measure of differentiation between the original and reconstructed time series. Although the results of

the GAN approaches seem remarkable as in previous and similar examples [125], simultaneous training of two competing sub-models may lead to a failure mode. Thus, a very unstable training process may cause a collapsing mode in which the Generator will always output the same value for any input time series.

The second group of columns in Table 1 lists the DL methods for anomaly detection based on prediction error. These methods predict the values of the time series for the next time steps and calculate residuals between the predicted values and the actual observations. Examples of such methods were reported by Munir et al. [159] and Ding et al. [44] with a slightly different choice of neural networks, CNN and LSTM respectively, and different selection of the metrics for calculating the anomaly score. In addition to the prediction models, Zhao et al. [251] and Zhang et al. [249] consider prediction and reconstruction errors mutually in their model architectures. Zhang et al. [249] proposed a Convolutional Autoencoding Memory (CAE-M) built of a convolutional encoder whose feature representations were fed into the predictive network. Zhao et al. [251] used the feature-oriented and time-oriented graph attention layer to model the relationship between the features and time steps, respectively, before simultaneous optimization of the reconstruction and predictive model. By combining reconstruction and predictive models, the aforementioned solutions bypass the shortcomings of the individual models [251].

Dissimilarity-based methods have been less appealing to the researchers, thus, the third group of columns in Table 1 contains fewer DL methods in comparison to the other columns. However, the examples of dissimilarity-based methods have shown promising results in tackling anomaly detection in multivariate time series [138, 190]. The core idea behind these methods is to measure the distance between the value obtained by the DL model and the distribution or cluster of the original data set [27]. In particular, Liu et al. [138] proposed an architecture that consists of a feature extractor and anomaly detector. The feature extractor is a stack of CNN layers used for extracting a low-dimensional feature vector while the anomaly detector uses Mahalanobis distance to calculate how far the current observation exists from the distribution of normal and abnormal data. Similarly, Shen et al. [190] used feature extractor in their temporal hierarchical one-class (THOC) model but differently, cosine similarity to measure the distance between the features and the cluster obtained by deep support vector data description.

In the next section we discuss the selection of the minimum feasible DL method in terms of simplicity and applicability for our and similar problem contexts, while mainly reflecting on the methods and classification presented in this section.

6 Guidance for a Minimum Feasible DL Method

Most of the currently available deep learning approaches for anomaly detection in multivariate time series are context-specific and their implementation may seem

overwhelming for researchers or practitioners that are novices in the field. Moreover, there is no single solution that fits all use cases. Thus, in this section, we aim to give guidance for the selection of a minimum feasible deep learning method for anomaly detection, specifically applicable to multivariate time series. This means that the suggested method may be used as the starting point when exploring the specific context and limitations of the available data set. Further, the method may be adjusted or optimized to be used for larger and more complex data sets.

Type of anomaly. The anomaly in time series data may be represented by one or more data points that significantly deviate from previous time steps. We consider a point and subsequent anomaly [14], as two types of anomalies relevant for our problem context. The point anomaly is a data point that significantly deviates from either its neighboring data points (local anomaly) or other points in the time series (global anomaly). The subsequent anomaly refers to a set of data points whose mutual behavior diverges from the rest of the time series [14]. Both types of anomalies may affect one or more time-dependent variables in time series. To detect either of these two anomaly types, the DL methods listed in Table 1 may be applied with an additional modification for detecting subsequent anomalies since the history of detected anomalies must be maintained. Thus, the methods including sliding windows may be explored when detecting subsequent anomalies [96, 243].

Dimension of the data set. Dimensionality in time series refers to a number of attributes measured in each time step. A multivariate time series is an ordered set of n-dimensional vectors recorded at a specific time, where n denotes the number of attributes, which can be equal or greater than two [27]. As presented in Section 5, various methods for dimensional reduction, including autoencoders (AE) [249] and convolutional feature extractor [138], may be employed for extracting features based on the relationship among attributes and reducing the dimensionality of the problem context.

Temporality. A time series is a collection of data points measured and indexed in time order. The observations are recorded at equal time intervals and each data point is dependent on its prior values. Based on the review in Section 5, LSTM layers are mostly used for modeling the temporal context since they have ability of keeping the information about previous states for long periods of time. This enables learning of the long-term dependencies. Further, CNN layers were a favoured selection for extracting spatio-temporal dependencies in multivariate time series with a spatial dimension as in [243, 254], which is usually not the case with the performance monitoring data.

Selection of the DL method. For the selection of the minimum feasible methods, we exclude solutions based on GANs since they may require more development time and computing resources for their implementation but still do not guarantee satisfactory results due to their unstable training process. Thus, we rather focus on reconstruction- and prediction-based methods to be the first methods to explore when tackling the anomaly detection problems in time series. Further, we present three variations of the DL methods that could be explored as the starting

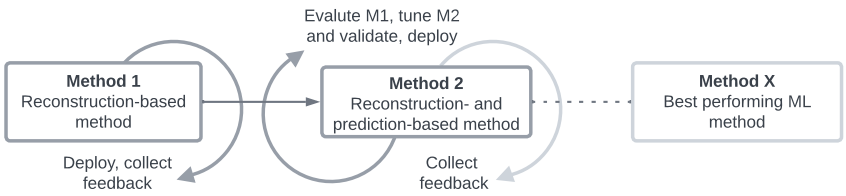


Figure 2: Overview of process for the method selection

point when addressing anomaly detection tasks.

- **Method 1:** A pure *reconstruction-based method* utilizing the encoder-decoder architecture for obtaining a reconstruction of the input data. For modeling the temporal context, LSTM layers are preferred while anomaly scores may be calculated as the Euclidean distance between original and reconstructed data points. Moreover, one could use a fixed window size to capture subsequent anomalies while keeping track of the previously detected anomalous data points as suggested by Hsieh et al. [91].
- **Method 2:** A combination of *reconstruction- and prediction-based methods* as proposed by Zhao et al. [251]. Inter-correlation may be applied in the case of high dimensionality [114] with previous data normalization. The suggestion is still to stick with the LSTM layers and use the Euclidean norm to calculate the difference between reconstructed and predicted values, and actual observations at the corresponding time steps.
- **Method 3:** A *dissimilarity-based method* with prior feature extraction using one of the approaches for discovering inter-correlations between attributes, such as LSTM autoencoders [27] or CNN feature extractor as in a solution proposed by Liu et al. [138]. The next step is to determine how far actual observed values are from the clusters or distribution of anomalous and non-anomalous samples using any clustering method, such as K-Means [181] or Gaussian Mixture Models [138].

We suggest exploring the methods for advancing the smart filter in the order they are presented. The three presented methods will be considered for advancing the smart filter, one at a time. This means that they will be deployed to the production environment one by one and evaluated. In each iteration, the development team will be assessing the precision of reported alerts. When the labeled data is obtained based on the feedback, we aim to explore the feasibility and accuracy of alternative approaches which may include second and third methods with no or minor adjustments. We continue this process until we reach the method with the highest accuracy as shown in Figure 2. In this paper, we present two iteration cycles. The details of the in-context implementation and evaluation of the proposed

solution design for anomaly detection in multivariate time series are presented in the next section.

7 Implementation and evaluation of anomaly detection approaches

In this section, we present the details of the in-context implementation of deep learning models and their deployment and integration in the feedback loop, utilizing the cloud infrastructure. This also includes a description of the model selection, data preprocessing, and the process of anomaly detection, as well as an overview of the results accomplished during the initial optimization step. Additionally, we provide a comparison of different unsupervised DL approaches in order to select the best-performing model for the next optimization cycle. We conclude the section by discussing the challenges and future improvements of the presented DL solutions.

7.1 Overview of the cloud solution for in-context implementation

As mentioned in Section 1, we aim to address a shortage of approaches for evaluating DL methods in the DevOps context where ground truth data on anomalous system's behavior is unknown. Hence, we present a cloud infrastructure used for implementation and optimization of the DL model, shown in Figure 3. When investigating examples of cloud infrastructure, we were mainly inspired by Microsoft tools and services since the system under study is implemented and deployed using this platform. A detailed explanation of each of the numbered parts in Figure 3 is given below.

1. **Model training** – It relates to the training process of the selected model within Method 1, introduced in Section 6. Depending on the size of the training data set and the model complexity, the training process can be performed either on the local machines or the cloud computing resources, such as Microsoft Azure ML Studio. The trained model is registered to the Azure Machine Learning workspace, which is needed to deploy and load the model when required.

2. **Cloud solution** – The overall solution is implemented using the Azure Logic Apps, a cloud-based platform for developing and running automated workflows. With this service, we automate fetching of monitoring data in near-real-time, predicting anomalies, sending an alert notification to the Slack channel, and saving the feedback from the development team to a shared storage. This is enabled through built-in *triggers* and *actions* that can be

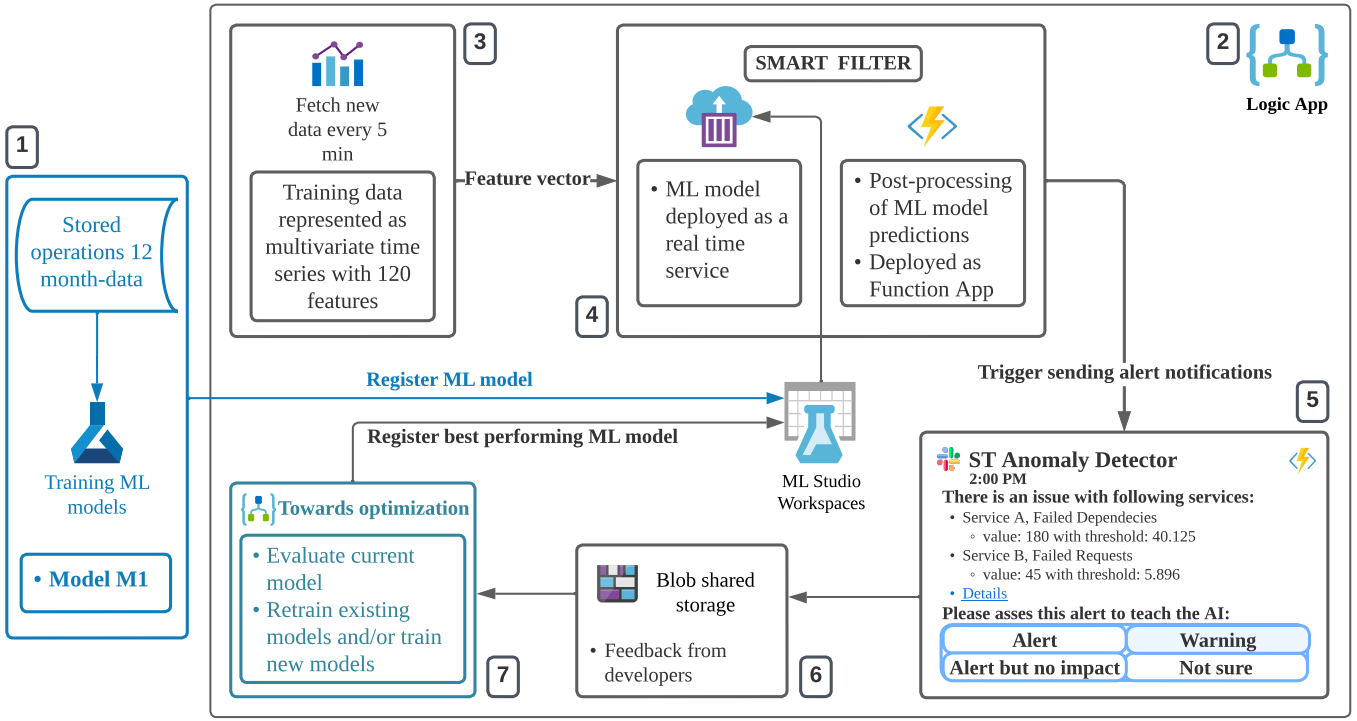


Figure 3: Overview of the proposed solution design for optimization of anomaly detection

executed sequentially or in parallel. A *trigger* represents the first step in any workflow and defines the condition for running any further steps, while *actions* are steps in a workflow and are executed after the trigger.

3. **Workflow clocking** – The automated workflow is triggered every five minutes to avoid overlap since the monitoring data is also fetched with time granularity of five minutes. This was in turn selected based on the current practice within the project. The data is collected through sequential loops iterating over performance metrics and services, thus the data is received in the same order every time. This process may take up to two or three minutes. The output of this action is a feature vector consisting of 120 performance metrics forwarded to the smart filter.

4. **Smart filter implementation** – The smart filter consists of two parts, the ML model deployed as a real-time web service to an Azure Container Instance (ACI) and a Function App where the rest of the logic is implemented. The Function App is a cloud solution for running even the smallest code snippets in the cloud on-demand, and in this case, we use it for post-processing of predictions, calculating anomaly scores, and decision making. Once deployed, the real-time web service can be accessed through its REST interface. To consume the web service, one could determine the endpoint of a deployed service and use an HTTP POST request with JSON data to call the service. The input to the service, the JSON data, contains a multidimensional feature vector, while the service should return the ML model prediction (e.g., reconstruction error in the case of the reconstruction model). Afterwards, this output is used to determine if a feature vector is non-anomalous or anomalous based on the calculated anomaly score. In case there is an observation that significantly deviates from a normal learned pattern, an alert notification with all needed details will be sent to the development team.

5. **Notifications** – Sending notifications to the development team is implemented within another Function App, which is triggered if the smart filter has previously detected an anomaly. For each reported alert during the learning period, the developer is required to provide feedback by selecting one of the available options: 1) alert; 2) alert but no impact 3) warning; 4) not sure. The feedback on the reported alerts will be continuously collected and saved in shared storage.

6. **Storage** – This is a shared storage mainly used for storing the trained DL models and feedback from developers. We utilize the benefits of the Azure Blob storage, a Microsoft object storage solution that is optimized for storing unstructured data in the cloud.

7. **Update** – In this part, we use the labeled data, generated through the feedback process, for evaluating and updating the current ML model and training

new ML models while selecting the best-performing one for the next optimization step. First iterations of optimization may be manually executed as they require extensive experimentation in order to select the model with the highest accuracy. When the desired accuracy is reached, then maintaining the DL model can be automated in a separate Logic App. For instance, the ML model may be updated every month since the Azure Monitoring platform keeps the metrics in the memory for up to one month.

The core idea of this cloud solution is to utilize feedback from the development team to evaluate and update the DL model based on generated labeled data denoting true positive alerts, true negative alerts, warnings, and unspecified alerts. The first selected method, Method 1, is evaluated after approximately two months of usage in the production environment against the feedback from the development. Further, collected labeled data is used for exploring and evaluating other DL methods, e.g., the Method 2 or Method 3 or variations thereof as suggested in Section 6. The method that gives the best accuracy is selected as the new best choice and will be deployed and periodically updated based on the feedback that is continuously collected and saved. In every iteration, more and more labeled anomalous data is collected and used to improve the unsupervised DL model and possibly learn new supervised models that will be optimized separately or in an ensemble with unsupervised DL models. When the best-performing model is selected after several iterations, further maintenance of the ML model(s) may continue in an automated workflow.

7.2 Results of the initial optimization step

Model selection

The initial optimization step includes implementing one of the state-of-the-art ML models, e.g., the ones suggested in Section 6. Thus, we selected the reconstruction-based method as one of the widely used approaches for addressing unsupervised anomaly detection problems. The neural network architecture capable of learning the identity function is shown in Figure 4. The model, namely autoencoder, contains encoder and decoder components, designed to compress the input $\mathbf{x}$ to low dimensional representation $\mathbf{z} = g_\phi(\mathbf{x})$, and to decompress it back to the original size in order to get the reconstructed input $\mathbf{x}' = f_\theta(g_\phi(\mathbf{x}))$. The parameters (θ, ϕ) are learned together during a training process to output the reconstructed data samples as close as possible to the original input, $\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x}))$. We used mean square error (MSE) as a loss function in order to quantify the difference between two vectors.

The encoder and decoder neural networks consist of a stack of hidden Long Short Term Memory (LSTM) layers, specifically designed to learn dependencies in an ordered data set such as time series. The architecture of the autoencoder model

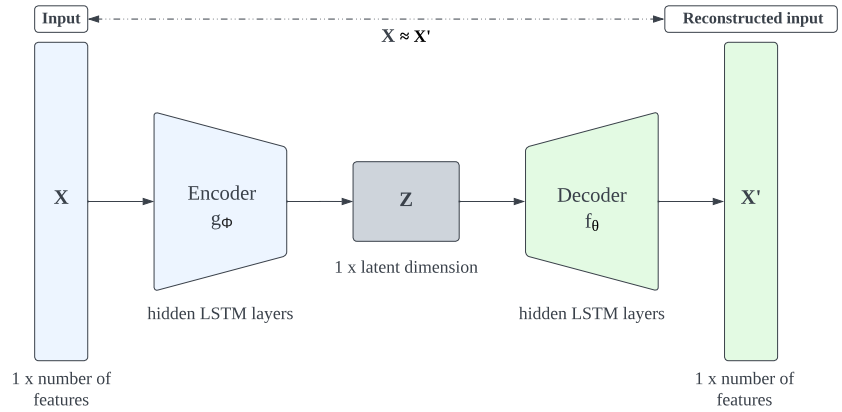
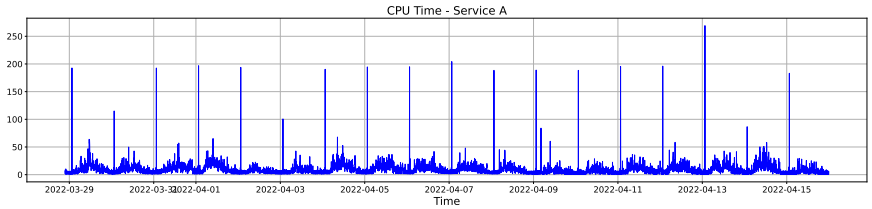


Figure 4: Illustration of the reconstruction autoencoder architecture

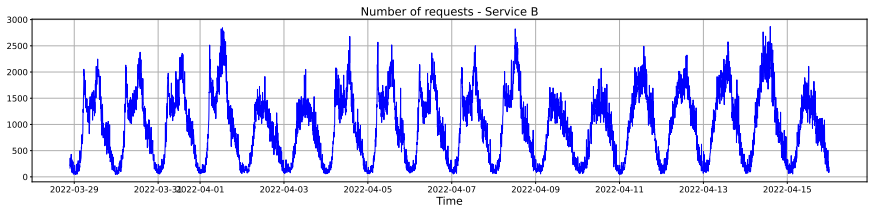
is implemented using Keras, an open-source software library with a Python interface that enables designing artificial neural networks. After varying several parameters in consecutive training runs, the lowest training loss (0.0036) is reached with input dimension 512, latent dimension 4, 5 hidden layers, `tanh` activation function, and 0.01 as a learning rate (see Appendix 2). Due to a shortage of ground truth data, the implemented model could not be evaluated before deployment. The results achieved with this model are assessed by the development team and presented in Section 7.2.

Data preprocessing

As described in Section 4, the training data set consists of observations indexed in time order and collected throughout the period of 12 months, a total of 104 256 samples. Before the training process, data is normalized to fit in a range of 0 to 1, which gives equal importance to each feature and, as a consequence, may improve model accuracy. Initial training runs were executed with a total of 120 features. It was noticeable that certain features contain seasonal variation, a change that repeats regularly over time (see Figure 5). Examples of time series shown in Figure 5 are referred to as non-stationary, and many anomaly detection algorithms are not robust to such seasonal behavior. Instead, specifically designed approaches are needed, such as Seasonal ARIMA (SARIMA) [230], where separate models are created for different seasonal time lags [18, 31]. For the system under study, the peaks and troughs are expected for certain features (CPU time or number of HTTP requests) and it is important that their existence does not degrade the performance of the ML model. Thus, in order to keep data closest to the seasonal stationary, without any seasonal components, the features with periodic fluctuations are re-



(a)



(b)

Figure 5: Examples of the features with the seasonal variation: (a) CPU Time and (b) Number of HTTP requests

moved from the training data set. Thus, the resulting training data set used in the final training run consisted of 83 features.

Anomaly detection

In order to detect anomalies, we need to define the criteria to be used for estimating if an observation is anomalous or not. We used Root Mean Square Error (RMSE) to calculate the reconstruction error, the difference between the original input and reconstructed input sample. As suggested by Islam et al. [96, 97] and Ahmad et al. [3], we keep a history of reconstruction errors on short-term (W') and long-term (W) intervals and measure how their mean and standard deviation vary over time. Thus, we calculate the likelihood of anomaly at time t as $A_t = 1 - Q(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t})$, $A_t \in (0, 1)$ [3], where Q is a Gaussian tail probability [110], $\tilde{\mu}_t$ is the mean of short-term interval, while μ_t and σ_t are the mean and standard deviation of long-term interval, retrospectively. If $A_t \geq 1 - \epsilon$, then the observation at time t is considered as an anomaly, for predefined user threshold ϵ. Since the approximate contamination in the training data is around 1%, we selected ϵ as 0.01 while W (long-term interval) is 36 and W' (short-term interval) is 3 which corresponds to 180 (36*5) min and 15 (3*5) min, retrospectively. The length of the intervals is configured based on how long the glitches and severe performance deviations persist in operations.

Results

The smart filer was implemented according to the cloud solution design presented earlier, where the autoencoder model was deployed as a real-time web service while post-processing and anomaly detection was deployed within a Function App. Whenever a new anomaly was detected, an alert notification was sent to the development team through a Slack channel. This implementation ran in operations for 50 days from 16^{th} of May to 5^{th} of July, which produced 14 400 samples in total. During that period, one of the developers was responsible for assessing the fired alerts and labeling them as alerts with high or low impact. Additionally, it was possible to label detected anomaly as a warning while there was also a category for detected deviations that are either potential false positives or don't belong to any of the three categories mentioned above.

The following describes each of the alert types, which is also an insight into how the qualitative assessment was done.

- **High impact**: The most important alert type to detect as they may affect several services simultaneously and end users. In most cases, there is an incident followed by this alert type, which indicates that end users reported an issue to customer service. When there are severe issues rolling out, multiple alerts of this type are reported.
- **Low impact**: This type may precede the alerts with high impact and signalize that temporary deviations in performance metrics may escalate into severe issues if not timely addressed. Low impact implies that only a few services are affected at the same time and they might self-heal but still caution is needed.
- **Warning**: A temporary glitch in the performance metrics or a few errors causing the latency of the affected services. In general, these issues should not be noticeable to any end user. They might however be worth tracking retrospectively to understand trends.
- **Unspecified**: The type of alerts that might represent false positives. However, they still signalize some kind of disturbance in a system even though any of the single services might not be affected. That is a reason for their troublesome classification under the three categories described above.

The number of detected alerts per category is presented in Table 2. In total, there were only 3 false negatives alerts, on one particular day, missed by the smart filter. It was one isolated event that was supposed to be detected with three consecutive alerts. However, the change caused in the performance metrics did not significantly affect the mean and standard deviation of the reconstruction error distribution. False negatives or reported alerts that falsely indicate that there is an anomalous observation, are among unspecified alerts. Considering the fact that unsupervised models can be retrained only in an unsupervised way, thus without

Table 2: Results of the initial optimization step based on the feedback from the development team (*false positives* are hidden within *unspecified* category)

Type of alert	High impact	Low impact	Warning	Unspecified
Number of true positives	7	29	59	27
Number of false negatives	3	0	0	0

labels, the presented autoencoder may be complemented with a supervised model to improve its precision. Building a classification model, using the generated labeled data, might improve the sensitivity of the smart filter to the high-priority alerts. Further improvements and challenges are presented in subsection 7.4.

The distribution of reported alerts is shown in Figure 6. Alerts with low importance occurred as isolated events during the observed period. Even though their criticality is not as high, it may evolve very quickly. There is a significantly large number of unspecified alerts that were reported as single events or, before or after warnings. Those are mostly alerts that were reported based on detected perturbations in the system but only a few or no data points in the single services were detected as anomalous. Since some of them also might be false positives, we set the second goal for the next optimization cycle: *to reduce the number of unspecified alerts.*

Even though the initial ML reconstruction-based model is already able to find most or at least the most serious alerts, we want to explore more variations of the same model and other ML techniques to check if higher precision can be reached. That we explore in the next optimization cycle.

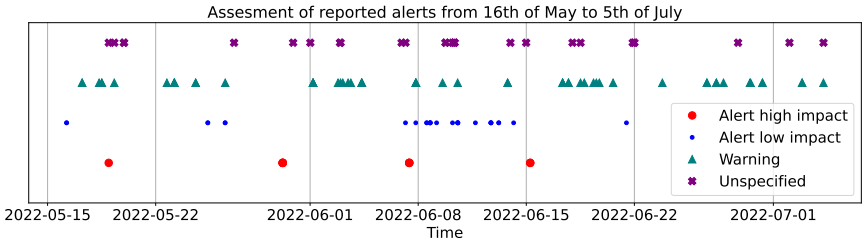


Figure 6: Distribution of assessed alerts from 16^{th} of May to 5^{th} of July 2022.

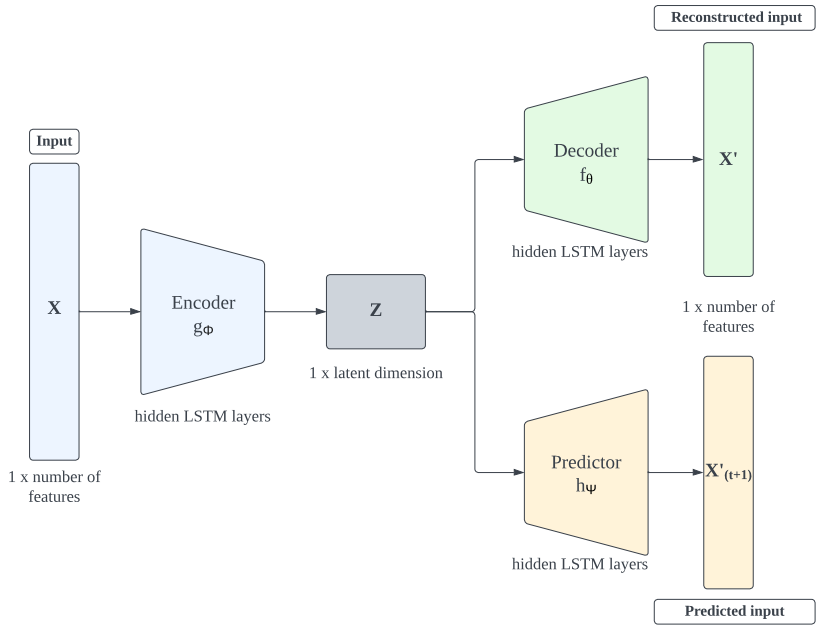


Figure 7: Illustration of the composite autoencoder architecture

7.3 Towards next optimization cycle: Evaluation and comparison of different unsupervised DL approaches

The purpose of the next optimization cycle is to evaluate different unsupervised ML approaches before selecting the one to be used in the final deployment. As proposed in Section 6, we further explore the combination of reconstruction- and prediction-based models. The architecture of such a model is shown in Figure 7. It is a composite LSTM autoencoder with a single encoder and two decoders with different roles, one is used for reconstruction while the other one is used for prediction. In addition to the reconstructed input, $\mathbf{x}' = f_\theta(g_\phi(\mathbf{x}))$, there is one more output from the predictor representing the prediction of the input in the next time step, $\mathbf{x(t+1)}' = h_\psi(g_\phi(\mathbf{x}))$. Similarly to the original autoencoder, all the parameters (θ, ϕ, ψ) are learned together during the training process. In this case, we have two types of errors, calculated as Root Mean Square Error (RMSE) between original input, and reconstructed or predicted input. The total error is a linear combination of these errors, $e_{total} = \lambda e_{rec} + (1 - \lambda)e_{pred}$, where $\lambda \in (0, 1)$ and it gives equal, higher, or lower priority to either of them. Regarding the data preprocessing and calculating the anomaly likelihood score, we followed the same procedure as for the initial optimization step.

Alongside this model, we evaluated the autoencoder designed in the initial

optimization step with different setups of parameters. Thus, for the two models, we altered the type of neural network, activation function, batch size, input, and latent dimension to observe their performance. Selected results of the multiple executed runs are presented in Table 3. Furthermore, for the composite autoencoder, the effect of different λ values was examined to see if either reconstruction or prediction error has a higher impact on the accuracy of anomaly detection. For each new set of parameters, we measured the total number of prospective alerts and how many of each alert type (high impact, low impact, warning, or unspecified) would be detected. Average accuracy of the model is obtained as $AverageAcc = \frac{1}{3}(\frac{Corr_{high}}{Total_{high}} + \frac{Corr_{low}}{Total_{low}} + \frac{Corr_{warning}}{Total_{warning}})$, where each of the summands represents the ratio of correct predictions to the total number of samples in a single alert category. We disregarded unspecified alerts for calculating the accuracy as this is not the preferred category to be detected. The goal was just to reduce the number of unspecified alerts in each iteration as the anomaly detector improves over time.

The overall results showed that autoencoder architecture is quite robust to any parameter modification and that for both types, the training loss converges to 0.0038 and 0.0072 for original autoencoder and composite autoencoder architectures, retrospectively. The parameters considered in Table 3 are chosen to reflect the main characteristics of the neural network layers and learning process. The selected combinations of values were determined according to the initial setup (see Section 7.2). Hence, we selected the next higher or lower values of parameters or different type of neural network layers or activation functions that are commonly used for such solutions [8,135,141]. We did not consider all possible combinations since we obtained very consistent results after a few executed runs.

A slight difference may be noticed in the total number of alerts while there is almost no change for the different types of alerts. Unclassified alerts are the ones that did not fall under any of the categories. However there were only a few of these alerts and the reconstruction- and prediction-based model demonstrated slightly higher precision by detecting almost all alerts within the defined categories. Average accuracy did not drastically change either, being around 0.9 for different combinations of parameters. In most of the training runs, the learning rate was set to 0.01, while lower or greater values just affected how fast the minimum of the loss function was reached. In all the cases, also regardless of the number of training examples in one iteration (batch size) or which function is used for activating neighboring neurons (activation function), the loss always converged around the same values.

One specific fact is that in all observed cases, there was a greater number of low-impact alerts than expected. The reason is that some of the alerts were detected five or ten minutes earlier before an actual alerting event, which is a desired characteristic since we want to detect any performance disturbance as earliest. The aforementioned parameter λ, which may prioritize either prediction or reconstruction error, did not significantly change the final results.

Table 3: Selected evaluation results of different ML approaches for anomaly detection

	Different variations of ML models					
	Reconstruction-based models			**Reconstruction- and prediction-based models**		
Alert types	LSTM layers (activation=selu, input dim=1024, latent dim=8, batch size=128)	GRU layers (activation=relu, input dim=512, latent dim=8, batch size=64)	GRU layers (activation=sigmoid, input dim=1024, latent dim=8, batch size=256)	GRU layers (activation=sigmoid, input dim=512, latent dim=4, batch size=128, $\lambda = 0.8$)	LSTM layers (activation=relu, input dim=1024, latent dim=8, batch size=256, $\lambda = 0.1$)	LSTM layers (activation=tanh, input dim=512, latent dim=8, batch size=256, $\lambda = 0.5$)
High impact (expected 10)	8	8	8	8	8	8
Low impact (expected 29)	29 (+6)	29 (+6)	29 (+6)	29 (+4)	29 (+4)	29 (+4)
Warning (expected 59)	53	55	55	54	55	54
Unspecified (expected 27)	22	22	23	23	22	23
Average accuracy	**0.9**	**0.91**	**0.91**	**0.9**	**0.91**	**0.9**
Unclassified alerts	5	6	2	0	2	1
Total	**123**	**126**	**123**	**118**	**120**	**119**

Thus, using only reconstruction or prediction components would lead to the same outcome. If we consider the worst case scenario, where alerts with high and low impact and warnings are *true positives* and unspecified alerts *false negatives*, the precision and recall are (0.73, 0.96) and (0.76, 0.97) for original and composite autoencoder, retrospectively. However, due to the uncertainty of unspecified alerts, the initial goal was to reduce their quantity, which was partially achieved in this iteration.

This optimization cycle continues by selecting the new model for deployment, which will be used in operations for a certain period while also being evaluated by the development team. We select the reconstruction- and prediction-based model for deployment, differently than in the initial optimization step.

Even though this evaluation indicates no or little difference, we want to evaluate the model that combines two types of errors. After the new evaluation phase, we will then decide which model we want to use as the final option.

7.4 Further improvements and challenges

Throughout the optimization cycles, we focused on the selection of the best-performing unsupervised approach. However, as already stated, unsupervised models can be retrained only with unlabeled data. Generated labeled data through feedback from development may only be used for evaluation of unsupervised and training new *supervised* models, such as classification models. If we want to further increase the precision of the anomaly detector, we may do it by building a new supervised ML model, that will complement the existing unsupervised model and increase its sensitivity to high-impact alerts.

There are plenty of examples of how such a classifier could be implemented [143, 204]. Furthermore, we may look for classical machine learning or deep learning approaches for multi-classification as we have four categories (three types of alerts and normal samples) where samples are not necessarily indexed in time order. For instance, the AdaBoost classifier is an ensemble learning method that combines numerous and usually weak classifiers and turns them into strong ones through sequential learning [143]. Other examples are K-Nearest neighbours [158], which uses proximity to make classifications, CNN-based transfer learning using pre-trained models [48], or another ensemble method, random forest classifier [108]. With multiple applications in various fields [108, 143, 158, 204], these types of classifiers could also fit in our context.

In a few trial runs, we evaluated some of the aforementioned classifiers using 50-days of labeled data but encountered the challenge of imbalanced classes. This means that our training data set consists of one majority class of normal samples (99% of data) and three minority classes denoting alerts (only 1% of data). State-of-the-art ML approaches with default setups might not be sufficient to address such skew data. Instead, more advanced and custom approaches are needed and they are about to be explored in future work. Possible directions are under-

sampling, reducing the size of the abundant class, or over-sampling for increasing the size of rare samples by generating synthetic data [239]. However, more thorough research is needed as both suggestions may lead again to conflicting results.

7.5 Threats to validity

In order to increase the validity in general, we surveyed state-of-the-art and widely adopted machine learning techniques, applicable to our industrial context. Additionally, we closely collaborated with our industry partner (interviews and regular discussions) to identify relevant needs and evaluate implemented solution. Potential validity threats are discussed in terms of conclusion, internal, and external validity.

Conclusion validity. Drawn conclusions about the effectiveness of the implemented solutions might be affected by the duration of the evaluation period (50 days) and the fact that only one practitioner was available for assessing reported alerts. Time and resource constraints are quite common and sometimes inevitable for academia–industry collaborations. To overcome this, our solution for anomaly detection in operations will be used by the DevOps team even after study completion, while we also plan for future reflections with practitioners after much longer usage.

Internal validity. Selection of the features described in Section 4 might be biased, which could also affect the overall results. They were selected based on the very detailed observations of the practices within the team specifically focusing on the services and metrics that were usually examined in case of severe failures. Further, we also tended to select some typical measures of software health and performance according to DevOps experts and insights from the literature.

External validity. Our implementation and evaluation are currently dependent on the studied industrial context, which restricts external validity and the scope of validity of our technological rule presented in Section 1. However, our ML solutions are based on state-of-the-art techniques that are evaluated in other contexts. With this, we assume that the results are generalizable and that the same or similar solutions could be used in different contexts, which will be further examined in future work.

8 Conclusion

In this paper, we address how continuous monitoring in DevOps contributes to revealing unexpected and unwanted failures. We specifically focus on distributed software systems, such as microservices, as their complexity additionally increases vulnerability in operations.

Our first contribution (**C0**) is to conceptualize problems related to the continuous monitoring practice. In addition to our previous observation of the *alert*

flooding problem [89], we now conclude that it is essential to analyze monitoring data across multiple services, namely multivariate data, to capture the overall health of the operating software.

Searching for a solution to this problem, we present our next contribution (**C1**) which is an overview of unsupervised deep learning approaches for anomaly detection in order to identify potential discrepancies in multivariate monitoring data (**RQ1**). Furthermore, we provide generic guidelines for the selection of the three minimum feasible methods for anomaly detection in time series, based on the review in Section 5 (**RQ2**).

Our last contribution (**C2**) is the deployment in a real-world DevOps context, using the Microsoft Azure cloud platform. Thereby we identified the needed capabilities of the cloud infrastructure (**RQ3**) which were met by the standard platform.

In this setting, we evaluated the DL models. The labels were generated through the same infrastructure, based on the feedback from development. Moreover, we investigated if other versions of DL methods, suggested in Section 6, may perform better on the data set used for evaluation. Both versions of the autoencoder and their variations showed stable and constant accuracy for different parameters (**RQ4**). However, we suggested using a composite autoencoder in the next optimization cycle as the prediction component might enable earlier detection of anomalies.

Still, there are opportunities for improvement, which include training new supervised models based on generated labels that may assist in revealing the category of the alerts and reducing the number of false positives. To proceed with this improvement, we first need to address the challenge of imbalanced classes (**RQ5**). Section 7 provides details about possible improvements.

This study refines and adds validity to the technological rule of *alert flooding as an optimization problem*, identified in our initial work [89]. Specifically, we have demonstrated that the additional element in the feedback loop is an efficient concept for discovering early signs of production failures. With the smart filter, multivariate monitoring data keeping the information about the health of operating software can be processed in real-time and declared as anomalous or non-anomalous. Thus, we have strengthened the grounding of our overall technological rule (**TR**) through the field test in our case study.

We used state-of-the-art deep learning algorithms to implement the smart filter and showed that such solutions can be used for addressing challenges in real industrial DevOps contexts such as alert flooding. Moreover, presented cloud solution (Section 7.1) can be used for generating labels to address one of the biggest struggles in the ML community, the shortage of labeled or ground truth data [15].

Overall, with timely anomaly detection through operation monitoring in DevOps, severe failures in operations might be prevented. The presented results are currently dependent on the implementation in this case context, but we aim to broaden our research by evaluating our proposed solution design in different contexts to demonstrate the generalizability of the solution. This is planned for future

work in addition to the new optimization cycle for obtaining the best-performing model.

Acknowledgments

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. We thank the DevOps teams for their willingness to share insights and respond to our questions.

Appendix 1: Training data

This supplementary material provides an example of a feature vector (an instance of training data) and can be found online at `https://ars.els-cdn.com/content/image/1-s2.0-S0950584923000952-mmc1.xlsx`.

Appendix 2: Jupyter Notebook

This supplementary material provides Jupyter Notebook with python code for loading data, preprocessing data, and training ML models used in this paper. It can be found online at `https://ars.els-cdn.com/content/image/1-s2.0-S0950584923000952-mmc2.zip`.

Appendix 3: Poster for practitioners

This supplementary material provides a summary of the contributions from this paper in a form more adapted for practitioners. It can be found online at `https://ars.els-cdn.com/content/image/1-s2.0-S0950584923000952-mmc3.pdf`.

PAPER III

AUTONOMOUS MONITORS FOR DETECTING FAILURES EARLY AND REPORTING INTERPRETABLE ALERTS IN CLOUD OPERATIONS

Adha Hrusto, Per Runeson, Magnus C Ohlsson

Abstract

Detecting failures early in cloud-based software systems is highly significant as it can reduce operational costs, enhance service reliability, and improve user experience. Many existing approaches include anomaly detection in metrics or a blend of metric and log features. However, such approaches tend to be very complex and hardly explainable, and consequently non-trivial for implementation and evaluation in industrial contexts. In collaboration with a case company and their cloud-based system in the domain of PIM (Product Information Management), we propose and implement autonomous monitors for proactive monitoring across multiple services of distributed software architecture, fused with anomaly detection in performance metrics and log analysis using GPT-3. We demonstrated that operations engineers tend to be more efficient by having access to interpretable alert notifications based on detected anomalies that contain information about implications and potential root causes. Additionally, proposed autonomous monitors turned out to be beneficial for the timely identification and revision of potential issues before they propagate and cause severe consequences.

ICSE-SEIP (2024)

1 Introduction

Maintaining cloud-based system operations free from failures is increasingly challenging, considering the complexity of software architectures and the wide variety of cloud deployment options used for developing and operating modern software systems. Streamlining the software development cycles by applying agile and DevOps practices partially addresses this ever-rising concern. Since modern applications rely heavily on underlying cloud infrastructures, managing and monitoring corresponding cloud resources has become essential in providing resilient large-scale software systems. Thus, cloud operations (CloudOps) [7] is another critical component of digital transformation together with DevOps that helps provide more reliable software systems. Reliability can be achieved with run-time monitoring of key performance indicators (KPI) and timely detection of deviations that signalize either a software bug or performance degradation of the cloud platform.

One of the widely used approaches for detecting abnormalities in monitoring data is *anomaly detection* [33]. Early detectability of observations that significantly deviate from the normal pattern might be crucial for preventing the roll-out of severe software failures. Further, an anomaly detection service that provides early warning of impending failure can reduce the costs associated with reliability and downtime [195]. To tackle this, many software development companies nowadays rely on commercial monitoring tools [205] for tracking and visualizing monitoring data, while reporting anomalies as *alerts* via email or communication platforms (e.g., MS Teams or Slack). This is often implemented with manually set alert rules or anomaly detectors provided by the monitoring tool. The approach, however, risks causing information overload for operators due to the diversity and abundance of metric indicators and detection mechanisms that can usually be configured only for single metrics (univariate data). Additionally, generated alert notifications are often not intuitive as they lack clarity and actionable information, making it difficult for operators to understand the underlying issues. This characteristic, which we refer to as *interpretability*, may be enhanced by providing more clear and concise insights on how to approach raised alerts.

Our recent paper [88] partially deals with this concern, where we proposed and implemented an approach for optimization of anomaly detection in multivariate data. The solution based on *LSTM autoencoders* is validated in the context of a microservice-based system. In this paper, we extend the scope of validity by collaborating with a new *case company* and implementing the solution in a new industrial context. The case company is responsible for developing and operating a Product Information Management (PIM) system using Microsoft Azure tools and services. The new system under study has a different software architecture but is distributed in nature and is cloud-based. Moreover, inspired by the latest advancements in the machine learning area, we enhance our solution with an additional deep learning (DL) model based on transformers [214] to be able to capture

broader temporal trends in data.

The two DL models, which we refer to as *LSTMAE* and *TRANAE*, are the main part of the *autonomous monitors*, i.e., the cloud solution used for near-real-time monitoring and detecting deviations in metrics, numerical observations of specific aspects of Azure resources. In addition to metrics, logs can provide significant insights into various activities and events in operations. However, the large amount of available logs is not usually favorable for in-depth analysis, considering their complex structure. Further, existing approaches are not applicable to different contexts due to a lack of standardized logging procedures [17]. Instead of combining metrics and log features together for anomaly detection [124, 175], we stick with only metrics analysis while leveraging the pre-trained large language model GPT-3 [162] to support the interpretation of logs and to generate interpretable alerts.

The solution approach was inspired by the practices of the CloudOps teams and the way they were handling and resolving alerts. Detecting deviations in metrics was the first line of defense and then digging deeper into the logs was the next step to find clues for root cause analysis. We designed the autonomous monitors to mimic this behavior by using two DL models to simultaneously detect discrepancies in metrics and the GPT-3 model [162] to give suggestions for root cause analysis, which is crucial for preventing the roll-out of severe failures. The output of the autonomous monitors is an interpretable alert notification, providing a detailed status report, to which we refer as a *smart alert*.

Contributions of the paper are threefold:

- **C1:** A novel approach for monitoring metrics and reporting *interpretable* alerts based on two state-of-the-art anomaly detection DL techniques and the GPT-3 language model, which is evaluated in a real-world setting.
- **C2:** Implementation and evaluation of a deep transformer network [214] in an industrial context, which was previously evaluated only on publicly available datasets.
- **C3:** Case-based generalization of two case studies resulted in assessing opportunities for *smart alerts* in multiple contexts.

2 Background and related work

Commercial cloud providers gained tremendous popularity in the last decade, especially among software development companies that seek scalable and flexible platforms for the deployment of their software. This means that such cloud resources can be dynamically allocated to accommodate varying workloads, providing enhanced performance when needed. However, cloud infrastructures are not immune to failures due to increasing computing demands, growing complexity of system architectures, and cost-effective but low-quality commodity hardware

used in cloud data centers [149]. Consequently, software systems relying on these cloud resources are also at risk of underperforming in operations [212]. Run-time failures in cloud-based software systems can be identified on the *application level* (e.g., service unavailability or slow response times), *platform level* (e.g., errors in deployment configuration settings), or *infrastructure level* (e.g., physical servers within a data center fail) [205]. This multilayer failure source makes it even more challenging to timely identify and resolve any potential threat to performance and reliability degradation [33].

Continuous monitoring is thus of high importance for tracking the health status of each system component in the cloud. Tamburri et al. [205] conducted a survey in over 70 different organizations, searching for monitoring practices in the industry. The key findings revealed that software development organizations are not fully aware of the potential benefits of monitoring data analysis, while 90% of respondents identified that one of the main challenges is the lack of standardization in the realm of monitoring tools. This implies that many organizations use a minimum of two or more different monitoring tools as they struggle to find a unified tool to fulfill all their needs regarding collecting, transferring, processing, storing, and visualizing monitoring data [205]. The same study reports that most adopted monitoring tools include Google Analytics, Nagios, Grafana, Amazon CloudWatch [201], Microsoft Azure Monitor [88], and ELK (Elastic, Logstash, Kibana) [105]. Many of these tools include some alert mechanism for current or forecasted observations with predefined thresholds [217].

Existing approaches for detecting failures based on monitoring data are mainly inspired either by anomaly- or signature-based strategies [149]. Signature-based approaches identify behavior associated with specific faults, aiming to detect similar faults during runtime, while anomaly-based approaches identify normal system behavior and aim to detect deviations from these norms. Since faulty behavior is unknown and hard to reproduce in many industrial contexts, anomaly detection approaches gained wider acceptance and usage. Thus, in this paper, we further focus only on anomaly detection techniques, designed for unlabeled data and based on machine learning.

The metrics, as an essential part of monitoring data, are usually represented as a time series, a sequence of data points recorded at specific time intervals. For this type of data, many anomaly detection techniques were implemented and evaluated in the context of cloud computing systems [96, 99, 139, 184]. However, as time series are widely used to model data in various domains (e.g., finance or healthcare), a number of approaches for anomaly detection were also proposed and evaluated on publicly available datasets or benchmarks. Some of the approaches that gained wide popularity include autoencoders [148], deep convolutional neural network (CNN) [159], GANs [125], and lately very attractive transformers-based networks [214]. In practice, the choice of which technique to use depends on the specific characteristics of the data and anomaly detection tasks. Often, a combination of these methods or hybrid models can yield better results. Logs, as the other

significant part of monitoring data, were also of interest for analysis and detecting anomalies, standalone or in combination with metrics [124, 175].

There is still a significant number of anomaly detection approaches that were not evaluated in industrial contexts of cloud-based systems. Thus, we take that opportunity to deploy *TRANAE* in operations and integrate it into the monitoring platform of the system under study. Additionally, we implement a custom monitor as we want to perform inference periodically and report interpretable alerts, which provide more valuable information than just a binary value, usually provided by inbuilt anomaly detectors within a monitoring platform [217].

3 Research approach

Similar to our previous work [88], we use the design science paradigm [182] to frame our research at the case company as it aims to *improve an area of practice*. The main contributions of such research are *technological rules*, i.e., prescriptive recommendations for practice. Since they are context dependent, our goal in this study is to extend the validity of already validated prescription for practice, which maps the challenge of *alert management* to the solution for *anomaly detection in multivariate data* [88]. By alert management, we refer to any activity related to reporting, interpreting, or resolving alerts. Insufficient engagement in regard to these activities will eventually cause *alert flooding* and burden operations teams. To avoid this undesirable consequence, we explore possibilities of improving alert management in a new industrial context while also considering the latest scientific contributions in the area of ML to advance the solution design for more efficient monitoring.

To reach our goals, we conducted four main activities of the design science cycle [182]:

- *Problem conceptualization* — By closely collaborating with the DevOps and CloudOps teams, which included interviews and observations of their practices, we managed to confirm that alert management and consequently alert flooding was one of the main issues in operations. Thus, we continued with a more detailed analysis of monitoring data used for designing the alert mechanism. Surveying literature and previous experience in similar studies enabled the formulation of problem constructs. Metrics and logs representing multivariate time series and a chronological record of events from operations were identified as critical data for which we needed to envision a matching solution.

- *Solution design* — We design a solution for monitoring and reporting interpretable alerts by replicating the current practices of the team and creating an automated workflow that consists of *monitoring*, *detection*, and *interpretation*. The monitoring part includes pulling the data from different

Azure resources periodically and creating a feature vector, formatted for use with DL models. Following the latest advancements in the machine learning world, we use deep *transformer network* [214] for anomaly detection, simultaneously with *LSTM autoencoders*, which was the only DL model in our previous work [88]. Since logs were used by the CloudOps teams in later stages to investigate detected anomalies, similarly, we employ the GPT-3 model [162] to analyze and interpret the logs whenever an anomaly is detected. These procedures of monitoring, detection, and interpretation of alerts are integrated into the monitoring system, which we refer to as the *autonomous monitors*.

- *Instantiation* — The proposed solution design of the autonomous monitors is implemented using Microsoft Azure services since the system under study is also developed and operated using the same cloud platform. We leveraged solutions for registering and deploying ML models, building and deploying event-driven code, and automating workflows.

- *Empirical validation* — For validation of implemented solutions, we used feedback from DevOps and CloudOps teams on reported alert notifications in the MS Teams channel. We draw conclusions by analyzing qualitative data collected in a survey using questionnaires and quantitative data representing the performance of the ML models for anomaly detection.

4 Problem context

This study is conducted in collaboration with the *case company* in Sweden, which specializes in developing and operating Product Information Management (PIM) software solutions. The architecture of the PIM system is highly distributed and built of multiple macro services. The functionality of the PIM software is used by businesses to manage and centralize product information, such as descriptions, specifications, pricing, and imagery, to ensure consistency and accuracy across various sales channels, such as e-commerce websites. It enables businesses to streamline their product data management processes and ensure that the right product details are available to customers. The system is continuously monitored using Microsoft Azure Monitoring service, where all monitoring data goes through either Application Insights (traces, exceptions, log messages, different performance indicators) or Log Analytics Workspace (database metrics, performance counters). Additionally, the most important monitoring data is visualized using Grafana, where simple alert rules are configured using manually set thresholds to detect and report anomalies in specific performance counters. Detected alerts are sent either via email or the communication platform, MS Teams.

At the beginning of our collaboration with the case company, the current practices within the operations teams were still insufficient to timely identify alerts

with the highest priority. Severe failures were mainly identified by customers when they had already experienced slowness or outage of some services. This is an undesired consequence that we mutually agreed to address, by exploring and identifying key performance indicators (KPIs) that require more efficient monitoring. In this step, which we refer to as *problem conceptualization* in the research approach, the following types of monitoring data are identified as crucial: (1) DTU (database transaction unit) and storage usage of elastic pools, containers that hold multiple databases; (2) CPU percentage, available memory and number of threads for three types of Service Fabric nodes (individual virtual machines responsible for hosting, running, and managing the services); (3) selection of performance metrics for frontend parts of the system deployed using App Services and queued long-running jobs; (4) selection of standard metrics available in Application Insights, including different indicators of service performance. The aforementioned data types are used for designing the solution for proactive monitoring across different parts of the system, anomaly detection, and creating more comprehensive and actionable alert reports.

5 Autonomous monitors

The main idea of proactive monitoring is to motivate operations teams to inspect and resolve alerts before they cause a chain reaction of undesired events within a monitored system. For this purpose, more actionable alerts are needed with as many details as possible regarding detected anomalies. In this section, we present the *solution design* (as introduced in Section 3) of the autonomous monitors for detecting and reporting *smart alerts*, that enables proactive monitoring.

5.1 Implementation

The main goal of the autonomous monitors shown in Figure 1 is to retrieve a preselected set of metrics periodically and utilize ML models for anomaly detection to detect deviations that significantly deviate from the normal pattern. The most critical metrics are identified with the Peak Over Threshold (POT) method [192], used for learning threshold values for anomaly scores. In case of an anomaly, the next step in the workflow is pulling logs and predicting possible root causes with a GPT-3 language model. The final output of the autonomous monitors is an interpretable alert notification, shown in Figure 2. We refer to this step as *instantiation*, as described in Section 3. The detailed description of each step in the workflow shown in Figure 1 is given below.

1. **Automated workflow** — The overall solution design of the proposed autonomous monitors is implemented using a Logic App in Azure, a cloud-based service for creating, scheduling, and automating workflows. With its

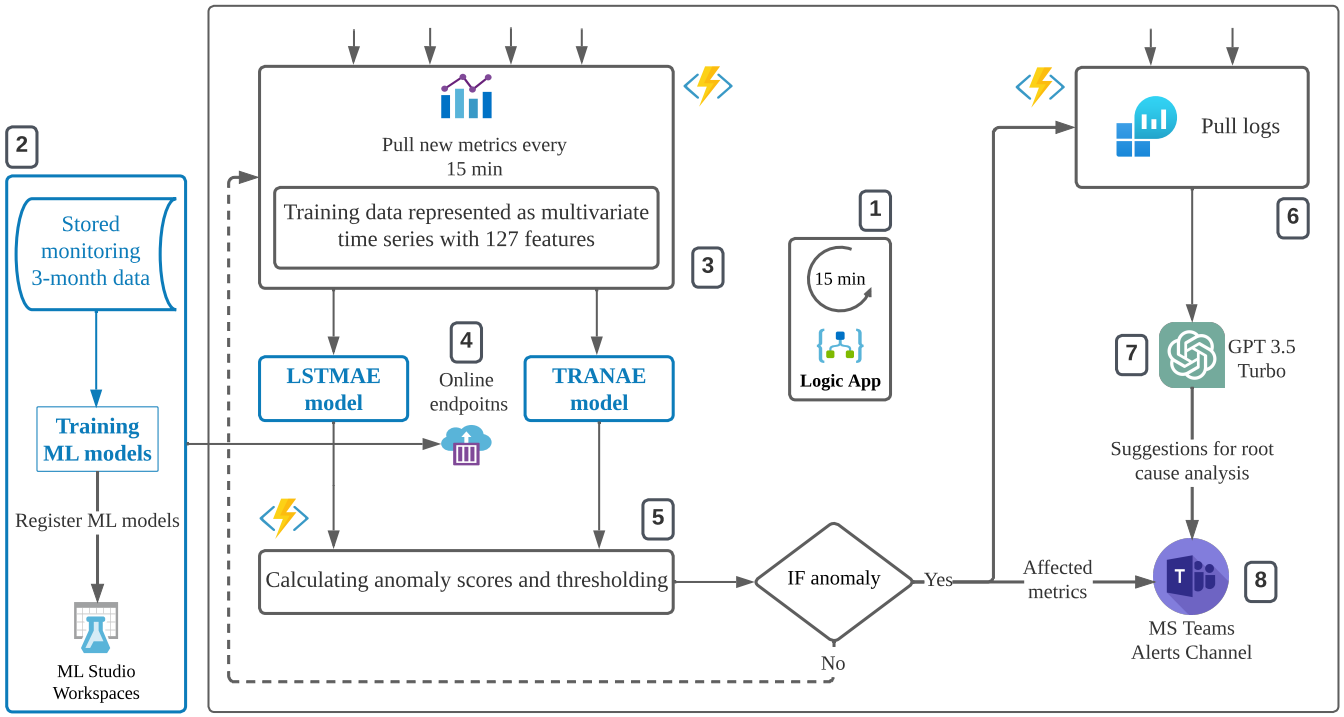


Figure 1: An overview of the autonomous monitors for detecting and reporting *smart alerts*

inbuilt *triggers* and *actions*, it is possible to define the condition or frequency of the execution and implement and connect each step.

2. **Training ML models** — We used 3-month data of metrics for training both *LSTMAE* and *TRANAE* models. A more detailed overview of data and preprocessing is explained in Section 5.1. Considering the size of this training dataset, we decided to perform training on our local machine. However, in the case of larger datasets and more complex ML models, the training can be performed on some cloud computing resources. The models were developed in Python using *Pytorch* library [168]. *LSTMAE* was developed by authors while *TRANAE* is an open source solution developed by Tuli et al. [214]. After the training, the models were registered in Azure ML Studio Workspace, for easier managing and deploying.

3. **Pulling metrics** — In agreement with the team in the case company, we configured the frequency of the automated workflow to be fifteen minutes. The first triggered step is the execution of the four Function Apps, responsible for fetching selected metrics. Azure Functions are lightweight cloud solutions for building and deploying code written in various programming languages. Thus, we developed and deployed Python scripts for accessing metrics via Azure Monitor Metrics and Query APIs and pulling all required data from Application Insights and Log Analytics.

4. **Deploying ML models** — Both models, *LSTMAE* and *TRANAE* are deployed as online endpoints in Azure ML Workspace. In this way, the models are accessible for real-time inference via REST APIs and can be seamlessly integrated into production workflows. We consumed deployed endpoints by sending the HTTP POST request to the endpoint's URL and providing the necessary input data. The request body included the feature vector contacting metrics, obtained in the previous step. Since both models are reconstruction-based, the output of each model is a reconstruction of the input data. To calculate the difference between the original and reconstructed input sample, the reconstruction loss, we used Mean Square Error (MSE) as a common metric in machine learning. Thus, the response to the HTTP request contains reconstruction losses from two ML models as a measure of anomalousness.

5. **Thresholding** — We used different approaches for each model to determine if reconstructed losses are anomalous. For the *LSTMAE* model we calculate anomaly likelihood as suggested by Islam et al. [96] and Ahmad et al. [3]. For the *TRANAE* model, we use the statistical method, Peak Over Threshold (POT), as it was originally proposed by Tuli et al. [214]. This method is used in extreme value analysis, and it's particularly valuable for detecting rare or extreme values, that are unusually high or low.

6. **Pulling logs** — Similarly as metrics, the logs are also pulled from the same sources using Azure Functions. This step is triggered only when an anomaly is identified in order to control the costs associated with the number of executions of the entire workflow (Logic App). Additionally, we agreed that analysis of logs is not necessary in the normal state as the logs will most probably contain only warnings and not errors.

7. **GPT-3 predictions** — We leveraged the latest advancements in the ML world by using the pre-trained language model GPT-3 [162] (GPT-3.5 Turbo) for the logs analysis. The main goal was to discover some implications of the identified anomaly in metrics and predict the clues for root cause analysis. The model was deployed within the Azure AI Studio. The input to the model was structured as a series of messages formatted for API communication. The first message contained the 15-minute history of log data, while the second message included a query asking the model to suggest the top two debugging strategies based on the logs, specifically highlighting the affected applications.

8. **Alerts in MS Teams** — The final step of the autonomous monitors is sending an alert notification to the MS Teams channel. An example of *such smart alert notification* is shown in Figure 2 where some sensitive information related to the case company is hidden and replaced with some arbitrary values. However, the main design of alert notification is unchanged and presents a real example of an alert sent to the operations team.

Data

As described in Section 4, the training set consists of metrics that we sorted into four categories. The total number of features considering all categories is 127, where a detailed overview of each category is presented in Table 1. The training data set consists of 8640 samples recorded every fifteen minutes within three months. Some of the features contained null entries, thus we replaced them with zeros to maintain data structures and avoid bias. The next necessary step before the training process is to equalize the scales of features to prevent domination of one of them during the learning process. For the *LSTMAE* model, we used *Scikit-Learn's* normalizer, which performs row-wise normalization on a dataset, independently scaling each row to have a specified norm, typically L1 or L2. For *TRANAE* model, we used a two-step normalization process [214]. Firstly, the data is scaled to a range [-1, 1] by dividing each element (feature) by the maximum absolute value across columns. Then, these values are scaled again to the range [0, 1] by dividing by 2 and adding 0.5. The goal of both approaches is to bring all features on a common scale, allowing ML algorithms to work more effectively and produce reliable and accurate results.

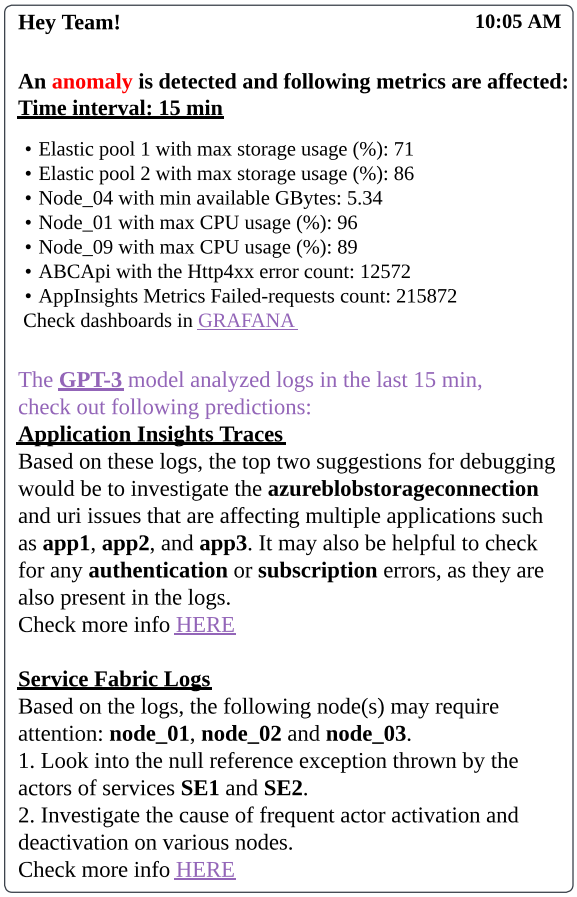


Figure 2: An example of the smart alert notification in MS Teams

ML models

The *LSTMAE* model, presented in our recent work [88], consists of two components built of LSTM (Long Short Term Memory) stack of layers which makes them suitable for capturing temporal dependencies. However, potentially, this can be computationally intensive for longer sequences. From the two components, the encoder compresses the input sequence into a fixed-size latent representation while the decoder reconstructs the original sequence from this representation. In this way, the encoder filters out noise while retaining the most relevant information and learning the *normal* model. The entire *LSTMAE* is implemented in Python using *Pytorch*, a library for building and training deep neural networks. By adjusting

Table 1: Overview of the four metric categories and the corresponding number of features used for model training

Description	Metrics	Features
Blended metrics of elastic pools	DTU Storage usage	16
Performance characteristics of Service Fabric nodes	CPU Available memory Threads count	87
Selection of metrics representing the health of internal and customer-facing services	Exceptions Http 4xx errors Http5xx errors	12
Standard metrics	Failed requests Failed dependencies Server exceptions Requests count Different performance counters	12

multiple parameters in several consecutive training runs, the lowest training loss (0.0049) is reached with input dimension 512, latent dimension 4, and 0.01 as a learning rate (see Appendix 2).

The *TRANAE* model, developed by Tuli et al. [214], is a deep transformer network with adjusted architecture for the task of anomaly detection in multivariate time series. It consists of two encoders and two decoders and leverages attention mechanisms for capturing complex temporal trends. In the first of the two adversarial training phases, the model aims to generate an approximate reconstruction of the input window. Deviations between this initial reconstruction and the actual input are used as a focus score in the second phase. Thus, this focus score is used to modify attention weights, giving higher importance to specific sub-sequences in the input data to extract short-term temporal trends. In this way, the *TRANAE* model is able to amplify deviations and capture anomalies more effectively. A more detailed description of the model is presented in the original paper [214]. The model was trained using the same environment, with the same programming language and libraries as *LSTMAE* model. We used the original architecture of the *TRANAE* model and varied only the number of epochs and learning rate during several training runs. We reached the convergence and the training loss of 0.0073 with the 10 epochs while keeping the learning rate at 0.0001.

The choice between the LSTM autoencoders and transformer networks often depends on the training data distribution and the complexity of the anomaly detection task. In this study, we evaluate both models for detecting anomalies in

multivariate metrics representing monitoring data of the cloud-based distributed system. Since the case company could not provide the ground truth data, the performance of models was only partially evaluated prior to deployment. For in-depth evaluation, the operations teams needed to review each reported alert and respond to the questionnaire regarding the overall functionality of the autonomous monitors. The results are elaborated in the following section.

Detecting anomalies

We calculated the anomaly likelihood for *LSTMAE* model as suggested by Islam et al. [96] and Ahmad et al. [3]. Thus, we maintain a record of reconstruction errors over short-term (W') and long-term (W) intervals and monitor how their mean and standard deviation change over time. Consequently, we calculate the likelihood of an anomaly occurring at time t using the formula $A_t = 1 - Q(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}), A_t \in (0, 1)$ [3]. In this equation, Q represents a Gaussian tail probability [110], $\tilde{\mu}_t$ represents the mean of the short-term interval, while μ_t and σ_t denote the mean and standard deviation of the long-term interval, respectively. If A_t is greater than or equal to $1 - \epsilon$, then the observation at time t is classified as an anomaly. Given that the approximate contamination in the training data is 5%, we set ϵ to be 0.05. The long-term interval (W) is set to 30, corresponding to 450 minutes $(30 * 15)$, while the short-term interval (W') is set to 3, corresponding to 45 minutes $(3 * 15)$. The choice of interval lengths is based on the duration of glitches and significant performance deviations in operations. We aim to investigate the effect of shorter window lengths (15 minutes) as a part of our future work, as well as using a smaller time granularity (5 minutes) for data samples.

For the *TRANAE* model, anomalies are detected using the POT statistical method, used in extreme value analysis to assess and model rare events that exceed a certain threshold. Thus, only data points exceeding a predetermined threshold value are analyzed and used to estimate the distribution of extreme values. More specifically and as proposed by Siffer et al. [192], the POT method fits Generalized Pareto Distribution (GDP) to the excesses, the difference between the observed value and threshold. In the initialization step of this method, we assumed that the excesses can be identified in the 5% of the data, which corresponds to the initial assumption about the contamination of the training data set. Thus, the *initial threshold* equals 95% of the maximum value observed within each feature. The output of this step is the first threshold z_q, computed based on the fitted distribution of excesses and fixed risk q. The objective here is to calculate the threshold value z_q such that the probability of the observed value exceeding it is smaller than q. For this step, we used the reconstruction losses from the training dataset. Further, we used the losses from the test data set to update the distribution parameters and the current threshold z_q if new values (test losses) exceed the initial threshold t. The value for the fixed risk q is determined experimentally after executing multiple runs of the POT method and trying to match the number of detected anomalies in the test data

set to predetermined contamination (5%). By using this method, we managed to compute thresholds for reconstruction losses obtained for each of the 127 features, as well as the threshold for the mean value of losses across the columns (features).

5.2 Evaluation

We conducted two stages of evaluation, pre- and post-deployment. The pre-deployment evaluation aimed at assessing the alert predictions and thus includes a brief comparison of the performance of two ML models, *LSTMAE* and *TRANAE*. The post-deployment evaluation aimed at assessing the whole concept of autonomous monitors, and thus comprises quantitative and qualitative data analysis based on feedback from the operations team.

Pre-deployment evaluation

A shortage of ground truth data prevented the evaluation of both models before deployment. Nevertheless, we compared the model outputs and corresponding thresholding methods on the subset of the dataset used for testing, which included 864 (10%) samples. Reconstruction losses calculated on the test data for each model are shown in Figure 3, where red and orange shading corresponds to data points showing anomalies. Anomalies detected with *LSTMAE* model and by calculating anomaly likelihood are highlighted in red, whereas those detected using the *TRANAE* model and POT method are highlighted in orange. The two reconstruction curves shown in Figure 3 appear very comparable, following similar patterns in certain regions with the MSE (mean square error) of $2.25 \cdot 10^{-5}$. However, due to different models' architecture, weights, and biases, the *LSTMAE* loss has a slightly higher amplitude than the *TRANAE* corresponding values. Even though reconstruction losses look alike, we could not deduce whether each of them has better reconstruction accuracy and thus better captures anomalous samples. Moreover, we used different methods for thresholding and detecting anomalies, which resulted in anomalous regions that differ to some degree. As shown in Figure 3, there is an overlap between four shaded anomalous segments detected by both approaches. Additionally, the *LSTMAE* and anomaly likelihood detected two isolated anomalous events, which were not detected with the fixed threshold obtained with the POT method. The anomaly likelihood method better adapts to changes in the data distribution over time, making it potentially more robust to varying anomalies. However, depending on the window sizes, this approach may introduce some delay in detecting anomalies compared to fixed thresholding. The POT method could eventually update the threshold based on newly detected anomalies in the time series. This would require computing the threshold in real-time, and due to its complexity, we decided to follow the original implementation [214]. By setting the threshold parameters to detect approximately 5% anomalies in streaming time

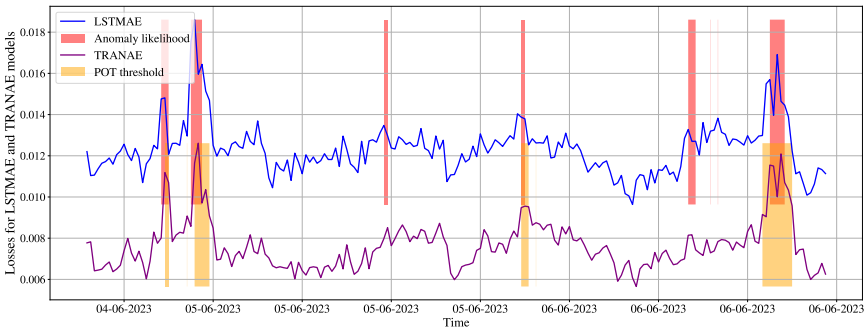


Figure 3: Anomaly detection with models *LSTMAE* and *TRANAE*: Regions in red show anomalies detected with *LSTMAE* and anomaly likelihood, while regions in orange show anomalies detected with *TRANAE* and the POT method

series (metrics), we approached the evaluation phase, where the operations team helped with the evaluation process.

Post-deployment evaluation

Due to time constraints, the evaluation period of the autonomous monitors deployed in the operations environment of the system under study was only three weeks. However, we used the first week for tuning parameters and verifying that all components shown in Figure 1 work as expected. Thus, the operations teams reviewed reported alert notifications during the remaining two weeks. The results of this evaluation period are shown in Figure 4.

The main criterion for reviewing reported alerts was the applicability of GPT-3 predictions to underlying issues based on log inputs. Even though both the *LSTMAE* and *TRANAE* models would identify a deviation in metrics, this perceived ap-

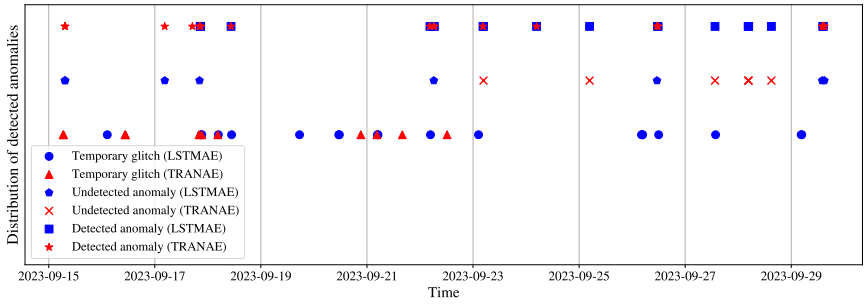


Figure 4: Overview of results based on the feedback from the operations team

plicability of GPT-3 predictions ultimately determined whether the detected event was an anomaly or just a temporary glitch. The total number of detected anomalies is 26, where $17(65.4\%)$ were detected by *LSTMAE* and $20(76.9\%)$ of them by the *TRANAE* model, with an overlap of 11 alerts. Temporary glitches are identified whenever there was a vague or recurring prediction by GPT-3. As shown in Figure 4, these false positive predictions appear in most cases before or after actual anomalies, even though there were some single events. This indicates that the thresholds need to be adjusted upward to meet the requirements of the case company. However, a positive observation is that an anomalous event was always detected by one of the deployed models, where the *TRANAE* model demonstrated slightly higher precision by $76.9 - 65.4 = 11.5\%$. Given that none of the anomalies are missed, the F1 score equals 0.64, representing the balanced measure of the performance of *smart alerts*. This result is still sufficiently good considering the customized definition of the anomaly construct, elaborated in Section 6.2.

To get a comprehensive overview of usage and user experience for the *smart alerts* in a new MS Teams Alert Channel, we conducted a survey among members of the operations teams. We created a questionnaire using Microsoft Form, including a list of eight questions that can be found in Appendix 1. We present results from five unique perspectives based on the seven received responses.

General insights. All respondents had very positive reactions regarding the new alert channel. They stated that the new way of reporting alerts is very useful and provides easy access to important information regarding the health of their system. Additionally, one respondent highlighted that *smart alerts* allow dynamic notifications for unexpected spikes in usage outside the regular daily patterns. This is a significant improvement compared to Microsoft's alerting system (IF/THEN rules with predefined thresholds).

Functionality. According to the respondents, the *smart alerts* provide a complementary view of the health state compared to the existing dashboards in Grafana. Moreover, they offer a better overview of components that should be further investigated, including various factors that could be of interest from an operational perspective. Overall, the combination of metrics and logs analysis for detecting deviations and predicting root cause suggestions proved to be a good match for identifying and resolving critical issues that affect customers.

GPT3-predictions. All respondents shared the same opinion regarding the GPT-3 suggestions for root cause analysis. They agree that suggestions are quite reasonable and offer clear and actionable guidelines on how to approach specific issues. However, the quality of predictions differs for some alert notifications depending on the input logs. Thus, predictions can sometimes be vague, requiring additional efforts to understand and resolve detected deviations in metrics.

Usage. The majority of respondents pointed out that they would use *smart alerts* for initial investigation before temporary anomalies roll out to severe issues that might affect customers. Additionally, one of the respondents brought up an interesting remark that the new alert channel would be helpful in the QA envi-

ronment for preventing failures during releases. However, they also shared their concern regarding the GTP-3 predictions, as they could overwhelm the operations team with the same or similar predictions. Even though they are related to recurring errors that are not immediately resolved, they could lower the overall usage of *smart alerts*.

Improvements. Even though some of the respondents were satisfied with the visual and functional aspects of *smart alerts*, we still received some very useful recommendations. They highlighted that the upper part of the notification should contain a better summary that stands out visually. In this way, they could scroll through the list of reported alerts without expanding the notification to conclude about the alert severity. Moreover, a few respondents pointed out that reported alerts should include only the most critical metrics, which correspond to setting the higher thresholds. One interesting comment is that an improved version of alert notification should include customer impact analysis. These recommendations will be considered in our future work.

Implementing and evaluating machine learning models in an industrial setting remains a challenging task. The difficulty increases even more when ground truth data is unavailable, as in our case. A priori assumptions and tuned parameters might not hold after deployment in operations and thus will require adaptations and fine-tuning. As suggested in our recent work [88], several iterations might be needed to reach an optimal version of the solution that meets the demands of the industrial context.

6 Discussion

In this section, we discuss the main findings of this study, their alignment with the contributions outlined in Section 1, and identified threats to validity and related mitigation actions.

6.1 Findings

As mentioned in Section 3, the goal of this study was to extend the validity of the technological rule implemented and evaluated in the context of the microservice system for ticket and payment management in public transportation (TPM system) [88]. The new context introduced a different cloud-based system in terms of functionality but similar in regards to the cloud architecture and environment. Having the same cloud provider of services for deployment and monitoring, Microsoft Azure, enabled an easy transition between the contexts. However, the monitoring data characterizing the health of both systems partially differed, as some parts of the systems were implemented and deployed using different cloud services. Additionally, the distinct functionalities of the systems contributed to prioritizing different data types, such as database-related metrics for the PIM system, since managing customer data was one of the main functionalities. Regarding

the source of the monitoring data, the TPM system mainly relied on performance metrics accessible in Applications Insights, while obtaining critical observations of the PIM system required executing customized queries in Applications Insigts and Log Analytics. These custom queries targeted specific parts of the system and environments and were defined by operations teams. This was a slightly undesired circumstance as fetching the data in near-real-time needed to be implemented with four Azure Functions. On the contrary, for the TPM system, the specific performance counters were accessed by running simple HTTP requests against Azure Metrics APIs.

By working on these two case studies, we confirmed a well-known principle stating there is no one-size-fits-all solution. The purpose of the implemented autonomous monitors was to improve reporting and managing alerts, consequently making it highly dependent on the available monitoring data. Thus, the monitoring data types, their source, and accessibility define the level of generalizability. Regarding ML solutions (*LSTMAE* and *TRANAE* or other ML models), they require the number of input features and tuning of parameters in each context. Hence, training ML models is straightforward if data collection and preprocessing are correctly carried out. Automating the entire workflow shouldn't be troublesome either in different contexts. Using different cloud solutions might eventually require a few more lines of code or a different mechanism for pulling the data from monitoring platforms. Thus, the case-based generalization of the proposed autonomous monitors is feasible for cloud-based systems that share compatible monitoring data types and tools for monitoring. For other incompatible systems and environments, additional efforts are required to understand the distribution of available data and preprocessing methods. However, we conclude that opportunities for *smart alerts* in multiple contexts are tremendous, but different levels of engagement might be needed depending on the data complexity and availability. We refer to this observation as contribution **C3**.

Besides case-based generalization, which included implementing and evaluating the autonomous monitors in another case context, we aimed to improve the proposed solution design in terms of functionality. We were mainly inspired by the observation from the previous case study [88], where we identified that operations teams were not sufficiently motivated to examine reported alerts, as they could not conclude the severity level based on the provided information. They required additional information upon which they could take action. Accordingly, we decided to include logs in the analysis and propose a novel approach for reporting interpretable alerts (contribution **C1**). Based on the qualitative feedback, the *smart alerts* containing actionable guidelines for root cause analysis seemed very appealing to the operations engineers. However, minor adjustments might be needed regarding the visual aspect to enable smooth analysis and investigation of reported alert notifications.

Even though our initial solution [88] included the widely adopted method for detecting anomalies in multivariate time series (*LSTMAE* model), we decided to

enrich the original design of the autonomous monitors considering the latest advancements in the ML field. Thus, the solution by Tuli et al. [214] seemed like a perfect fit as it showed very promising results on the publicly available datasets and only needed to be instantiated and evaluated in an industrial setting. We refer to this validity extension as contribution **C2**. Industrial contexts without ground truth data are always challenging, as testing selected ML models before deployment is not feasible. Furthermore, the distribution of data used for training might affect the learning process and bring even more uncertainty to the evaluation phase. In our case, the *TRANAE* model showed solid performance and higher precision rate by 11.5% compared to the *LSTMAE* model. It can be used to detect both short- and long-term anomalies, as shown in Figure 3. Due to its high potential, it will be used for similar solutions in our future work.

6.2 Threats to validity

To increase the overall validity and relevance of the proposed autonomous monitors, we considered the latest advancements in the ML world during the design development. For this purpose, we surveyed state-of-the-art machine learning techniques and explored recent innovations. Furthermore, we established a close relationship with our industry partner, involving interviews and recurrent dialogues. Potential threats to validity are discussed in the context of construct, conclusion, internal, and external validity.

Construct validity. The central construct of this study is the notion of *anomaly*. It is hard to precisely define and distinguish from other events that indicate variations, but are still within the range of what characterizes a healthy system. In our approach, we move away from the threshold-based definitions of anomalies and train ML models. These models are no perfect representation of the anomaly construct, but our additional GPT-3 analysis helps the operators to analyze the events in depth and judge whether they are anomalous or not.

Conclusion validity. The duration of the evaluation period (which spans 14 days) could potentially have an impact on drawn conclusions. Additionally, reported alert notifications were reviewed by only two engineers from the operations team, thus, the final results might be biased. To address this challenge, the operations team will continue using our anomaly detection solution even after the study concludes. Additionally, we are preparing for more comprehensive evaluations and discussions with practitioners following a considerably longer period of use in the future.

Internal validity. Selection and preprocessing of the features can significantly impact the performance of ML models and overall results. The 127 features described in Section 4 were carefully selected based on the very thorough observations of monitoring data visualized in Grafana and inputs we received from the operations team. We mainly focused on data types (features) that were usually monitored and examined when some severe failures were detected. However, con-

sidering the system dimensionality and the volume of available monitoring data, the selected set of features could be adjusted in the future by adding new or removing some of the existing features. This will, however, require training new ML models, but revising the existing and exploring new features is necessary for building trustworthy and reliable machine learning solutions.

External validity. With this study, we extend the validity of our solution to another industrial context. However, this still limits the scope of the solution to cloud-based systems deployed and monitored using MS Azure. Considering the similarity between different providers of cloud solutions, we assume that the same solution could be implemented using matching cloud tools like Amazon CloudWatch and corresponding cloud services. This will be further investigated in our future studies initiated with other industrial partners.

7 Conclusion

Using ML-inspired solutions for improving software engineering environments and different quality aspects of cloud-based systems has become a standard nowadays. The possibilities are endless and yet to be explored. In this paper, we report one of the applied solutions that leverage state-of-the-art ML models and cloud-based services for the timely detection of anomalies and reporting of interpretable alerts. To remain at the forefront of recent research, we included ML models (*TRANAE* [214] and *GPT-3* [162]) based on deep transformer networks in the design of the solution, the autonomous monitors. Thus, we extended the validity scope for the *TRANAE* model as it has not been evaluated previously in the industrial context of the cloud-based system. Additionally, the validity was extended for the overall solution for improving alert management, expressed in the form of the technological rule in our recent work [88]. The solution design for reporting *smart alerts* proved to be efficient in proactive monitoring and early investigation of raised issues before they escalate and cause severe consequences.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We thank the DevOps and CloudOps teams for their willingness to share insights and respond to our questions.

Appendix 1: Questionnaire form

The DevOps and CloudOps teams were asked to fill in a questionnaire form, where we provided a detailed description of the autonomous monitors and the following

questions:

1. What is your general opinion regarding the ML Channel and new alert reports?
2. Does ML Channel provide more (useful) information than the current one (General)?
3. Is it easier to understand current health state of different parts of the system?
4. Does GPT-3 give reasonable suggestions for root cause analysis?
5. Would you use ML Channel for investigation of raised issues (by customers)?
6. Does ML Channel produce more alerts than DevOps and CloudOps teams are able to handle during the day?
7. Is there anything you would like to change in the visual representation or functionality?
8. Would you be willing to provide additional details or clarification if needed regarding your response?

Appendix 2: Online Resources

All Python scripts and Jupyter notebooks used in the study are available upon request at `https://github.com/adha7/inAlerts-icse` due to privacy concerns of the case company. Implementation of the *TRANAE* model by Tuli et al. [214] can be found at the `https://github.com/imperial-qore/TranAD`.

PAPER IV

ADVANCING SOFTWARE MONITORING: AN INDUSTRY SURVEY ON ML-DRIVEN ALERT MANAGEMENT STRATEGIES

Adha Hrusto, Per Runeson, Emelie Engström, Magnus C Ohlsson

Abstract

With the dynamic nature of modern software development and operations environments and the increasing complexity of cloud-based software systems, traditional monitoring practices are often insufficient to timely identify and handle unexpected operational failures. To address these challenges, this paper presents the findings from a quantitative industry survey focused on the application of Machine Learning (ML) to enhance software monitoring and alert management strategies. The survey targets industry professionals, aiming to understand the current challenges and future trends in ML-driven software monitoring. We analyze 25 responses from 11 different software companies to conclude if and how ML is being integrated into their monitoring systems. Key findings revealed a growing but still limited reliance on ML to intelligently filter raw monitoring data, prioritize issues, and respond to system alerts, thereby improving operational efficiency and system reliability. The paper also discusses the barriers to adopting ML-based solutions and provides insights into the future direction of software monitoring.

Euromicro SEAA-KKIO (2024)

1 Introduction

Large-scale cloud computing systems can fail in various unpredictable ways, with faults cascading across components, leading to service outages or degraded performance. Understanding how these systems behave under failure conditions is crucial for planning effective failure management and prevention strategies [33]. Effective monitoring plays a vital role in detecting these operational failures. Software development companies rely heavily on monitoring tools [65] that collect and analyze vast amounts of operational data. However, the large volume and complexity of monitoring data pose challenges in identifying meaningful patterns and extracting actionable insights. Traditional monitoring systems often struggle to effectively manage data overload, leading to alert fatigue and the potential overlooking of critical alerts. Moreover, there is a lack of standardization and an overabundance of different monitoring tools, which leads to inconsistent and ineffective monitoring practices [205].

To address these challenges, Machine Learning (ML) techniques have emerged as a promising approach to enhance the analysis and management of monitoring data [149, 212]. ML can significantly improve system observability and enable extensive system performance analysis to identify anomalies that could indicate a potential failure as shown in our previous case studies [88–90]. However, despite significant research and commercial solutions for advanced monitoring using ML, many organizations have yet to utilize these capabilities fully. Current monitoring methods primarily depend on alert thresholding for key performance indicators (KPIs) or log querying, with limited use of ML-based solutions due to uncertainties about their usefulness, reliability, and cost-effectiveness [174].

To advance understanding of the aforementioned challenges in monitoring cloud computing systems and adopting ML-based approaches, our industry-academia collaboration research team conducted a national quantitative survey among industry practitioners. We targeted companies responsible for developing, testing, and operating cloud-based software systems. By engaging with practitioners who are directly involved in the lifecycle of such systems, we obtained a comprehensive understanding of the current monitoring practices, their limitations, and the potential impact of incorporating advanced ML techniques for more efficient monitoring strategies. More details about the survey setup can be found in Section 3.

Despite the growing interest in ML technologies and their potential applications [33, 193], there is a scarcity of comprehensive survey studies specifically focusing on how industries are implementing ML for monitoring purposes and detecting operational failures. Most recent and relevant surveys [195, 205] provide valuable insights, but each from unique and differing viewpoints regarding cloud monitoring practices and the adoption of machine learning in the industry, particularly for software failure prediction. Thus, there is a need for more focused research in this area that integrates both perspectives, to understand the benefits, practical challenges, and the extent of ML adoption in industrial monitoring and

proactive alert management. Therefore, this study aims to contribute to this area by providing empirical insights into the current state of ML usage in industrial monitoring and exploring the factors influencing its adoption and effectiveness for early detection of operational failures.

2 Background and Related Work

Recent studies have significantly advanced software monitoring, particularly through ML-driven approaches, highlighting the challenges and innovations in this area. The review by Giamattei et al. [65] explores a variety of monitoring tools for large-scale systems like microservices, noting challenges in their selection and usage. Research by Candido et al. [36] addresses complexities in log data and introduces AIOps for improving operational workflows. *The IntelligentMonitor* study [209] discusses an adaptive system that reduces data overload and alert fatigue through ML, enhancing monitoring efficiency. Additionally, Gill and Hevary [66] identify major challenges in cloud monitoring, including issues in technology, virtualization, and performance, emphasizing the need for innovative solutions.

According to the recent survey by Tamburri et al., [205], it is evident that monitoring practices are crucial for detecting operational failures. Additionally, monitoring should be recognized as a strategic asset for improving system observability [205]. There are examples of successful intentions to address the early detection of failures by leveraging data accessible through monitoring tools. Mariani et al. [149] introduce PreMiSE, a method that predicts failures in multi-tier distributed systems. This approach, tested on a telecommunication system prototype, showcases high precision in failure prediction with minimal false positives. Similarly, Cotroneo et al. [33] propose a method for analyzing failure data in cloud systems, leveraging Deep Embedded Clustering to classify failures efficiently without manual feature engineering.

However, despite the availability of numerous monitoring tools, Tamburri et al. [205] find that adopting advanced monitoring technology in the industry is still in its early stages due to required substantial investments and lack of industry standards. To investigate this further, our survey study aims to understand to what extent the companies leverage monitoring tools and data to detect operational failures and perform root-cause analysis, beyond single case studies.

Hrusto et al. have undertaken two case studies in collaboration with industrial partners [88–90]. They reveal current alert management practices and the limitations of existing monitoring solutions. Interviews and observations in the case studies highlight specific challenges, such as undetected operational failures, alert flooding, difficulty in interpreting alerts, and the need for more efficient alert mechanisms, such as autonomous monitors [90]. To address these, they developed and evaluated a cloud-based solution for monitoring, detecting anomalies, and reporting interpretable alerts.

Additionally, it is crucial to understand the integration of AI and ML in the industry, as these technologies play a significant role in enhancing productivity and decision-making. Surveys by Rana et al. [174] and Holmström [83] offer insights into the factors influencing AI/ML adoption. Our study adds a practical perspective by evaluating the real-world applicability of AI/ML in software development, helping to bridge the gap between theoretical frameworks and industry implementation.

3 Research methodology

We conduct an industrial quantitative survey study following the recommendations and guidelines from three key publications on designing and conducting surveys in software engineering authored by Molléri et al. [155], Kasunic [112], and Linåker et al. [136]. The survey process involves several key steps [155], detailed in subsequent subsections, including reflections on threats to validity important for ensuring the reliability and credibility of research findings.

3.1 Research objective

The survey aims to describe challenges in handling operational failures and monitoring data in a set of Swedish software development companies. Since our aim is to observe the existence of challenges and assess the validity of proposed solutions, we conduct a judgment study with experts rather than a sample survey [200]. We report the occurrence, detection mechanisms, and types of failures while considering the use of monitoring data for better understanding and designing prevention mechanisms. Further, we evaluate the benefits and limitations of ML-based anomaly detection and alert tool [90], assessing its broader industry applicability. Additionally, we analyze the impact of AI/ML solutions on developing, testing, and operating cloud-based software systems and the organizations' capability to adopt these tools.

3.2 Research questions

We defined the research questions based on a synthesis of the authors' industry experiences and the latest contributions in the field, as outlined in Section 2 as follows:

- **RQ1:** How do different types of operational failures impact software development environments, considering their occurrence and consequences?
- **RQ2:** To what extent are monitoring data and its specific types used for detecting and analyzing operational failures?

- **RQ3:** Are there recognized needs for more advanced detection (alert) mechanisms in operations based on state-of-the-art machine learning approaches?
- **RQ4:** What is the current level of readiness and attitude of software development, testing, and operations teams towards adopting and relying on ML-based solutions, given the latest advancements in AI?

3.3 Defining and sampling the population

We target a population of Swedish software companies that develop and operate software systems deployed in the cloud, referred to as *software development companies*. Their names are not disclosed on their request to remain anonymous. The target audience consists of intended respondents from these companies, to whom we refer as *software practitioners*.

The first and fourth authors are employed at a global software quality assurance (GSQA) company with headquarters in Sweden that offers a wide range of services within the Software Development Life Cycle (SDLC). The company has a significantly large network of loyal customers across Sweden who focus on high-quality software products and processes, which we aim to examine in our study. We used a non-probabilistic sample from this customer network, combining convenience and purposive sampling, as discussed by Baltes and Ralph [11]. To minimize biases, we expand the sample with the authors' LinkedIn connections, specifically targeting software practitioners from companies that prioritize software quality in their businesses.

3.4 Designing and validating the instrument

We used a structured questionnaire as a survey instrument for data collection. The questionnaire was carefully designed to include closed-ended questions, which offered respondents both single- and multiple-response options [155]. This format was chosen to simplify the response process and to ensure consistency and comparability in the collected data. We utilized a specialized survey design tool provided by our university to develop the questionnaire, ensuring a rigorous and systematic approach to data collection. We formulated questions with predefined categorical responses specifically designed based on the authors' experiences and considering definitions of quality characteristics standardized by ISO[1]. The survey assesses software quality across *usability* (user roles and experience), *reliability* (operational failures, detection methods, and alerts), and *maintainability* (monitoring tools and data analysis), while also evaluating the impact of ML solutions on *efficiency* and *functionality* of cloud-based systems.

The questionnaire form, openly available[2], begins with an overview section that includes the project title, the research team involved, the purpose of the survey,

[1] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[2] https://doi.org/10.5281/zenodo.10986352

and information on data privacy. This part sets the context for the respondents and assures them of the confidentiality of their responses. The questionnaire is structured into distinct sections corresponding to our research questions:

- Respondent Profile: This section comprises six questions designed to gather respondents' demographic and professional background information.
- Operational Failures and Monitoring Data Usage: It consists of nine questions, targeting **RQ1** and **RQ2** to collect respondents' experiences and perceptions regarding operational failures and monitoring practices in their respective environments.
- ML for Smart Monitoring & Alerts: This segment includes eight questions focused on using ML in smart monitoring and alert management (**RQ3**).
- Adoption of ML-based Solutions: The final section, with seven questions, explores the attitudes and readiness of respondents towards adopting ML-based solutions in their work processes (**RQ4**).

To ensure the validity of our survey, our team leveraged its expertise to align the survey questions accurately with the research questions. One author with in-depth domain knowledge designed the initial questionnaire. Three other authors, with academic and industrial insights, reviewed it thoroughly. This review identified several concerns, leading to collaborative discussions that refined and validated the questions.

3.5 Managing participants and responses

In managing participant engagement and responses, our approach involved the distribution of the online questionnaire through three primary digital channels: email, Microsoft Teams, and Viva Engage. To effectively reach potential respondents, we shared the survey URL along with a brief context description, explaining the purpose of the study and its significance. We directly reached out to 30 potential respondents through our professional network, including consultants at GSQA company and contacts on LinkedIn, and relied on them to further distribute the questionnaire among their respective teams and companies. Throughout the survey period of four weeks, we actively monitored the response rate to gauge participant involvement. Based on these observations, we regularly reached out to all previously contacted individuals to check on the status of their participation, as well as to express our gratitude and encourage their ongoing involvement, acknowledging their valuable contributions. This strategy of regular communication and appreciation played a crucial role in maintaining a good response rate.

3.6 Analyzing and reporting results

We initially exported the results into an Excel file to analyze and report our survey results. We manually inspected the file and determined the most efficient approach for detailed analysis. To facilitate this, we divided the results into five separate CSV files corresponding to the respondent profile and the four research questions. Each question and its corresponding answers were coded systematically (e.g., F1 for a question and F1A1 for its answer related to operational failures), streamlining the process of data handling. This structured coding system enabled us to efficiently write Python code snippets for loading, analyzing, and extracting relevant information that directly addressed our constructs and survey objectives. For comprehensive transparency and reference, the auto-generated report from the survey tool, encompassing all the detailed results, is openly available[3].

3.7 Threats to validity

Addressing potential threats to validity is crucial for ensuring the accuracy and credibility of our survey findings [155]. We considered the following threats to validity:

- **Internal Validity** – We ensured through internal reviews that questions were designed to minimize misunderstanding or ambiguity and that the survey was distributed to a representative sample of the target population.
- **Construct Validity** – The brief description of the ML-based anomaly detection tool in the questionnaire could have affected respondents' ability to accurately answer related questions, thereby impacting the reliability of our findings. We assumed that respondents would have sufficient background knowledge to understand the main goal of the described tool, given the widespread adoption and growing use of AI in the software industry. However, we also included a question regarding their experience in AI to gauge their familiarity.
- **External Validity** – Given that our sample was non-probabilistic and drawn entirely from Sweden, the generalizability of our findings to a global context is limited. However, we aimed to expand upon our previous findings from two Swedish case companies, focusing on generalizing to a broader, but yet limited, population of similar companies. To achieve this, we included participants from different software engineering subfields in Sweden to capture diverse population characteristics.
- **Conclusion Validity** – We ensured this by employing a systematic approach that combined manual data inspection with automated analysis using a Python

[3] https://doi.org/10.5281/zenodo.10986352

script. This approach enabled us to thoroughly examine the data while efficiently extracting information relevant to our research questions.

4 Results

This section presents the outcomes of our quantitative survey, structured to address each of our four research questions in Section 3.2. Each subsection provides a comprehensive analysis of the survey responses, offering a detailed understanding of the current state and future directions in operational failure management and machine learning adoption in software development environments. The results are analyzed based on 25 responses across 11 different software development companies, out of 30 invitations. We examined the data from both the company and software practitioner perspectives.

4.1 Respondent profiles

The respondent profiles, shown in Table 1, include a diverse spectrum of positions primarily in the software industry. Quality assurance is the most prevalent role within the target audience (10/25 from 8/11 companies), but DevOps engineers (6/25 from 6/11 companies) and software developers (5/25 from 4/11 companies) are also dominant positions. The range of roles also includes two CloudOps engineers, one test manager, and one project manager.

These practitioners are sourced from a diverse range of companies, where the number of companies corresponding to each position follows a similar distribution. In terms of professional experience, there's a wide range, from those with less than a year to those with over a decade in their field. The experience with AI tools varies, with a significant number of respondents having engaged with AI either a few times or moderately, indicating a growing interest in using AI to enhance their working processes.

Table 1: Overview of the respondent profiles

Position	No. of respondents per position	No. of unique companies
Quality Assurance	10	8
DevOps Engineer	6	6
Software Developer	5	4
CloudOps Engineer	2	2
Test Manager	1	1
Project Manager	1	1

4.2 RQ1: Operational failures

The main objective of this research question was to investigate one of the critical aspects of software development related to the types, occurrence, and consequences of operational failures. The results showed that the occurrence of operational failures varies across companies, reflecting the differing resilience and vulnerability of systems in the industry. With seven companies experiencing weekly failures and three of them encountering them monthly, it's evident that operational failures are a common and recurrent challenge. Interestingly, respondents in different roles within the same company often reported different occurrences of operational failures, indicating role-specific challenges and perspectives on issues. To address this discrepancy, we considered the worst-case scenario as the reference. This means that when respondents from a single company reported different occurrences of failures, we considered the more frequent instance as the baseline. In this way, we focused on addressing the most significant and recurrent operational failures.

The reported methods for detecting operational failures show the industry's reliance on technology and human oversight. Companies are increasingly integrating technology-driven methods, such as automated alert systems, used by all eleven surveyed companies, with human-centric approaches like manual monitoring and user reports, employed by eight companies each. Interestingly, some companies employ a mix of these methods to create a more robust and comprehensive detection strategy. These blended strategies for detecting operational failures highlight the need for more comprehensive monitoring and detection mechanisms.

Continuing the analysis, the collected data from the surveyed companies revealed that the most frequent problems arise from the interaction between different software components. This was reported by nine companies, indicating the importance of the integration or compatibility issues. Following closely are performance issues, signalizing the difficulties the companies face in maintaining optimal system performance and response times. System outages or crashes also represent a major concern, as they were experienced by six companies. This demonstrated an urgent need for robust infrastructure and proactive maintenance to minimize downtime. Although data-related issues and security vulnerabilities are less frequent among operational failures, their relevance is still highly important. Data corruption or loss can have severe consequences for data integrity and reliability, while even a single security incident can cause data breaches, financial losses, and damage to a company's reputation. Considering the broad range of potential failures, their early identification is crucial for maintaining overall system health.

An additional overview of operational failures per company is given in Figure 1. It becomes evident that certain operational failures are more prevalent in specific companies, indicating potentially unique challenges or vulnerabilities within their operational environments. For instance, the results suggest that many companies should direct their improvement efforts to address performance issues.

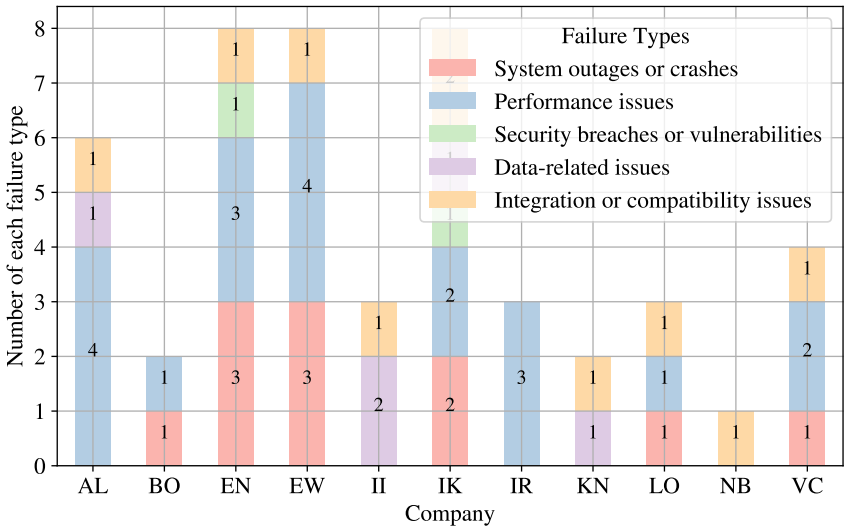


Figure 1: Distribution of different failure types per company

Complementing these findings, the results also indicate consequences of operational failures, shown in Figure 2. It was evident that several companies experienced a range of failure consequences simultaneously. Among the closely distributed four responses (blue, green, purple, and yellow), the highest rate of companies, 9/11 (81.8%), reported increased development time and costs due to operational failures, highlighting a burden on resource allocation and project timelines. A comparable rate of 8/11 (72.7%) experienced impacts on both product quality and customer satisfaction, indicating the extended effects on business reputation and profitability. An increase in technical debt, reported by 7/11 (63.3%), points to the accumulating challenges in software maintenance and development efficiency. These insights collectively revealed the complex nature of operational failures yet nearly even distribution of consequences affecting various aspects of business.

The findings of RQ1 highlight the extensive and persistent challenges regarding operational failures in the software industry. This indicates an urgent need for proactive and effective mitigation strategies. The diversity of these failures, as shown in Figure 1, emphasizes the necessity for robust monitoring solutions capable of addressing a wide range of issues.

4.3 RQ2: Usage of monitoring data

With RQ2, we aimed to understand the evolving landscape of monitoring data management, which is crucial in detecting and analyzing operational failures. Fur-

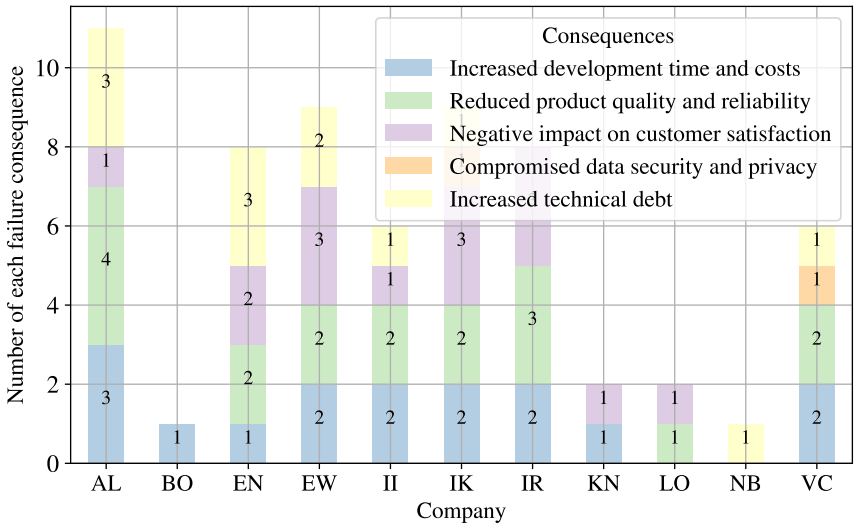


Figure 2: Distribution of different types of failure consequences per company

thermore, we examined the utilization of diverse monitoring tools and alert mechanisms and considered their impact in providing real-time insights and proactive responses to operational anomalies.

The data reveals how different organizational roles utilize monitoring data. DevOps engineers and quality assurance teams primarily engage with it for continuous integration and quality checks, focusing on system vulnerabilities and integration issues. Managers, though less represented, primarily rely on data for decision-making and oversight. In contrast, CloudOps engineers and software developers use data to track performance trends and identify vulnerabilities, aligning with their responsibilities in cloud infrastructure and software development. These patterns highlight the tailored applications of monitoring data to meet specific role-based demands.

Next, we analyze the results related to the usage of different monitoring tools and types of monitoring data, shown in Figure 3. Among the monitoring tools, Amazon CloudWatch, Azure Monitor, and Grafana emerge as the organizational leaders, signaling their widespread acceptance likely due to robust features and seamless integration capabilities with respective cloud environments. In contrast, tools like Prometheus and Kibana, while still utilized, show a relatively lesser degree of adoption, which may reflect preferences based on specific organizational needs or system compatibility.

When analyzing the types of monitoring data, performance metrics, particularly focusing on CPU and memory usage, stand out as the most monitored data type in Figure 3. This dominant usage highlights the critical importance of sys-

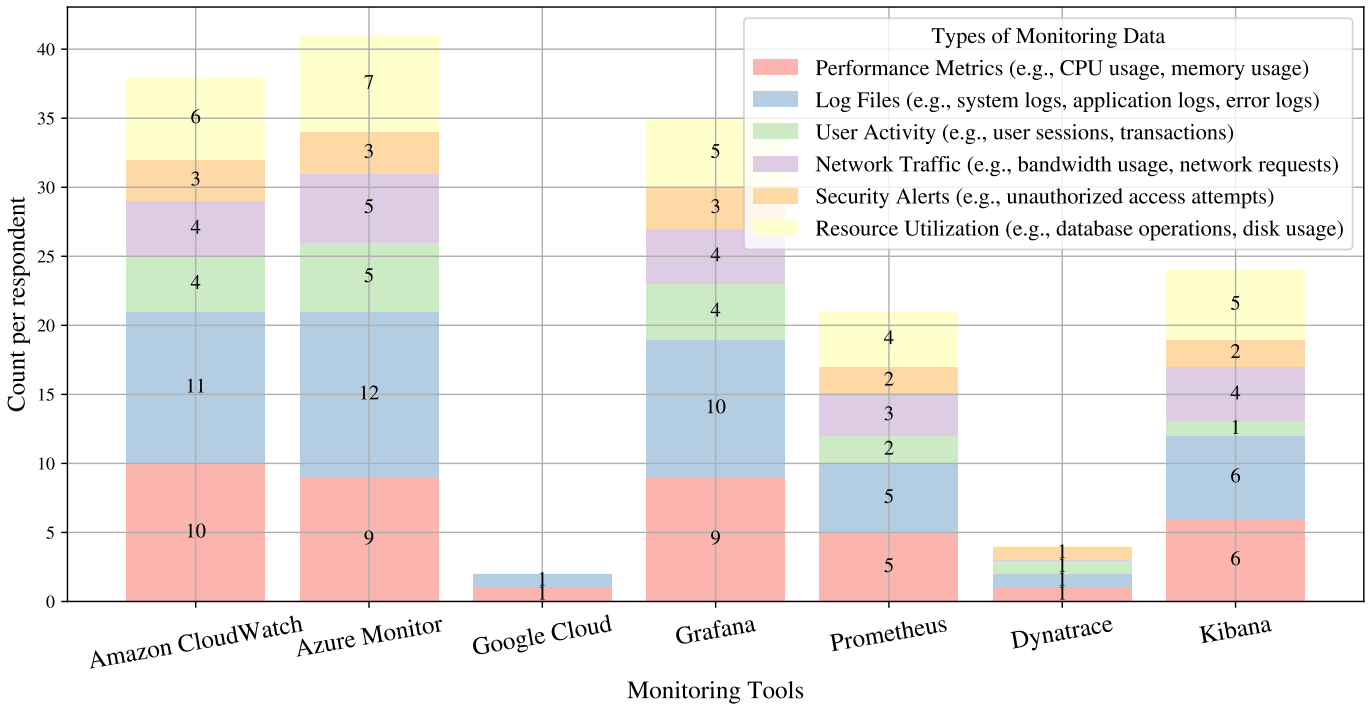


Figure 3: Overview of monitoring tools and types of monitoring data per respondent out of 25

tem performance and cloud infrastructure optimization in maintaining operational efficiency. Log files, comprising system, application, and error logs, demonstrate a slightly higher usage compared to performance metrics, highlighting their importance in diagnosing issues and gaining insights into operational failures. We have additionally identified that there is a consistent distribution of different types of monitoring data across a range of monitoring tools. This uniformity suggests that irrespective of the specific monitoring tool employed, the aforementioned data types are always prioritized.

In comparison, resource utilization, user activity, and network traffic, although integral to comprehensive system monitoring, appear to be less prioritized. This may suggest an area for increased focus, especially considering the insights they can provide into user behavior and network efficiency. Security alerts remain a significant concern even though they don't dominate in usage, indicating a relatively limited emphasis on identifying and mitigating potential security risks. The data collectively illustrates diverse monitoring approaches using various tools tailored to specific system health and security needs, emphasizing strategic organizational choices.

Many commercial monitoring tools offer the possibility to configure different alert mechanisms, which can be highly significant for timely identification of operations failures. The survey results reveal that a larger proportion of companies reported having basic or limited alert systems, compared to those with fully integrated systems, suggesting a trend towards incremental adoption of sophisticated monitoring tools and alert strategies. Interestingly, most of these companies consider their systems to be very effective, indicating a positive reception towards the existing alert mechanisms despite their varying levels of complexity. However, fewer companies are in the process of implementing or planning to implement alert systems. This shows a growing awareness and need for advanced monitoring solutions. This industry shift towards proactive, efficient monitoring strategies underscores the growing importance of effective alert systems for organizational resilience and efficiency.

4.4 RQ3: ML for advanced monitoring and alerting

The survey findings revealed a diverse perspective among different organizational roles, as shown in Figure 4. Quality assurance practitioners mainly reported a moderate need, with a few noting a strong need, which may denote a balanced perspective on integrating advanced technologies and the possible benefits for quality assurance processes. DevOps and CloudOps engineers, with the majority perceiving little and moderate need, expressed satisfaction with current monitoring systems or possible concerns associated with implementing and maintaining such ML-driven solutions. On the contrary, developers recognized a strong need for advanced monitoring, most likely because they are usually impacted by operational failures and are mainly responsible for their resolution. Interestingly, manager

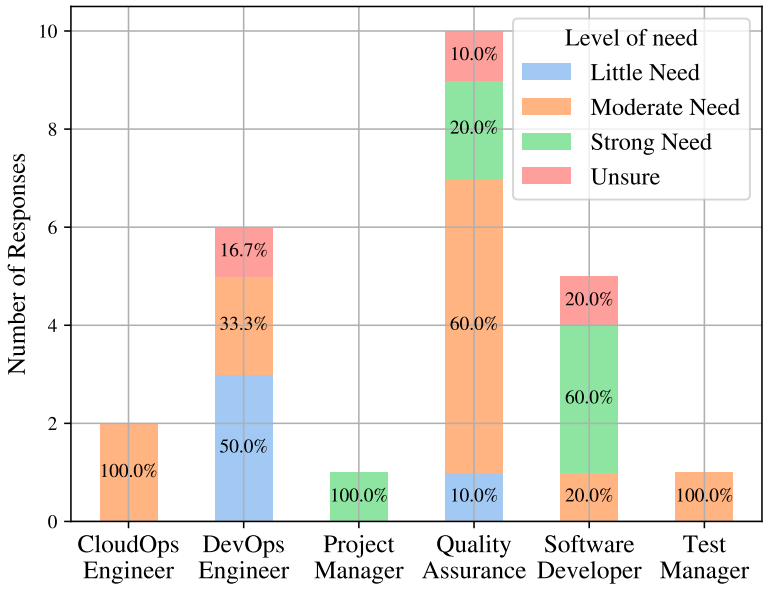


Figure 4: Need for advanced analysis of monitoring data perceived by different organizational roles

roles pointed towards a strong and moderate need for ML-driven monitoring solutions, which may reflect their prioritization of efficiency and risk management in project delivery. These varied responses highlight the complexity of organizational readiness for advanced ML monitoring solutions, emphasizing the relevance of our study to understanding current and future trends.

As previously explained in Section 2, one of the reasons for examining the applicability of ML-based monitoring solutions was inspired by the authors' recent work [90]. For this purpose, we specifically constructed questions to target the benefits and challenges of such solutions. Table 2 highlights the top three benefits and challenges identified in our analysis. Among the benefits, *the fastest detection of operational failures* stands out as the most acknowledged advantage, marked by 23 out of 25 respondents, highlighting its significance. This suggests a strong agreement on the importance of ML in quickly identifying operational issues, which is crucial for timely problem-solving and efficiency. The second most frequent answer, *the reduction in manual monitoring efforts*, indicates a significant awareness of ML capabilities, highlighting its role in easing the workload of human operators. *More accurate root cause analysis*, although rated lower, is still notably recognized as a benefit, pointing to the analytical strengths of ML in diagnosing issues.

The top three identified challenges, including *integration with existing systems*,

Table 2: Top three benefits and challenges of ML solution for reporting interpretable alerts

Benefits	**# respondents**
Faster detection of operational failures	23
Reduction in manual monitoring efforts	19
More accurate root cause analysis	15
Challenges	**# respondents**
Integration with existing systems	15
Cost and resource allocation	15
Data privacy and security concerns	15

cost and resource allocation, and *data privacy and security concerns*, shared the highest score, with each receiving 15 responses. This uniformity in responses suggests a balanced perception of these challenges, emphasizing that while ML implementation is promising, it comes with a set of equally important considerations. These include technical integration complexities, significant investment requirements, and the critical need to maintain data integrity and security. The balanced view on these challenges reflects a well-rounded understanding of what implementing such a solution entails, emphasizing the need for strategic planning and resource management.

The aforementioned ML-based solution for proactive monitoring [90] utilized the capabilities of the GPT3.5 Turbo model for getting interpretations of the log data. Therefore, we aimed to understand how confident different organizational roles are in the predictions from such a large language model. A significant percentage of participants found *GPT suggestions somewhat effective* and primarily helpful for basic alerts and routine tasks. A relatively smaller group acknowledged the effectiveness of GPT in accurately identifying issues and suggesting solutions. However, only a few respondents rated GPT as highly effective, pointing to its potential for delivering in-depth analysis and actionable solutions, while a minimal number perceived it as not effective, highlighting limitations in certain contexts. In terms of confidence in GPT's predictions, a similar pattern emerges. Many participants trust GPT for routine tasks but remain cautious about its use in critical decisions. This disparity underscores the necessity of continually assessing GPT's reliability and effectiveness in diverse operational contexts.

4.5 RQ4: Overall adoption of ML-based solutions

The current level of readiness towards adopting AI/ML-based solutions is a crucial indicator of how these cutting-edge technologies are currently being integrated into various software environments. The collected survey data, as shown in Table 3, re-

Table 3: Readiness to adopt new AI/ML-based solutions

Organizational role	Not ready or hesitant	Somewhat ready, facing constraints	From cautious to proactive
CloudOps Engineer	0	2	0
DevOps Engineer	4	1	1
Project Manager	0	0	1
Test Manager	0	1	0
Quality Assurance	3	4	3
Software Developer	0	1	4

flects varied levels of readiness among different organizational roles. For instance, a significant number of quality assurance practitioners and software developers indicated cautious readiness or somewhat readiness, facing constraints. This implies a recognition of the potential benefits of AI/ML but with a raised awareness of the challenges and constraints involved. The data also indicates a hesitancy among some roles, particularly DevOps engineers, who showed a mix of hesitation and cautious readiness. These results could be related to potential concerns regarding the technical complexities, integration challenges, or possible disruption to established workflows and processes. Overall, the distribution of answers highlights the need for tailored strategies in AI/ML integration, considering the unique needs and constraints of each role within the software development and operations.

Another significant observation from the survey results shown in Table 4 is the contrast between the current and forecasted impacts of AI/ML-based solutions. A major percentage (56%) of respondents reported no significant impact from recent AI advancements on their development and testing processes. However, there are positive expectations about the future, with an equal proportion of respondents (56%) anticipating a noticeable increase in development cycle speed due to AI/ML solutions. Therefore, while the immediate benefits of AI are not yet widely recognized across organizational units, there is strong optimism in its potential to enhance efficiency and delivery pace in the near future.

Even though practitioners in the majority of organizational units do not perceive the immediate effects of AI/ML tools on the software development life-cycle, a significant percentage (60%) have already started using some of the basic tools, such as chatbots and simple analytics. This captures a typical transitional phase in the adoption of new technology. Industry professionals acknowledge the potential of AI in enhancing software development and operations, as they are keen to explore widely used tools on the market. However, their practical outcomes in software development environments are still not apparent but are enthusiastically awaited.

Table 4: The current and forecasted impact of AI/ML-based tools

Current impact on DevOps	**Answers (%)**
No significant impact	56%
Slightly improved processes	20%
Moderately improved processes	20%
Forecasted impact on delivery pace	**Answers (%)**
Noticeable increase in speed	56%
Minor increase in speed	36%
No effect on pace	4%

5 Discussion and conclusion

In this section, we discuss the main findings of this survey study, focusing on the implications and future directions of each research question.

The diversity of identified operational failures and their occurrences across different organizational units and roles **(RQ1)** highlights the dynamic and complex nature of software development environments. A need for more advanced monitoring solutions is evident to ensure operational resilience and efficiency in such environments. This may include interpretable monitoring strategies that detect issues and provide natural language suggestions for addressing them. This evolution towards systems that can interpret operational failures and propose solutions in a human-readable format represents a significant step forward. It merges the efficiency and precision of technology with the intuitive understanding of human experts, aiming to enhance decision-making and streamline the resolution process in software development environments.

Detailed insights into the use of monitoring data revealed its significance for investigating underlying issues and maintaining software quality **(RQ2)**. The diverse usage of tools like Amazon CloudWatch and Microsoft Azure, favored for their comprehensive features and ease of integration, highlights the need for tailored approaches to meet varied operational demands. Commonly analyzed data includes performance metrics and log files, with operational teams prioritizing system performance monitoring and issue diagnosis. Organizations adopt alert systems of varying sophistication to enhance effectiveness, and there is industry consensus on the value of alert strategies for the early detection of operational failures.

Even though some organizational roles are currently confident in existing monitoring solutions, there is still a growing interest in integrating advanced technologies to enhance alert detection **(RQ3)**. This is crucial for minimizing downtime and resolving issues proactively. Additionally, the potential reduction in manual monitoring efforts through ML suggests an important shift towards automation,

freeing up human resources for more complex tasks. However, the possibilities of ML in monitoring and alerting are approached with caution. There are concerns about integration, costs, and data security, as well as the varied levels of confidence in ML predictions, particularly in critical operational tasks. This indicates a need for further validation and refinement of these ML-enabled systems. The common perspective leans towards a gradual, thoughtful integration of ML, with a focus on balancing innovation with practicality, efficiency with reliability, and automation with human oversight.

Regarding **RQ4**, the findings showed a cautious but growing readiness among different organizational roles towards adopting AI/ML solutions. The varied readiness levels indicate a recognition of the potential benefits and challenges associated with these technologies. The contrast between the current limited impact and the optimistic future outlook for AI/ML in software development suggests an ongoing transition phase. As practitioners start to engage with basic AI/ML tools, there is a growing expectation for these technologies to considerably enhance operational efficiency and development processes in the future. This points to a period of exploration and gradual integration, where the full potential of AI/ML in software environments is yet to be realized.

The collective findings from our survey study offer valuable insights into the evolving dynamics of software development and monitoring, particularly in the context of operational failures, monitoring practices, alert strategies, and the integration of AI/ML technologies. The results revealed a combination of complex challenges and emerging opportunities where organizations increasingly recognize the potential of advanced technologies to enhance efficiency and problem-solving capabilities. Given their source in a survey of selected experts, we do not claim our findings to be generally valid, but rather indicate variation among a set of quite homogeneous companies to be addressed in further research and implementation. The anticipated impact of AI/ML on software development and operations promises a new era of innovation and productivity, considering that the challenges of integration, cost, and data security will be effectively addressed in the near future.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We thank all the industry practitioners who participated in this survey, whose invaluable insights and expertise have greatly contributed to the success and depth of this research.

PAPER V

MONITORING DATA FOR ANOMALY DETECTION IN CLOUD-BASED SYSTEMS: A SYSTEMATIC MAPPING STUDY

Adha Hrusto, Nauman bin Ali, Emelie Engström, Yuqing Wang

Abstract

Anomaly detection is key to maintaining cloud systems, allowing early failure detection. With ongoing advances and complex industrial applications, a review of techniques using real-world operational data is needed. This study aims to complement existing research with an extensive catalog of the techniques and monitoring data used for detecting anomalies affecting the performance or reliability of cloud-based software systems that have been developed and/or evaluated in a real-world context. We perform a systematic mapping study to examine the literature on anomaly detection in cloud-based systems, particularly focusing on the usage of real-world monitoring data. Based on a review of 104 papers, we categorize monitoring data by structure, types, and origins and the tools used for data collection and processing. We offer a comprehensive overview of data preprocessing and anomaly detection techniques mapped to different data categories. Our findings highlight practical challenges and considerations in applying these techniques in real-world cloud environments. The findings help practitioners and researchers identify relevant data categories and select appropriate data preprocessing and anomaly detection techniques for their specific operational environments.

Submitted to TOSEM

1 Introduction

The widespread adoption of cloud-based software solutions in various industries is driven by their scalability, reliability, and cost-effectiveness [75]. However, maintaining the operational health of these systems presents significant challenges [39]. Detecting anomalies in system behavior is a critical issue, as these deviations from normal operational patterns often serve as early indicators of underlying problems that could compromise the reliability and performance of cloud-based systems [244]. Thus, anomaly detection (AD) plays a crucial role in maintaining the operational health of systems deployed in cloud environments.

In practical scenarios where cloud systems operate under diverse conditions and varying workloads, monitoring data is essential for understanding system behavior and creating robust anomaly detection strategies. Such data gathered from cloud operations provides critical insight into the dependencies and interactions between hardware and software components. This information is crucial in root cause analysis and proactive fault management frameworks for cloud applications [220]. Furthermore, the use of operational data in the development and evaluation of AD techniques indicates the likely practical applicability of such techniques in real cloud environments. Therefore, leveraging data from cloud operations is critical for developing more resilient and fault-tolerant cloud systems [212].

Existing literature studies that aggregate anomaly detection research have often not made a distinction between synthetic datasets and monitoring data from real-world systems. However, synthetic data or data from lab deployments does not accurately reflect the complexities and nuances of real-world cloud environments. Similarly, the existing literature often falls short of providing comprehensive insights into the application of anomaly detection using data collected from cloud operations, leading to a gap in practical, empirical evidence. Instead, existing reviews on anomaly detection focus on other and specific aspects of cloud computing systems, such as cloud infrastructure, or on characteristics of detection techniques without paying attention to the operational context or data [22, 32, 36, 66, 75, 100, 156, 161, 223].

To address this gap, we explore and synthesize the data used for anomaly detection with a particular focus on its application and evaluation in cloud-based systems. We conducted an extensive systematic mapping study, focusing on different data characteristics and corresponding techniques for processing the data and detecting anomalies.

Based on information extracted from the reviewed papers, we provide catalogs of data based on type, structure, and origin; tools used for monitoring; and data preprocessing and anomaly detection techniques. We further provide reflections on the contexts in which these techniques have been applied. Our catalogs are designed to aid practitioners in prioritizing relevant operational data and developing effective anomaly detection strategies tailored to their specific contexts. By using

monitoring data from real industrial settings, anomaly detection techniques can be trained and tested on datasets that accurately reflect the complexities and dynamics of cloud-based software systems. Highlighting the practical aspects of anomaly detection in cloud environments, we aim to contribute to the body of knowledge with actionable insights that can be readily implemented in real-world scenarios.

This paper makes several key contributions to the field of anomaly detection in cloud-based systems:

- **C1: A catalog of monitoring data collected from operational environments.** This classification helps in understanding the diverse nature of monitoring data, their properties, and their relevance to anomaly detection in real-world cloud environments.
- **C2: An overview of monitoring tools.** We compile a list of monitoring tools used in various cloud environments for collecting and storing monitoring data while discussing their strengths and limitations.
- **C3: Comprehensive overview of data preprocessing and anomaly detection techniques.** This contribution is a valuable resource for practitioners seeking to enhance their anomaly detection in cloud environments. Additionally, we provide a detailed mapping of anomaly detection techniques to various monitoring data categories, aiding in selecting suitable techniques based on data characteristics.
- **C4: Analysis of real-word aspect.** This analysis reveals which anomaly detection techniques are directly applicable and relevant to industry practices and real-world challenges.

By addressing these areas, we aim to provide a solid foundation for the development and implementation of effective anomaly detection strategies in real-world cloud environments. By focusing on real-world data and providing comprehensive catalogs, we offer valuable guidance to practitioners seeking to enhance their anomaly detection capabilities.

2 Background and Related Work

In this section, we briefly introduce the task of anomaly detection and its importance in cloud-based systems. In addition, we provide an overview of existing reviews on anomaly detection techniques applied in these environments and outline our unique contributions to the field.

2.1 Anomaly detection

In the field of software engineering, anomaly detection is the process of identifying deviations from the expected or normal behavior within a software system [21]. It

involves continuously analyzing monitoring data, which provides observability to identify potential anomalies that require further investigation or action. Monitoring data exist in different modalities, such as logs, metrics, and traces [63].

Cloud-based systems generate vast amounts of monitoring data from various components, such as servers, services, databases, load balancers, and virtual machines [1]. Anomalies can occur in any of these components. Manually analyzing monitoring data from different components to detect anomalies for the cloud-based system can be time-consuming and error-prone. Modern anomaly detection leverages automation techniques to continuously monitor and analyze monitoring data from various components [1, 75]. Automated anomaly detection can help to quickly identify and remediate potential anomalies, optimize system performance, and reduce operational overhead [21].

2.2 Existing reviews

We systematically searched[1] for related literature reviews or mapping studies on anomaly detection in cloud-based systems. We identified seven relevant papers [32, 36, 66, 75, 100, 161, 223].

An overview of these reviews on anomaly detection in cloud-based systems is presented in Table 1 and Table 2. These reviews focus primarily on the underlying cloud infrastructure and analysis of its characteristics, challenges, or most relevant topics with regard to cloud monitoring or management. Only three reviews [36, 75, 100] cover anomaly detection in cloud computing environments. Hagemann and Katsarou focus their review on anomaly detection techniques in cloud environments. Cândido et al. identify relevant research with regards to software monitoring in general and introduce anomaly detection as one of the application areas for log analysis. The most recent review by Jayaweera et al. [100] identified only 21 relevant papers published between 2017 and 2023, despite employing an extensive search string (see Table1).

These reviews present an overview of anomaly detection techniques and their areas of application, but they offer little to no attention to the context in which these techniques were applied. Furthermore, none of the identified literature reviews specifically focus on the types of data used and how these data are mapped to the reviewed anomaly detection techniques, which is the primary goal of our review.

2.3 Our contribution

We identify the need for a review of publications that report case studies concerning data and techniques for anomaly detection in cloud-based software systems.

[1] We used the following search string to identify potentially relevant literature reviews on anomaly detection in cloud-based systems. *TITLE-ABS-KEY ("anomaly detection" OR "monitoring")* ***AND*** *TITLE-ABS-KEY ("cloud" OR "microservices" OR "software system*")* ***AND*** *TITLE-ABS-KEY ("systematic" AND ("mapping" OR "review" OR "map"))* . The approach for the construction of the search string and identifying relevant reviews is described in Appendix 1

Table 1: Overview of related literature reviews

Paper	Aim	Context	Search period	Search strings
Costa et al. [32]	Taxonomy design	Edge/fog computing	2012-2022	(Fog OR Edge) AND (Monitor* OR Observability)
Hagemann and Katsarou [75]	Comprehensive overview of anomaly detection techniques and areas of application in cloud computing infrastructures	Anomaly detection for cloud computing environment	2010–2020	("Cloud Computing" OR "Cloud Monitoring") AND "Anomaly Detection"
Odun-Ayo et al. [161]	A visual map of publications in cloud monitoring and management considering the topics, contributions, and research type	Cloud management and monitoring	2004–2020	(TITLE(“CLOUD management”) OR TITLE(service level agreement “) OR TITLE(“SLA”) AND (TITLE(adaptive) OR TITLE(monitoring) OR TITLE(autonomic) AND KEY(CLOUD) OR KEY(SLA)
Gill and Hevary [66]	Identify and synthesize the challenges of virtual CPU utilization monitoring data	Cloud computing environment	2008–2016	((“virtual processor”) OR (“virtual CPU”)) AND ((“monitoring tool*”) OR (“monitoring technique*”)) and ((“cloud computing*”));
Cândido et al. [36]	Identify relevant research, categorize and summarize key research results in log-based software monitoring	DevOps, modern software and operations environments	1992–2019	log AND (trace OR event OR software OR system OR code OR detect OR mining OR analysis OR monitoring OR web OR technique OR develop OR pattern OR practice)
Wang et al. [223]	(1) classify existing research based on application fields and technology (2) describe the current state of research in terms of load balancing, fault detection and auto-scaling	Microservice architecture systems	2009–2021	No search strings are reported; Some of the used keywords: Fault monitor; Distributed monitor; Fault diagnosis; Fault localization; Cloud computing; Microservice monitor;
Jayaweera et al. [100]	Identify machine learning approaches for anomaly detection using network data	Cloud environment	2017–2023	(“anomaly” OR “ irregularity” OR “abnormality” OR “deviation” OR “malformation”) AND (“detect” OR “ monitor” OR “recognition” OR “uncovering”) AND (“cloud”)

Table 2: Overview of related literature reviews

Paper	Databases	Selection criteria	No. of included papers	Main results
Costa et al. [32]	-	Works that present solutions, architectural models, techniques or methods applied to monitoring in fog computing	-	Taxonomy of a fog monitoring solution
Hagemann and Katsarou [75]	SpringerLink, Web Of Science (WoB), open-access archive ArXiv	Include works that belong to a relevant field such as computer science or engineering; Anomaly detection should be focus of the work;	216	Three main clusters of anomaly detection methods (machine learning, deep learning, and statistical methods); Three main areas of application (intrusion detection, performance monitoring, failure detection)
Odun-Ayo et al. [161]	IEEE Xplore, Springer, Science Direct, ACM, Scopus	The abstract explicitly mentions management and monitoring; Abstracts that relates to SLA and autonomic;	105 – topic and contribution category; 136 – topic and research type category	5 topics (SLA monitoring, Security, Autonomous management, Self-adaptive SLA, Architectures) mapped across contributions of primary studies (metric, tool, model, method, process) and research types(evaluation, validation, solution, philosophical, experience, opinion)
Gill and Hevary [66]	Web of Science, IEEE, Google Scholar, Gartner, Scopus	1) Title == search keywords 2) Abstract== CPU or Virtual CPU or VCPU 3) Text CONTAINS Virtual CPU OR Monitoring Tool or Cloud Computing	24	Five major challenges of cloud monitoring data identified: monitoring technology, virtualization technology, energy, availability and performance.
Cândido et al. [36]	ACM Digital Library, IEEE Xplore, SpringerLink, Scopus and Google Scholar	C1: It is an English manuscript; C2: It is a primary study; C3: Peer-reviewed paper; C4: The paper uses the term "log" in a software engineering context;	108	Publication trends in research on log-based monitoring over the years; overview of research copes in the life-cycle of log data (logging, log infrastructure and log analysis)
Wang et al. [223]	IEEE Xplore, ACM Digital Library, Springer-Link, Web of Science, Google Scholar	Related papers of SOA, distribution and cloud computing; Related papers of operation and maintenance governance;	144 in abstract 139 in intro	Main research directions for research in the operations and maintenance governance of microservices architecture systems are: 1) load balancing, (2) fault detection, and (3) autoscaling
Jayaweera et al. [100]	IEEE Xplore, ACM Digital Library, Springer-Link, Science Direct, Research Gate	Papers about the use of machine learning for anomaly detection in the cloud environment;	21	Lists five categories of techniques and 11 datasets used in anomaly detection research

Hagemann and Katsarou [75] take a rather broader overview of all anomaly detection techniques in cloud computing or monitoring. We aim to complement this review, broadening the scope to specify the type of systems and context for which anomaly detection techniques may be applied, but also looking deeper into the importance of such techniques for improving the performance characteristics of the software in operations. Moreover, our focus is on monitoring data as a source of valuable information regarding failures, performance issues, or anything that could degrade the reliability of large-scale software systems in cloud operations. We also aim for an extensive overview of anomaly detection techniques applied to different categories of monitoring data. In addition, we include an analysis of the tools for managing data and a discussion on the techniques evaluated with data representative of real-world scenarios.

3 Research Methodology

We conducted a systematic mapping study following the guidelines proposed by Kitchenham and Chaters [116]. The aim of this mapping study is to present an overview of the published literature reporting applications of anomaly detection, particularly using real-world data collected through the monitoring of cloud-based software systems in real industrial contexts.

3.1 Research questions

The following research questions outline our study on the use of monitoring data for anomaly detection in cloud-based software systems:

- **RQ1:** What are the main categories of monitoring data in terms of structure, types, and origin used for anomaly detection in the operational environment of cloud-based software systems?
- **RQ2:** Which monitoring tools are used to collect, store, and process monitoring data?
- **RQ3:** What types of data preprocessing and anomaly detection techniques are used in different monitoring data categories?
- **RQ4:** Which anomaly detection techniques have been evaluated using data representative of real-world cloud environments?

3.2 Data sources and search strategy

The selection of databases can significantly influence the scope and depth of a literature review. Kitchenham and Chaters [116] recommend using a comprehensive set of digital libraries, including IEEE Explore, ACM Digital Library, Scopus,

SpringerLink, and ScienceDirect, to ensure a wide coverage of relevant academic literature in systematic reviews. In our context, Scopus emerges as a particularly effective digital library due to its extensive coverage of peer-reviewed literature across diverse publishers. In a comparative analysis conducted by Valente et al. [215], findings suggested that the inclusion of IEEE Explore alongside Scopus did not significantly improve the coverage of the literature, as IEEE Xplore content was already included in Scopus. This redundancy indicates that Scopus alone may be sufficient for comprehensive literature searches, especially in fields well represented in its database, such as computer science. Furthermore, Scopus is acknowledged for its broader interdisciplinary coverage and updates, which include journals and conference proceedings from both ACM and IEEE, making it a singularly efficient source in the context of maximizing both the breadth and depth of literature exploration without compromising the quality or relevance of selected studies [51]. Given these attributes, we relied solely on Scopus for this systematic mapping study.

Regarding the search strategy, we started with trial searches, while preliminary search strings were derived from research questions with the following features:

- **Task:** Monitoring and anomaly detection. *Keywords:* anomaly detection, outlier detection, cloud monitoring, runtime monitoring, or software monitoring.
- **Data:** Related to quality attributes. *Keywords:* performance or reliability.
- **System type:** Software systems deployed and operating in the cloud. Keywords: cloud system, cloud application, cloud platform, cloud computing, edge computing, fog computing, microservice, or software system.
- **Context:** Real cloud and software environments. *Keywords:* real-world or case study.

The keywords within each feature category were joined using the Boolean operator *OR*, and the four categories were joined using the Boolean operator *AND*. Four different variations of preliminary search strings (PSS1-PSS4) were created as shown in Table 3, combining keywords from different categories to examine the search scope from very broad to narrow. To objectively validate the effectiveness of these four search strings, we relied on a Quasi-Gold Standard [42].

3.3 Quasi-Gold Standard (QGS)

A quasi-gold standard [42] refers to a curated benchmark collection of papers known to be highly relevant to a specific research question or topic. The "quasi" aspect acknowledges that while the QGS is carefully selected to be as comprehensive and relevant as possible, it may not be exhaustive or perfectly representative

Table 3: Overview of search strings (PSS2 was initially selected, while FSS is the final search string obtained after refining PSS2)

PSS1	TITLE-ABS-KEY(("anomaly detection" OR "outlier detection" OR "monitoring") **AND** ("reliability" OR "performance") **AND** ("cloud" OR "edge computing" OR "fog computing" OR "microservice*" OR "software system*"))
PSS2	TITLE-ABS-KEY(("anomaly detection" OR "outlier detection" OR "cloud monitoring" OR "runtime monitoring" OR "software monitoring") **AND** ("reliability" OR "performance") **AND** ("cloud system*" OR "cloud application*" OR "cloud platform*" OR "cloud computing" OR "microservice*" OR "software system*"))
PSS3	TITLE-ABS-KEY(("anomaly detection" OR "outlier detection" OR "cloud monitoring" OR "runtime monitoring" OR "software monitoring") **AND** ("reliability" OR "performance") **AND** ("cloud system*" OR "cloud application*" OR "cloud platform*" OR "cloud computing" OR "microservice*" OR "software system*") **AND** ("real world" OR "case study")
PSS4	TITLE("anomaly detection" OR "outlier detection" OR "cloud monitoring" OR "runtime monitoring" OR "software monitoring") **AND** TITLE-ABS-KEY("reliability" OR "performance") **AND** TITLE("cloud system*" OR "cloud application*" OR "cloud platform*" OR "cloud computing" OR "microservice*" OR "software system*")
FSS	TITLE-ABS-KEY("anomal*" OR "outlier*" OR "cloud monitoring" OR "runtime monitoring" OR "software monitoring") **AND** TITLE-ABS-KEY("reliability" OR "performance" OR "health" OR "failure*") **AND** TITLE-ABS-KEY("cloud system*" OR "cloud application*" OR "cloud platform*" OR "computing platform*" OR "cloud computing" OR "microservice*" OR "software system*")

of all possible relevant literature due to inherent limitations such as the subjective selection of papers or incomplete database indexing.

To develop the Quasi-Gold Standard (QGS) for this study, we systematically analyzed references from the systematic review by Hagemann and Katsarou [75], which included 216 references. Furthermore, we analyzed ten papers and their 62 citations (February 2023), which were selected based on our prior knowledge and expertise in the field. After the exclusion of duplicates and incomplete references, a pool of 237 papers was formed. The titles and abstracts of these papers were reviewed considering our selection criteria presented in Section 3.5. The relevance of each article was assessed by at least two authors. We had an average inter-rater reliability of 83.55%. The rigorous selection process resulted in forming a QGS comprised of 36 papers, which was intended to serve as a benchmark for evaluating the effectiveness of our search strategies.

3.4 Selection of the search string

An overview of the steps in the process of selecting the search string is illustrated in Figure 1. The left part of Figure 1 shows the selection of the QGS, while the right part corresponds to the evaluation of search string proposals with this established benchmark of relevant papers. Thus, we designed and tested four preliminary search strings (PSS1-PSS4, see Table 3) to explore the effectiveness of each search strategy in capturing the key literature included in the QGS.

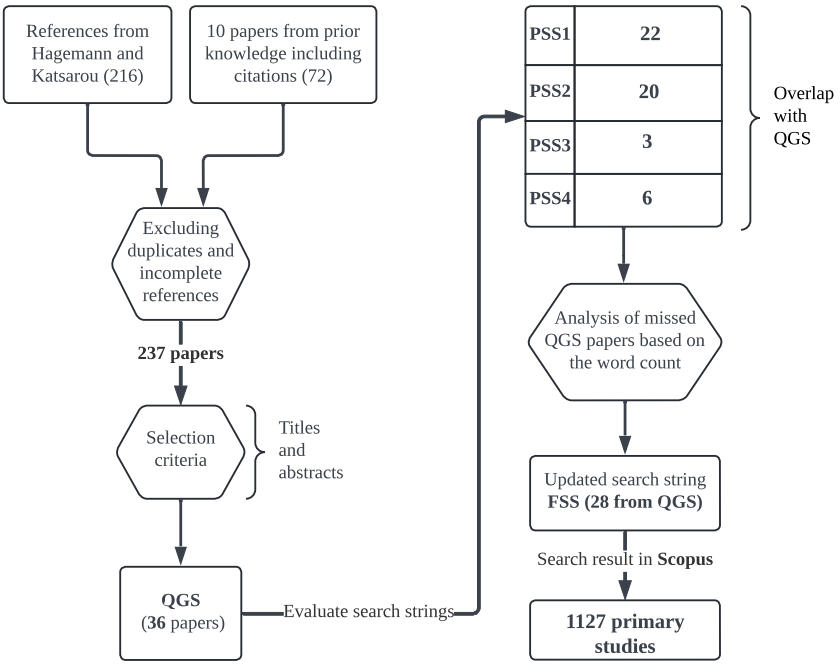


Figure 1: Overview of the steps in the selection of the search string (PSS1-4 represent preliminary search strings while FSS is a final search string)

The results of this testing phase indicated different levels of overlap between the retrieved documents and the QGS. Specifically, PSS1 demonstrated the highest level of overlap, retrieving 22 relevant papers from 6021 hits. This was closely followed by PSS2, which identified 20 relevant papers from 501 hits. In contrast, PSS4, with 49 hits, retrieved 6 relevant papers, while PSS3, with 61 hits, identified the fewest relevant papers, only 3. We decided to proceed with PSS2 due to its highest balance between precision and recall. While PSS1 retrieved a slightly higher number of relevant papers, it produced an excessively large number of hits, making it less efficient and more cumbersome for manual review. PSS2, with its higher precision and manageable number of hits, ensured comprehensive coverage of relevant literature without overwhelming the review process.

To further improve the search string PSS2, we performed a detailed analysis of the papers that were missed by the initial searches, particularly examining the most frequent keywords. This led to minor updates in the search string PSS2 and creating the final search string FSS (see Table 3). We identified two additional keywords related to data and quality attributes (*failure and health*) that we decided to include in the final search string FSS to achieve a more comprehensive search

mechanism. This updated search string was then used to conduct a search in the Scopus database, which resulted in identifying **1127** potentially relevant primary studies, including 28 papers from QGS. This iterative refinement of the search strings, guided by the insights from the QGS, helped us to maximize both the coverage and relevance of the identified literature.

The following is the resulting search string FSS:
(TITLE-ABS-KEY("anomal" OR "outlier*" OR "cloud monitoring" OR "runtime monitoring" OR "software monitoring")*
***AND** TITLE-ABS-KEY("reliability" OR "performance" OR "health" OR "failure*")*
***AND** TITLE-ABS-KEY("cloud system*" OR "cloud application*" OR "cloud platform*" OR "computing platform*" OR "cloud computing" OR "microservice*" OR "software system*"))*

The proximity operator was used to enhance the relevance of search results while also allowing variations in terminology used by different authors to describe specific features associated with the categories outlined in Section 3.2. This approach improves the precision of our search strategy, increasing the likelihood of retrieving contextually related documents that are more relevant to our research focus.

3.5 Study selection criteria and procedure

The selection of primary studies was carried out in two phases:

First phase. In this phase, we screened the titles and abstracts of the papers to check for relevance. We divided the papers among four reviewers (1st – 4th author) and filtered them according to our exclusion criteria, which are presented below. Note that 51 papers were excluded from this phase as 28 belong to QGS, and 23 had been previously identified as irrelevant during the QGS selection process. Before applying the exclusion criteria specific to our review, all search results were initially filtered through a search string FSS that incorporated generic exclusion criteria, including:

- Publications in non-peer-reviewed venues such as books, master's or Ph.D. theses, keynotes, tutorials, and editorials.
- Documents not available in English.
- Publications where research contributions belong to fields other than computer science or software engineering.

Or translated to Scopus sytnax: *(LIMIT-TO (DOCTYPE , "ar") OR LIMIT-TO(DOCTYPE , "cp"))* ***AND*** *(LIMIT-TO(LANGUAGE , "English"))* ***AND*** *(LIMIT-TO(SUBJAREA , "COMP") OR LIMIT-TO(SUBJAREA , "ENGI"))*

Thus, the exclusion criteria were applied to 1076 primary studies after generic exclusion. The exclusion criteria are as follows:

- **Duplicates.** The paper is a duplicate. In this first phase, we did not treat the subsequent publication of a conference paper as a journal article as a duplicate. However, this aspect was addressed later during the full-text review of the articles. Excluding duplicates is important to avoid double-counting of evidence [16].
- **Task.** The main focus of the paper is not on monitoring sub-tasks, primarily on *anomaly detection* or similar.
- **Quality attribute.** The anomaly detection is not focused on the performance or reliability aspects of the cloud-based software system.
- **System.** The research focus is not on a cloud-based software system (at the application, platform, or infrastructure level). Papers focusing only on data centers or public clouds in general without clear reference to a software system under study were disregarded.
- **Data.** The study does not utilize monitoring data derived from real-world environments. Monitoring data refers to metrics, logs, traces, or other data types actively collected through monitoring or passively gathered, such as archived data.

To ensure a rigorous and efficient review process, we implemented a streamlined single review approach, as justified by the very high inter-rater reliability discussed in Section 3.3. In this process, each paper was labeled as relevant, unclear, or irrelevant. Following the initial phase of screening titles and abstracts, 111 papers were identified as relevant, 104 as unclear, and 861 as irrelevant. Based on a cost-benefit analysis, we decided to focus solely on the **111 relevant papers** for further selection and data extraction.

Second Phase. The final selection of papers was based on the full-text review of papers identified as relevant during Phase 1 (111) applying the same exclusion criteria. To streamline the review process, this full-text assessment was conducted concurrently with data extraction. The final dataset comprises **104 papers**, which includes 24 papers from the Quasi-Gold Standard (after removing 4 duplicates from the original 28 papers).

3.6 Data extraction

We used a random sample of five papers from selected primary studies to design the draft of the data extraction form. The first author proposed an initial version of the data extraction form based on the four research questions, utilizing a shared spreadsheet. Then, all authors reviewed and piloted this initial version during data extraction from the five sample papers. Throughout this process, the form was incrementally developed and improved through several consecutive meetings. The spreadsheet integrates a classification scheme, where the first column lists all

relevant papers, and subsequent columns are dedicated to research questions and corresponding categories [2].

Specifically, the first research question (**RQ1**) focuses on extracting types of monitoring data and relevant properties in the context of anomaly detection for cloud-based software systems. Our primary emphasis is on monitoring data utilized for the evaluation of proposed anomaly detection techniques, as these data typically represent real-world usage scenarios. For RQ1, we capture several key data properties: the *entity* (e.g., infrastructure, cloud application), *quality attributes* (e.g., performance), *base measures* (e.g., CPU time, response time), and the *data's source* (e.g., metrics, logs, traces). Additionally, we classify the data structure as either structured or unstructured, noting structured data's organized nature, often stored in databases, versus unstructured data's more complex, formatless nature. We also examine the origin of the monitoring data and labels, distinguishing between *primary data* actively collected for the study and *secondary data*, which includes previously collected data sets used for evaluation.

Subsequent research questions expand on this extraction framework. **RQ2** specifically targets the tools used in monitoring operational data critical to anomaly detection in cloud-based systems. **RQ3** expands further into the technical aspects of anomaly detection by focusing on the *data processing techniques* and the specific *anomaly detection algorithms* employed. This question explores the entire data pipeline from initial data collection to the final detection of anomalies. We extract detailed information about data preprocessing steps, such as cleaning, normalization, and augmentation, which are critical to ensuring data quality and reliability before applying detection techniques. In addition, we catalog the anomaly detection techniques associating them with various data categories. With **RQ4**, we assess the *representativeness of real-world scenarios* in monitoring data and anomaly labels, noting whether data or labels are more representative when sourced from operational environments or less so when derived from simulated conditions or public archived datasets. Furthermore, we use this information to argue which of the anomaly detection techniques are more or less applicable to real-world settings.

3.7 Data analysis

To further streamline the data analysis process based on the extracted data, we performed a standardization across all the papers included in the review process. This standardization involved identifying general categories for the data in each column, tagging the data accordingly, and augmenting the data extraction form to include additional columns for these categories. Each of these new columns represents a standardized category to which a paper can be mapped. We marked the mapping between a paper and a category with an 'X'. This systematic structuring allows for the straightforward aggregation and quantification of data belonging to

[2] Data extraction form

each category. Note that not all of these categorizations are mutually exclusive, which means that a paper may report several types of results.

3.8 Threats to validity

This section discusses potential threats to the validity of our systematic mapping study.

Internal validity. One potential threat to internal validity is the bias introduced during the selection of primary studies. While we employed a rigorous selection process involving multiple reviewers and resolving disagreements through discussions, the analysis of extracted data, as explained in Section 3.7, was mainly reviewed by one author. This could introduce personal biases and inconsistencies. To mitigate this, we attempted to align the data extraction process with the specific backgrounds of the authors, leveraging their expertise to ensure more accurate and relevant categorizations. However, this approach may still lead to inconsistencies due to subjective interpretations. In addition, out of the 1127 search hits, we excluded 104 papers because their relevance was unclear (after reading the titles and abstracts of the papers), which might have led to missing some important studies in our review. Our decision was made to ensure the review process remained manageable as we had identified a large number of papers (i.e., 111 papers) that we considered relevant according to our selection criteria. Given the large sample size, we consider that the threat of missing relevant papers among the 104 excluded papers does not risk the conclusions of this mapping study.

Construct validity. Given the vast scope of cloud-based systems and the rapid evolution of technologies and methodologies, there is a risk that some relevant studies or important aspects of anomaly detection may not have been fully captured within the final search string. Additionally, variations in the quality of reporting in primary studies may have affected how well we identified and categorized the constructs. Inconsistent or incomplete reporting could lead to misinterpretation of the data or failure to capture all relevant aspects of the measured construct. To address this, we carefully reviewed and cross-validated the information from the primary studies during data analysis whenever we encountered doubts.

External validity. The generalizability of our findings may be limited by the specific focus on cloud-based systems. Although our study encompasses a wide range of cloud environments and operational contexts, the results may not be directly applicable to other domains, such as hybrid cloud environments. Nevertheless, the principles and catalogs developed in this study provide a foundation that can be adapted and extended to other areas.

4 Results and Analysis

In this section, we present the results of our systematic mapping study on anomaly detection in cloud computing systems, ensuring a thorough overview of the current

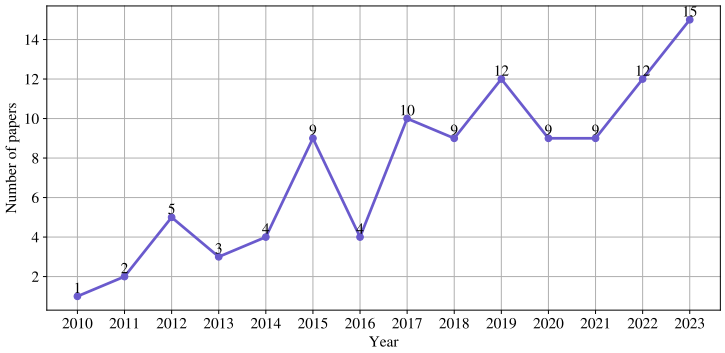


Figure 2: Distribution of publications over the years

state of research and highlighting significant areas of interest and ongoing development. We begin by analyzing publication years to identify research trends and review the venues of these studies, including conferences and journals. Next, we explore the study context, covering various system types and architectures. Finally, we address the results related to each of our research questions, offering in-depth analysis regarding the main findings from 104 papers included in our review.

4.1 Publication years

Figure 2 presents a line graph illustrating the distribution of primary studies on anomaly detection in cloud computing systems published over a span of years from 2010 to 2023. The data reveal a clear upward trend in the number of studies published, reflecting growing interest and increasing research activity in this field. Interestingly, the earliest publication in the reviewed set of papers dates back to 2010. This may indicate that, while the broader field of anomaly detection has existed for longer, its application, specifically within cloud computing systems, began to gain attention around this time. This specific starting point marks a period when cloud computing was becoming more prevalent, requiring specialized research in anomaly detection to address unique challenges within cloud environments. Over the years, there has been a significant increase in publications, with notable peaks in 2019 and 2023, where the number of published studies reached 12 and 15, respectively. The increase in publications highlights a growing interest in anomaly detection in cloud environments, indicating an expanding and important research field.

4.2 Publication venues

Additionally, the distribution of publications across leading conferences and journals further supports this upward trend, as shown in Table 4. Among the 104 papers reviewed, 37 were published in journals and 67 in conferences. Table 4 lists venues with at least two publications, while there is a total of 79 unique venues, which shows the diversity and broad interest in this research area. The IEEE International Conference on Cloud Computing (CLOUD) stands out with *eight publications*, highlighting its significance as a leading platform for research dissemination. Other notable conferences with multiple publications are closely related to cloud computing and software engineering research areas. With regard to journals, four stand out, each with three publications. The spread of studies across numerous prestigious venues indicates a well-established and rapidly growing research community focused on tackling anomalies in cloud computing environments.

Table 4: An overview of the leading publication venues based on included papers

Venue type	Venue name	Count
Conference	IEEE International Conference on Cloud Computing (CLOUD)	8
	IEEE International Symposium on Software Reliability Engineering (ISSRE)	3
	IEEE International Conference on Cloud Computing Technology and Science (CloudCom)	2
	USENIX Annual Technical Conference (USENIX ATC)	2
	International Conference on Software Engineering (ICSE)	2
	IFIP/IEEE International Symposium on Integrated Network Management (IM)	2
	International Conference on Cloud Computing and Services Science	2
	The ACM Web Conference	2
Journal	Journal of Systems and Software	3
	IEEE Transactions on Network and Service Management	3
	Future Generation Computer Systems	3
	Concurrency and Computation: Practice and Experience	3
	Journal of Cloud Computing	2
	Software: Practice and Experience	2

4.3 System context

All included primary studies are framed within the overarching context of cloud computing, as shown in Table 5. However, within this broad context, some studies focused particularly on specific layers or architectures, offering deeper insights and solutions for a narrower scope. For example, multiple papers specifically address the cloud application layer and enhance application performance, scalability, and reliability in order to adapt to varying workloads and user demands. On the other hand, a larger number of studies concentrate on the cloud infrastructure layer, which serves as the backbone for cloud services and applications. These papers examine the core elements that ensure the robust and efficient operation of cloud infrastructure to support increasingly complex and demanding cloud applications.

Moreover, according to Table 5, there are various types and architectures within the cloud computing ecosystem. However, the included papers are mainly focused

on microservice architectures. This attention to microservices may reflect a shift towards modular and flexible service design, which enables the development of more agile and resilient cloud applications.

Table 5: Overview of different system contexts (*Note that some papers reported only a general context, while others report instantiating studies in specific single or multiple contexts*)

General	Layer		Type/Architecture			
Cloud Computing Systems	Cloud Application	Cloud Infrastructure	Virtualized systems	Large-scale complex systems	Microservice	Distributed
Count of papers	25	38	6	6	24	4

4.4 RQ1: Main categories of monitoring data

Observability in cloud-based software systems is essential to ensure the reliability, performance, and overall health of applications deployed in distributed and dynamic cloud environments. According to Kosińska et al. [118], observability is defined as the ability to accurately capture, analyze, and present information on the performance of a cloud system. A key component of observability is **monitoring data**, which serves as the foundation for understanding the behavior of complex, multi-layered cloud systems, including infrastructure resources and the applications they support [205]. Monitoring data provides direct insights into system operations, enabling proactive monitoring and timely intervention in case of severe disruptions. Thus, by systematically collecting, organizing, and analyzing monitoring data, organizations can achieve a higher level of operational resilience in their cloud applications. We present a detailed overview of monitoring data used for anomaly detection, including the structure, types, and origins as identified in the reviewed papers.

Structure

The structure of monitoring data refers to the way in which input data is organized and formatted. It impacts how efficiently monitoring data can be processed and utilized for observability purposes [118].

Structured data is typically well-defined and easily searchable, often organized in tabular formats or databases. Moreover, JSON (JavaScript Object Notation) is frequently used for structured data (mainly traces) due to its clear and organized key-value pair format. This type of data allows for straightforward searching, filtering, and aggregation, facilitating quick and efficient analysis. We found that approximately 92% (96/104) of papers reported that data was structured. This

indicates that most anomaly detection efforts rely on this data, likely because of its ease of use and the straightforward nature of the analysis it enables.

In contrast, 12% (13/104) of papers used unstructured data, which requires more advanced techniques for effective processing and use. This smaller proportion reflects the additional complexity and effort required to analyze unstructured data. Despite being less prevalent, unstructured data remains crucial for comprehensive observability, as it can provide deeper insights into system behavior and interactions that structured data alone cannot capture.

Types

A comprehensive catalog of monitoring data is crucial for evaluating various aspects of performance and reliability in cloud-based systems. The classification of monitoring data into metrics, logs, and traces is presented in Tables 6 and 7, together with references to the papers that utilized these data types for anomaly detection. These types of monitoring data encompass various base measures collected to evaluate different aspects of system performance and reliability. These include:

- **Metrics:** Quantitative measures that provide insights into system performance are reported in 76% (79/104) of papers. They include: 1) *system data* like CPU statistics, disk usage, and memory usage; 2) *infrastructure data* such as Kubernetes and container metrics; 3) *application/service data* like response times and service faults; 4) *network data* encompassing web traffic flows and latency across network layers; 5) *database data* covering read/write throughput and MySQL metrics. The prevalence of *system, application, and network data metrics* can be attributed to their critical roles in ensuring system health. System data metrics maintain infrastructure stability and capacity by monitoring hardware and operating systems. Infrastructure metrics play a key role in managing the underlying platforms that run applications, supporting container orchestration, scaling, and cloud resource optimization. Further, application data metrics ensure high-quality user experiences by tracking service performance, and network data metrics ensure seamless communication and data flow between system components, which is vital for the efficiency and reliability of distributed cloud environments.

- **Logs:** Records of events that occur within the system, which can be used to trace the sequence of actions leading up to an issue. System logs and application/service logs, identified in 18% (19/104) of papers, provide a comprehensive view of operations and are vital for diagnosing issues.

- **Traces:** Detailed records of the flow of requests through the system, useful for identifying bottlenecks and understanding system interactions. Traces reported in 16% (17/104) of papers, capture the execution paths of requests

Table 6: A catalog of monitoring data for evaluating various aspects of performance and reliability in cloud-based systems

	Types of monitoring data in cloud-based software systems		Percentage
Metrics	**System data**	CPU Statistics Disk Usage Memory usage	**76%**
	Infrastructure data	Kubernetes metrics Container metrics Resource metrics	
	Application/Service data	Response time: Service faults (e.g., number of HTTP errors or failed requests) Processing time Application latency	
	Network data	Traffic metrics RPC execution times Web traffic metrics API Response Time and Throughput Communication latency	
	Database data	DB Read/Write Throughput MySQL Metrics	
Logs	**System data**	System logs	**18%**
	Application/Service data	Executions logs and log events	
Traces	Structured data capturing the execution paths of requests through the system's components		**16%**

through the system's components, offering a structured view of how different parts of the system interact.

Interestingly, 5% of the papers did not specify the types of monitoring data used, suggesting the need for a more comprehensive and standardized approach to documenting monitoring data in research.

Origin

The origin of monitoring data refers to the source and method of data collection. Data can be actively collected from the operational environment through instrumentation and monitoring tools, or passively obtained from publicly available datasets. Understanding the origin of monitoring data is crucial for assessing its accuracy and reliability. Active data collection provides real-time insights, whereas passive data may offer historical context.

Table 7: Types of monitoring data mapped to corresponding papers

Types of monitoring data		Papers
Metrics	System data	[206], [72], [231], [212], [149], [154], [121], [78], [68], [157], [35], [245], [147], [43], [197], [69], [81], [76], [41], [234], [216], [183], [126], [169], [198], [167], [24], [111], [179], [86], [227], [94], [113], [248], [40], [25], [232], [228], [194], [71], [88], [180], [73], [176], [246], [99], [188], [38], [85], [186], [201], [2], [236], [185], [47], [5], [187], [95], [221], [247], [224], [165], [150], [13], [70], [84], [52]
	Infrastructure data	[212], [149], [24], [179], [233], [99], [5]
	Application/Service data	[149], [137], [121], [157], [147], [160], [81], [76], [240], [41], [234], [216], [183], [126], [169], [235], [129], [142], [225], [9], [180], [107], [73], [140], [188], [226], [151], [196], [166], [101], [23], [150], [13], [34], [152], [117], [98]
	Network data	[206], [72], [149], [213], [68], [74], [35], [245], [147], [43], [197], [81], [183], [126], [169], [198], [167], [24], [111], [227], [40], [25], [232], [228], [194], [71], [57], [58], [222], [73], [176], [186], [201], [47], [187], [221], [224], [165], [70], [52]
	Database data	[188], [95]
Logs		[127], [137], [78], [102], [76], [126], [169], [179], [172], [113], [106], [99], [166], [201], [2], [5], [101], [52], [152]
Traces		[213], [160], [240], [129], [24], [233], [250], [9], [180], [140], [226], [151], [196], [23], [150], [34], [117]

Table 8 presents a catalog of the origin of data monitoring in cloud-based systems, categorizing them into primary and secondary data sources. Primary data, corresponding to active collection methods, includes predominantly data from production systems, research and experimental systems, and benchmarking systems. Production systems are a significant source, providing real-world data from companies like IBM [206] and Google [231]. These datasets are essential for tuning performance and diagnosing failures in actual operational settings. Research and experimental systems, including lab environments and testbeds, offer controlled conditions for experimenting and validating new techniques, as seen in studies by Jia et al. [101] and Liu et al. [140]. Benchmarking systems, such as Acme-Air [102] and Hipster-shop [197], provide standardized environments to assess the performance and reliability of various cloud-based solutions. Furthermore, large-scale distributed systems [206], IP Multimedia subsystems [149], web and database servers/applications [121], and web and e-commerce systems [129] further enrich the variety of primary data sources by offering insights from specific and often complex operational contexts. Notably, 85% of the reviewed papers utilized primary data, reflecting the importance of active data collection for real-time monitoring and reliability assessment.

Secondary data sources, corresponding to passive collection methods, encompass a wide range of preexisting datasets. Cloud service metrics from providers like IBM [102], Windows Azure [179], and Alibaba [180] are vital for developing and testing new algorithms without needing extensive data collection infrastructure. Benchmark datasets, including those from Yahoo, RUBIS, and NAB [92, 206], offer robust, standardized data for performance evaluation and anomaly

Table 8: A catalog of monitoring data origins in cloud-based systems

	Origin of monitoring data		Papers
Primary data	**Production systems**	Public	IBM [206], [38], [166]; Google [231]; Microsoft [88], [248] Amazon [227], [52], [201], [2]; OpenStack [106], [165], [34] Huawei Cloud [25]; AppScale [98]; Alibaba Cloud [151], [152] Once Cloud [99], [224]; Other or not specified [121], [78], [76], [216], [169], [235], [94], [188]
		Private	[212], [149], [245], [160], [129], [250], [176], [85], [185], [47], [187], [101], [221], [117], [197], [81], [240], [233], [58], [73], [140], [226], [186], [150], [70], [72], [127], [102], [234], [198], [167], [24], [86], [113], [128], [40], [194], [71], [5], [132], [13], [84]
	Research and experimental systems	Lab environment	[212], [245], [160], [129], [106], [188], [166], [101], [183], [9], [107], [186], [154], [102], [234], [111], [86], [246], [23], [84]
		Testbeds	[140], [70], [72], [127], [68], [157], [35], [147], [69], [198], [167], [113], [128], [40], [232], [194], [71], [5], [132], [13]
	Benchmarking systems (*not open-source)		Acme-Air [102]; Hipster-shop [197], [81], [129] Trainticket [24], [9], [226], [140], [240]; E-Shopper [213] SockShop [231], [73], [226]; TPC-W [41], [183] PiggyMetrics [226]; Social Network [150] Bench4Q [227], [225], [224], [150] Trade6* [38]; RuBBoS [107], [38]; RUBiS [70] Other or not specified [78], [176], [166], [57], [58], [111], [222]
	Large-scale/distributed (backend) system		[206], [76], [250], [88], [151], [85], [95], [52], [117], [23]
	IP Multimedia subsystem Clearwater		[149], [185], [47], [187]
	Web and database servers/applications		[121], [201], [2], [221], [165], [152], [98], [186], [246], [34]
	Web and e-commerce systems		[129], [142], [188], [197], [81], [41], [233], [227], [225], [58], [73], [224], [213], [222], [99]
Secondary data	**Cloud service metrics, logs, traces**	IBM Windows Azure Alibaba	[179], [180], [188], [247], [236], [102]
	Benchmark datasets	Yahoo RUBIS NAB	[206], [92], [25], [93], [2], [173], [74], [43], [62], [248], [196], [40]
	Datasets from AIOps Challenge Events		[129], [25], [173], [240], [126], [62]
	Web Logs		[137]
	Synthetic dataset		[179], [233]
	Public datasets	HDFS BGL Thunderbird SMD	[172], [228], [234], [24], [128]

detection, ensuring consistency in research outcomes. Data from AIOps Challenge Events offers another source of secondary data, often used to test advanced operational solutions in controlled scenarios [25, 129]. Public datasets, such as HDFS and BGL [172], contribute to the pool of resources available for researchers, enabling extensive testing and validation of new methods across diverse scenarios. The reliance on secondary data, used in 27% of the papers, may stem from researchers lacking access to real cloud environments, necessitating the use of publicly available datasets to validate their methods. This use of secondary data ensures that research can progress even without direct access to operational systems, enabling the development and testing of new anomaly detection techniques that can later be applied in real-world settings. It can also facilitate benchmark comparisons and replicability of research.

The overlap in the usage of primary data (85%) and secondary data (27%) in research indicates that some papers leverage both types of data to enhance the robustness of their studies. This dual approach allows researchers to validate their

findings in diverse datasets, increasing the reliability and generalizability of their results. Primary data offers real-time, context-specific insights crucial for accurate performance assessment and anomaly detection in cloud-based systems. This data is invaluable as it reflects the actual operational environment, ensuring that research findings and developed algorithms are grounded in reality. For example, data from Microsoft production systems [88] and Huawei Cloud [25] provide crucial insights for researchers to develop and test algorithms that are practical and effective in real-world conditions. The granularity and relevance of primary data enable researchers to address specific issues and challenges encountered in live environments, enhancing the robustness and reliability of cloud-based solutions.

4.5 RQ2: Overview of monitoring tools

As shown in Figure 3, the frequency of monitoring tools used across various research papers highlights the diverse functionality required to maintain robust observability in modern cloud computing environments. Prometheus, Sysstat/Perf, and Jaeger emerge as frequently used tools due to their comprehensive capabilities in metrics collection, performance analysis, and distributed tracing, respectively. **Prometheus** [24, 197, 212, 231, 232, 234, 245] is particularly valued for its powerful metrics collection and alerting system, which enables real-time monitoring and alerting of system performance. **Sysstat/Perf** [68–72, 194, 198] provides performance analysis with detailed insights into system resource usage and application performance. **Jaeger** [9,113,150,226,250], on the other hand, provides distributed tracing, allowing a user to trace the flow of requests across microservices and identify performance bottlenecks and latency issues.

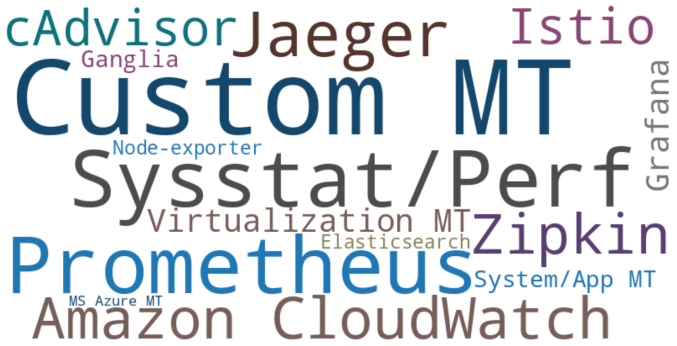


Figure 3: Overview of monitoring tools

The diversity of tools reflected in the data also indicates a trend toward specialized monitoring solutions tailored to specific use cases and technological stacks. For instance, **cAdvisor** [24,47,197,231] and **Node-exporter** [231,232] are pivotal in containerized environments, providing detailed insights into the performance and resource usage of individual containers. **Istio** [197, 231, 232, 240], another frequently used tool, offers comprehensive service mesh observability, enabling traffic management, security, and microservice monitoring. Additionally, **Zipkin** [34, 150, 160, 213, 226] is particularly useful for distributed tracing, helping in the collection and analysis of trace data from complex microservice architectures, thus identifying latency issues and performance bottlenecks across services. **Ganglia** [111, 185, 186] is essential for monitoring large clusters and grids, offering scalable distributed monitoring solutions.

Leading commercial tools including **Amazon CloudWatch** [2, 52, 137, 188, 201] and **MS Azure MT** [88] also provide scalable monitoring and logging services, useful for maintaining the health and performance of cloud-based applications. Moreover, **Grafana** [47,73,212] is widely used to visualize metrics, providing a powerful platform to build dashboards and alerts. This trend highlights the increasing complexity of IT environments and the need for specialized monitoring tools to ensure comprehensive observability and proactive system management across heterogeneous infrastructures.

However, the dominance of **custom monitoring tools (MT)** reflects the unique and diverse requirements of modern monitoring infrastructure. Custom tools are designed to meet specific needs that off-the-shelf solutions may not fully address, offering flexibility and precision in data collection and processing. Examples include custom monitoring infrastructures integrating SNMPv2c for application-level monitoring, Ceilometer for IaaS level monitoring, and Linux OS agents for operating system-level monitoring [149]. These specialized setups ensure comprehensive coverage across different layers of the IT infrastructure, capturing a wide range of metrics and performance indicators. Other custom setups reported include the use of Docker stats for live container data, mpstat for CPU utilization, vmstat for memory usage, Heapster for data aggregation, and InfluxDB for time-series data storage [183].

In research, the widespread use of custom-built monitoring tools can be attributed to the unique and specific requirements of experimental setups and the flexibility needed to accommodate evolving research questions. Custom tools offer researchers the ability to tailor their monitoring infrastructure precisely to their needs, ensuring comprehensive data collection and accurate analysis. For example, some setups use the CloudSim simulation environment [154]. The variety of custom tools and techniques, such as those used in Performance Anomaly Detector (PAD) [169], SHoWA testbed [147], and CloudDoc framework [151], underline the need for tailored solutions that can adapt to specific environments and workloads, providing granular insights that are crucial for optimizing performance and ensuring system reliability.

4.6 RQ3: Data preprocessing and anomaly detection techniques

In the analysis of data preprocessing techniques employed across the reviewed studies, distinct techniques have been identified as particularly suited to different types of data, whether derived from metrics, logs, or traces. These techniques are categorized based on their application to textual data, categorical data, numerical data, general-purpose tasks, and graph-based data, as shown in Figure 4. It should be noted that this classification is not exhaustive, as some techniques may be applied to multiple data types. The categories are derived on the basis of the techniques used in the reviewed studies.

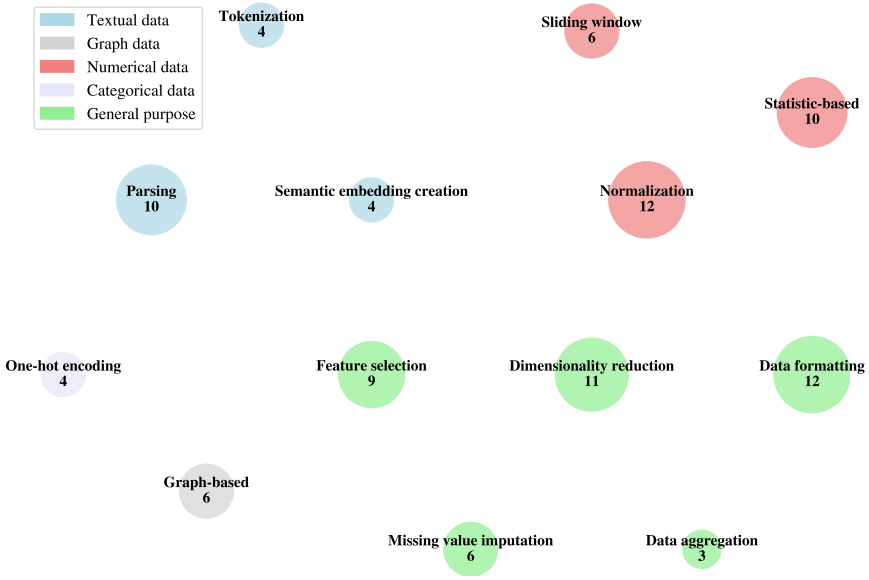


Figure 4: Overview of data preprocessing techniques and their categories with the number of papers employing each technique

Textual data processing techniques

In cloud-based systems, textual data is usually derived from logs and traces. This data often requires detailed preprocessing to extract meaningful insights and patterns.

Parsing transforms unstructured raw data (e.g., text) into structured formats or templates by breaking it down into manageable components like words or phrases. For instance, in log analysis, parsing might involve identifying key fields such as

log messages, timestamps, event types, and error codes, which can then be used for further processing. Multiple studies demonstrate its widespread use [76, 101, 102, 106, 117, 127, 140, 160, 172, 194].

Tokenization is a preprocessing step in NLP that involves splitting raw text into smaller, manageable units known as *tokens*. These tokens typically represent words, phrases, or even individual characters, depending on the specific requirements of the analysis. Given the abundant amount of data generated by logs and traces, tokenization helps in reducing the complexity of the data, making it more manageable and easier to process in real-time, as highlighted in several studies [127, 140, 160, 172].

Semantic embedding creation transforms textual data into dense vector representations. These vectors, often referred to as *embeddings*, capture the semantic meaning of the text, enabling more advanced text analysis. For example, if two logs are semantically similar, meaning they describe similar system states or events, their embeddings will be close in the vector space. Several studies have shown the effectiveness of semantic embedding creation in cloud-based systems [78, 127, 140, 172].

Categorical data processing techniques

Categorical data in cloud-based systems are typically derived from structured representations such as service types, event sequences, or status codes, primarily found in logs and traces.

One-hot encoding is a technique used to convert categorical data into binary vectors, where each unique category is represented by a distinct binary feature. This technique could be applied to represent different *services*, *events*, or *call paths* as binary vectors, indicating their presence or absence in a trace. It can also be adapted to structured data in microservice environments, where the goal is to transform categorical attributes like service types or event keys into a format suitable for ML models [24, 117, 140, 240].

Numerical data processing techniques

Numerical data, which are abundant in metrics and traces, require specific preprocessing techniques to ensure that they can be used effectively by ML models.

Normalization is a widely used technique for preparing numerical data by scaling it to a specific range, depending on the requirements of the analysis. Normalization helps reduce the potential for bias in ML models by preventing any single feature from dominating others due to its scale. Studies in our review have shown that normalization is a critical step in preprocessing pipelines, leading to improved model accuracy [2, 69, 76, 88, 106, 111, 113, 121, 160, 173, 187].

Statistical techniques [86, 101, 106, 111, 113, 129, 151, 201, 216] are frequently applied to numerical data from metrics and traces. The development of these techniques often involves sophisticated mathematical modeling and a deep understand-

ing of the underlying data structures. Custom statistical techniques are essential for pushing the boundaries of conventional data analysis, providing tailored solutions that enhance the depth and accuracy of insights derived from complex datasets.

Sliding window is specifically used in time-series analysis of metrics and traces to create overlapping segments of data or "windows". This technique captures temporal dependencies crucial for understanding data dynamics over time, as shown in some of the studies [41, 69, 74, 81, 173, 240].

Graph data processing techniques

In cloud-based systems, data involving network structures or relationships are usually represented as graphs and are primarily derived from logs and traces.

Graph-based techniques [24, 102, 113, 196, 232, 233] are essential for representing data in graph form. In cloud environments, data is often highly interconnected, with numerous dependencies between different components, such as microservices, network devices, or user sessions. Graph-based techniques allow this interconnected data to be visualized and analyzed in a way that highlights these relationships, making it easier to identify patterns, dependencies, and potential issues.

General purpose processing techniques

Broadly applicable to different types of data such as metrics, logs, and traces, these general-purpose preprocessing techniques are valuable tools for data analysis.

Data formatting is essential for ensuring that data from different sources are aligned to a common format. Each of these sources may have different data structures, formats, units of measurement, and levels of granularity. Without proper formatting, comparing or combining these datasets for analysis would be challenging. This technique is frequently used in studies dealing with heterogeneous data [24, 43, 72, 74, 81, 106, 147, 173, 185, 227, 231, 233].

Feature selection is used to identify and retain the most important and relevant features within a dataset. In cloud environments, datasets often contain numerous features, many of which may be redundant, irrelevant, or noisy. Feature selection helps to streamline the dataset, ensuring that only the most informative features are retained for analysis. The importance of this technique is well-documented in various studies [62, 72, 111, 132, 187, 212, 231, 234, 247].

Dimensionality reduction techniques [41, 68, 71, 72, 76, 167, 198, 212, 221, 224, 234] aim to transform the entire dataset into a lower-dimensional space while still preserving its essential characteristics. This transformation is critical in scenarios where the original data contains a large number of features.

Missing value imputation [71, 72, 95, 121, 194, 212] is particularly used for handling incomplete datasets, where some data points may be missing. Imputation techniques work by filling in these gaps with estimated values, ensuring that the dataset remains as complete as possible.

Data aggregation involves the process of combining and summarizing data from various sources into a single, unified dataset. This technique is particularly useful when dealing with large volumes of data that originate from different components of a system, each of which might generate data at different granularities or with different formats [147, 187, 233].

It is interesting that 38 papers did not report any data preprocessing techniques, and 11 papers explicitly stated that no preprocessing was needed. This variation highlights the context-dependent nature of data preprocessing. In some cases, raw data (e.g., numerical data) may already be in a suitable format for analysis, or the research focus might be on methods that inherently handle raw data without the need for extensive preprocessing. Thus, the choice of data preprocessing techniques is highly influenced by the nature of the dataset and the specific requirements of the analysis.

Anomaly detection techniques

Regarding the second part of this research question, Figure 9 provides a comprehensive overview of various anomaly detection techniques and their applicability to different types of monitoring data. This classification is essential for researchers and practitioners aiming to select the most appropriate technique for their specific anomaly detection tasks, ensuring that the chosen technique aligns with the data characteristics and requirements of the task. The techniques are broadly categorized into statistical, machine learning, and deep learning, each offering unique advantages and areas of application based on the data involved.

According to Figure 9, statistical techniques, such as **SPOT**, **VAR**, and **ARIMA**, are fundamental methods effective in handling time-series data like metrics and system data. **Bayesian methods**, known for their probabilistic approach, are widely applicable across various data types, including logs and network data. **Custom statistical methods** demonstrate versatility by being applicable to nearly all data types. These techniques are particularly beneficial in scenarios where the underlying data distribution is either known or can be estimated, allowing for straightforward implementation and effective anomaly detection.

The next category, including machine learning techniques, enhances capabilities for more complex and dynamic data patterns. **Ensemble-based methods**, including decision trees and random forests, excel in handling heterogeneous data sources such as logs, application data, and network data. These techniques aggregate multiple models to improve robustness and accuracy, making them suitable for environments with varied data characteristics. Isolation trees are also an ensemble-based anomaly detection technique that isolates anomalies by recursively partitioning the data. This method relies on the concept that anomalies, being few and different, are easier to isolate and thus require fewer partitions. This technique is particularly well-suited for high-dimensional datasets and is com-

Table 9: A catalog of anomaly detection techniques mapped to data types

	AD techniques		Metrics	Logs	Traces	System data	Infrastructure data	Application data	Network data	Database data
STATISTICAL	SPOT [166]; VAR [76]		✓	✓				✓		
	ARIMA [187]		✓			✓			✓	
	Bayesian methods [72], [43], [94], [113], [194], [71], [47]		✓	✓		✓			✓	
	Custom statistical [212], [149], [137], [35], [147], [197], [169], [179], [142], [227], [113], [225], [57], [58], [222], [180], [73], [99], [236], [132], [13], [52], [152], [98]		✓	✓		✓	✓	✓	✓	
MACHINE LEARNING	**Ensemble-based**	Decision tree [41], [198], [111], [113], [71]	✓	✓		✓		✓	✓	
		Random forest [62], [113], [185], [47]	✓	✓		✓			✓	
		Isolation trees [154]	✓			✓				
	Distance-based	Nearest Neighbors [47]	✓			✓			✓	
	Margin-based	SVM [41], [167], [201], [47]	✓	✓		✓		✓	✓	
	Linear variation	[41], [234], [221]	✓			✓		✓	✓	
	Neural-network	ELM [245]	✓			✓			✓	
	Clustering	Distance-based [231], [240], [113], [232], [246], [52], [152]	✓	✓	✓	✓		✓	✓	
		Centroid-based [213]			✓				✓	
		Incremental algorithm [212], [68], [227]	✓			✓	✓		✓	
		Data stream algorithm [186]	✓			✓			✓	
	Density-based	Density Spatial Clustering [93], [247]	✓			✓				
		(Local Outlier Factor) LOF [76]	✓	✓		✓		✓		
	Dimensionality reduction	PCA [74], [69], [106], [38], [2], [224], [23], [152]	✓	✓	✓	✓		✓	✓	
		SVD [92]	✓							
		SOM [121], [78]	✓	✓		✓		✓		
		Matrix sketching [23]			✓			✓		
	Similarity-based	Similarity measure algorithm [150]			✓	✓		✓		
		kTail Algorithm [151]			✓			✓		
	Hierarchical Models	HTM, HMM [43], [183], [86], [176], [85], [34], [84]	✓		✓	✓		✓	✓	
	Pattern sketching	ADSketch [25]	✓			✓			✓	
	Median-based	PMAD [216]	✓			✓		✓		
DEEP LEARNING	**Feed-forward**	MLP [113]		✓		✓				
	Sequential	LSTM, GRU, RNN [127], [173], [74], [160], [81], [40], [88], [9], [188], [95], [117]	✓	✓	✓	✓		✓	✓	✓
		Transformers [128]	✓			✓			✓	
	Autoencoder variations	[74], [81], [233], [94], [250], [88], [9], [140]	✓		✓	✓	✓	✓	✓	
	Graph-based	GNN, TCFG [81], [196], [101], [102]	✓	✓	✓	✓		✓	✓	

monly applied to metrics and system data, where they efficiently detect outliers across various data dimensions.

Distance-based methods like nearest neighbors are effective for metrics, system, and network data due to their ability to detect anomalies based on proximity measures. **Margin-based methods** such as Support Vector Machines (SVM) are

advantageous for binary classification and can detect anomalies in high-dimensional data, often found in system and application data, as they can effectively classify anomalies by creating hyperplanes that separate normal and abnormal instances [201].

Clustering techniques are particularly useful for unsupervised anomaly detection tasks. Distance-based clustering methods group similar data points and identify outliers that do not conform to any cluster, making them ideal for traces and system data where labeled anomalies are scarce. Centroid-based clustering, such as k-means, is effective in identifying central points in data clusters and recognizing anomalies as deviations from these centers. Incremental algorithms and data stream clustering methods are designed to handle continuously evolving data, making them suitable for real-time anomaly detection in dynamic environments.

Density-based anomaly detection techniques identify anomalies by analyzing the density of data points in a given space. The core idea is that normal data points typically belong to dense regions, while anomalies lie in sparser regions. Techniques like Density Spatial Clustering and Local Outlier Factor (LOF) are commonly used. These techniques are particularly effective for detecting anomalies in complex datasets, including metrics, system data, and application data, which may have varying shapes and densities.

Dimensionality reduction techniques like Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) reduce the complexity of high-dimensional data, facilitating anomaly detection in metrics and network data by isolating significant patterns. These techniques facilitate anomaly detection in metrics and network data by reducing noise and highlighting key features that distinguish normal from abnormal behavior. Techniques like PCA can also help in visualizing complex data structures, making it easier to identify and interpret anomalies.

Similarity-based techniques identify anomalies by comparing data points to each other, looking for those that deviate significantly from the norm. Techniques like the Similarity Measure Algorithm and kTail Algorithm work by calculating similarity scores or distances between data points, flagging those with low similarity to others as anomalies. These methods are particularly effective in scenarios where consistent patterns are expected, making them well-suited for detecting anomalies in traces, system data, and application data. Additionally, Hierarchical models, such as Hidden Markov Models (HMM) and Hierarchical Temporal Memory (HTM), are effective for sequential data like traces, capturing temporal dependencies to identify anomalies over time.

The third category of **deep learning techniques** are increasingly popular due to their ability to model complex, non-linear relationships in large datasets. Feedforward neural networks, such as Multi-Layer Perceptrons (MLP), are effective for logs and system data, capturing intricate relationships between features. Sequential models, including Long Short-Term Memory (LSTM) networks and Transformers, are highly effective for time-series data like metrics as well as for logs and traces due to their proficiency in capturing temporal patterns and long-term

dependencies. Autoencoder variations, including denoising and variational autoencoders, are versatile in learning data representations, making them suitable for traces, system data, and application data. Graph-based techniques, such as Graph Neural Networks (GNN), excel in handling data with complex interdependencies, often found in logs and network data, by leveraging graph structures to model relationships and detect structural anomalies.

It is important to note that the type of learning (**supervised, unsupervised, or semi-supervised**) was not specified in the table, as it depends heavily on the availability of labels and the specific context of the application. For example, LSTM networks can be used in a supervised manner when it is trained with labeled data to predict specific anomalies, as demonstrated by Qiu et al. [173]. However, in some frameworks, LSTM may only be used to model sequences in monitoring data without utilizing labels [74,81]. For instance, in unsupervised settings, LSTM networks can be employed to capture and learn the inherent patterns and dependencies within sequential data. In this context, LSTM networks act as feature extractors that understand the temporal structure of the data, which can then be used as input for other unsupervised models like autoencoders or clustering algorithms. Additionally, semi-supervised approaches are common, where representations of logs or traces are generated using unsupervised models, and then sequence models or graph-based models are used for prediction or classification [24].

The flexibility in learning types ensures that the selected anomaly detection technique can be adapted to the specific context and data characteristics, providing a robust solution for identifying and mitigating anomalies across various domains. Understanding the relationship between data types, anomaly detection techniques, and learning frameworks is crucial for developing effective AD solutions tailored to the unique challenges of different datasets. This comprehensive catalog serves as a valuable guide for selecting the most appropriate AD techniques based on specific data characteristics and application requirements.

4.7 RQ4: Representativeness of real-world operational environments

This section describes the categories of representativeness identified in reviewed papers regarding monitoring data and labels in cloud operational contexts. By **representativeness**, we refer to how well the data and labels used for training and evaluating anomaly detection techniques mirror real-world scenarios, capturing the complexities and authentic conditions encountered in operational environments. **More representative data** typically originates from systems deployed in production, encompassing actual usage patterns and the anomalies that occur in practice. This category also includes benchmark datasets from real systems, often used as standards for evaluating anomaly detection techniques due to their relevance and realism. **More representative labels** are those verified by domain

experts to ensure that the detected anomalies are considered valid and significant to practitioners.

In contrast, **less representative data** is often collected from emulated usage of systems, testbeds, or simulated environments. While useful for preliminary testing and development, these data sources lack the complexity and unpredictability of real-world scenarios. Consequently, models trained on such data may perform well under controlled conditions but struggle when deployed in production. Simplified or synthetic data do not capture a range of factors that influence real-world systems, leading to models that may not generalize well.

In our analysis, we found that 71% of the papers used more representative data sources, while 29% used less representative data sources. The high percentage of more representative data indicates a strong focus on using realistic data sources to validate anomaly detection techniques, ensuring their relevance in real operational contexts. However, when we examined the representativeness of the labels used, a different picture emerged. We found that 73% of papers used labels that were less representative, being either artificially generated, labeled by researchers, injected without verification by domain experts, or resulting from common issues such as infrastructure problems and performance degradation. Thus, while the data used for training and evaluation are increasingly realistic, the labels often lack the same level of representativeness. This discrepancy can lead to models that are well-calibrated to detect anomalies in realistic data, but may still suffer from inaccuracies due to the less reliable labeling.

Therefore, we used the categorization of more and less representative data to assess the relevance of anomaly detection techniques and their likely applicability to real industrial contexts. Figure 5 provides an overview of anomaly detection techniques evaluated in more and less representative cloud environments based on the data used for training and evaluation, serving as a guide for researchers and practitioners.

Accordingly, techniques evaluated with data from deployed production systems are often tailored to handle the complexities of actual operational environments, ensuring their robustness and practical applicability. Such thorough evaluation with real-world data provides confidence that these techniques are likely to deliver reliable performance in industrial settings, i.e., reducing the risk of false positives or missed detections, which can be costly in operational environments. Furthermore, using more representative data in several evaluations shows that these techniques are tailored to handle the actual challenges found in production settings. This makes them more useful and easier to apply in different scenarios, giving practitioners a strong starting point to tailor the models according to their specific operational needs. Examples of these techniques include custom statistical models, clustering, sequential models (LSTM, GRU, and RNN), autoencoder variations, and PCA.

On the other hand, techniques evaluated with *less representative* data might perform well under controlled conditions but may struggle to generalize to the

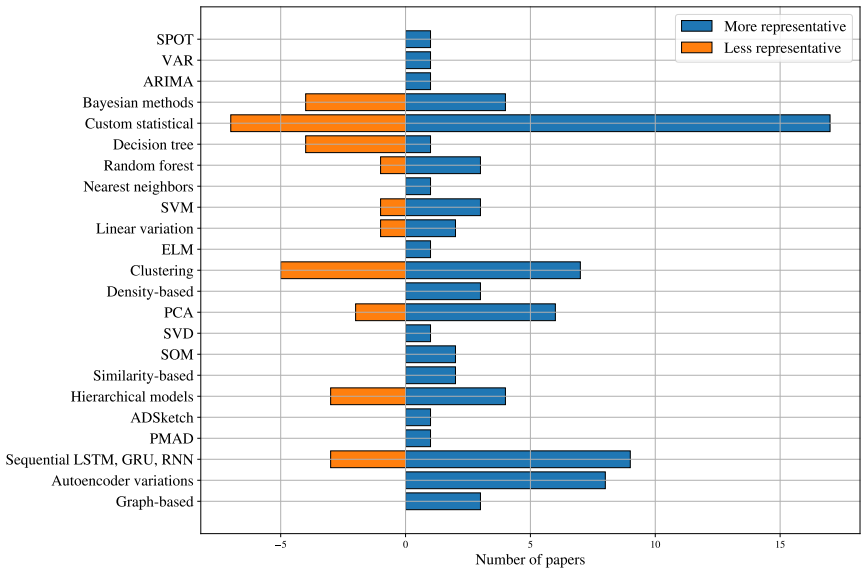


Figure 5: Overview of anomaly detection techniques trained and evaluated with more and less representative data

complexities of real-world operations. This lack of exposure to realistic scenarios indicates that these techniques are less mature and may not be fully reliable when deployed in operations. For instance, while decision trees and random forests may be useful, their current evaluation in primarily less complex environments means that they may not perform as well in real-world scenarios. Similarly, as hierarchical models and isolation trees are often tested in simulated settings, there is a need for their further validation in real-world environments to ensure their maturity and effectiveness.

5 Discussion and Conclusion

This study may significantly advance the understanding and practice of anomaly detection in cloud-based systems by offering a structured and empirical approach to the topic. A key contribution of this study is the classification of monitoring data collected from operational environments. This classification aids in understanding the diverse nature of the monitoring data, their properties, and their relevance to anomaly detection in real-world cloud environments **(C1)**. Furthermore, the study provides an extensive overview of monitoring tools used across various cloud environments. This resource is invaluable for practitioners looking to implement or enhance their monitoring infrastructures **(C2)**. The diversity of tools presented in

this study reflects the complexity and varied requirements of modern cloud systems, emphasizing the need for flexible and adaptable monitoring solutions.

One interesting observation is that there is a lower use of commercial monitoring tools in research, which may be due to their predefined features, limited flexibility, and potentially high costs that may not align with the specialized needs of research projects. Instead, researchers might prioritize custom-built tools for their ability to integrate seamlessly with experimental workflows and address specific research challenges. A similar pattern emerges when comparing DevOps and microservices monitoring tools identified in a recent review of grey literature [65] to the tools identified in our study. Both sources identified widely adopted tools (custom/research and commercial/industry) like Prometheus, Jaeger, Zipkin, and Elasticsearch, which are crucial for metrics collection, distributed tracing, and log management in cloud-based systems. However, the review of the grey literature includes a larger number of tools primarily from the industry, such as Nagios, Zabbix, and Dynatrace, underscoring the importance of advanced and multipurpose monitoring solutions, while our findings emphasize the need for more customized tools tailored to specific monitoring and research requirements.

Moreover, the detailed examination of data preprocessing and anomaly detection techniques in our mapping study offers practical guidance for selecting suitable methods based on specific data characteristics. This comprehensive overview includes an in-depth analysis of different preprocessing techniques, such as normalization, log parsing, and feature selection, which are critical for preparing the data before applying anomaly detection algorithms. Additionally, the mapping of anomaly detection techniques to various categories of monitoring data helps practitioners understand which techniques are most effective for different kinds of data, such as metrics, logs, and traces. This detailed understanding enables the optimization of anomaly detection workflows, as practitioners can select the most appropriate techniques for their specific datasets, thus improving the accuracy and reliability of detecting operational anomalies **(C3)**.

A critical component of this study is its analysis of real-world aspects, examining the extent to which monitoring data and labels reflect actual operational and cloud contexts. This insight is essential to ensure the practical applicability of anomaly detection techniques **(C4)**. Understanding the **representativeness** of the data and labels used in evaluating anomaly detection techniques is crucial for developing robust, reliable models that perform effectively in production environments. By focusing on more representative data and striving for higher-quality labels, researchers and practitioners can ensure the development of solutions that are truly applicable to real-world industrial contexts.

In summary, this study makes significant contributions to the field of anomaly detection in cloud-based systems by addressing the multifaceted challenges of real-world scenarios, including diverse data sources, system architectures, and dynamic operational conditions. The structured approach, with its comprehensive catalogs and detailed analyses, offers researchers and practitioners the necessary

tools and knowledge to develop effective anomaly detection frameworks applicable to real operational environments. Additionally, comprehensive catalogs of monitoring data enable straightforward identification and prioritization of critical health indicators in a cloud-based system. By focusing on real-world applicability, our study bridges the gap between theoretical research and practical implementation, providing actionable insights that can improve operational resilience in industrial contexts.

Furthermore, our emphasis on leveraging empirical data from actual cloud operations offers insights beyond the limitations of synthetic datasets typically used in existing research. By grounding our findings in real-world data, we ensure that the recommendations are both relevant and immediately applicable, making a direct impact on the reliability and performance of cloud environments. This dual focus on advancing academic knowledge and offering practical guidance contributes to the development of more resilient and efficient cloud-based systems.

Acknowledgments

The authors thank Prof. Kai Petersen, Hochschule Flensburg, Germany, for insightful discussions and input on the design of the study. This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Furthermore, this work was supported by ELLIIT, the Swedish government-funded Strategic Research Area within IT and Mobile Communications, and a research grant for the GIST project (reference number 20220235) from the Swedish Knowledge Foundation. The work was also supported by a grant from the Research Council of Finland (grants n. 349487- MuFAno).

Appendix 1: Search to establish the need for review

The first step of our review process involved conducting a search within electronic databases to identify existing systematic reviews relevant to our research questions. We limited the search only to the Scopus library as argued in Section 3.2. Accordingly, we aimed to identify the need and research gap for conducting our systematic mapping study. In our search, we relied on keywords derived from our research questions and the Quasi-Gold Standard (QGS), whose selection is discussed in Section 3.3. The search for relevant secondary studies is conducted using the following search string:

TITLE-ABS-KEY (("anomaly detection" OR "monitoring") AND ("cloud" OR "microservices" OR "software system") AND ("systematic" AND ("mapping" OR "review" OR "map"))) AND (LIMIT-TO (DOCTYPE , "ar") OR LIMIT-TO (DOC-*

TYPE , "re") OR LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "sh")) AND (LIMIT-TO (SUBJAREA , "COMP")) AND (LIMIT-TO (LANGUAGE , "English"))

Thus, our search is limited to English-language, peer-reviewed conference papers, articles, reviews, or short surveys in the field of computer science. In total, this search strategy returned 139 hits. After reading the titles, only seven papers [22, 32, 66, 75, 100, 156, 161] were selected as related reviews.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescapè. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115, 2013.

[2] Bikash Agrawal, Tomasz Wiktorski, and Chunming Rong. Adaptive real-time anomaly detection in cloud infrastructures. *Concurrency and Computation: Practice and Experience*, 29(24):e4193, December 2017.

[3] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[4] Joan E. van Aken. Management research based on the paradigm of the design sciences: The quest for field-tested and grounded technological rules. *Journal of Management Studies*, 41(2):219–246, 2004.

[5] Ahmad Alnafessah and Giuliano Casale. A Neural-Network Driven Methodology for Anomaly Detection in Apache Spark. In *11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 201–209, Coimbra, September 2018. IEEE.

[6] Ahmad Alnafessah, Alim Ul Gias, Runan Wang, Lulai Zhu, Giuliano Casale, and Antonio Filieri. Quality-aware devops research: Where do we stand? *IEEE Access*, 9:44476–44489, 2021.

[7] Juncal Alonso, Leire Orue-Echevarria, and Maider Huarte. Cloudops: Towards the operationalization of the cloud continuum: Concepts, challenges and a reference framework. *Applied Sciences*, 12(9), 2022.

[8] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, 2021.

[9] Laigang Bai and Cheng Zhang. Trace-based microservice anomaly detection through deep learning. In *Second International Conference on Electronic Information Engineering, Big Data, and Computer Technology (EIBDCT 2023)*, page 126422T, Xishuangbanna, China, May 2023. SPIE.

[10] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Microservices architecture enables DevOps: an experience report on migration to a cloud-native architecture. *IEEE Software*, 33:1–1, 05 2016.

[11] Sebastian Baltes and Paul Ralph. Sampling in software engineering research: A critical review and guidelines. *Empirical Softw. Engg.*, 27(4), jul 2022.

[12] Md Abul Bashar and Richi Nayak. TAnoGAN: Time Series Anomaly Detection with Generative Adversarial Networks. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1778–1785, Canberra, ACT, Australia, December 2020. IEEE.

[13] Souhila Benmakrelouf, Cédric St-Onge, Nadjia Kara, Hanine Tout, Claes Edstrom, and Yves Lemieux. Abnormal behavior detection using resource level to service level metrics mapping in virtualized systems. *Future Generation Computer Systems*, 102:680–700, January 2020.

[14] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. A Review on Outlier/Anomaly Detection in Time Series Data. *ACM Computing Surveys*, 54(3):1–33, June 2021.

[15] Olimar Borges, Julia Couto, Duncan Ruiz, and Rafael Prikladnicki. Challenges in using machine learning to support software engineering. In *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021)*, pages 224–231, 01 2021.

[16] Jürgen Börstler, Nauman Bin Ali, and Kai Petersen. Double-counting in software engineering tertiary studies - an overlooked threat to validity. *Inf. Softw. Technol.*, 158:107174, 2023.

[17] Nathan Bosch and Jan Bosch. Software Logs for Machine Learning in a DevOps Environment. In *46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 29–33, August 2020.

[18] Rodrigo N. Calheiros, Kotagiri Ramamohanarao, Rajkumar Buyya, Christopher Leckie, and Steve Versteeg. On the effectiveness of isolation-based anomaly detection in cloud data centers. *Concurrency and Computation: Practice and Experience*, 29(18):e4169, 2017.

[19] Antonio Capizzi, Salvatore Distefano, Luiz J. P. Araújo, Manuel Mazzara, Muhammad Ahmad, and Evgeny Bobrov. Anomaly detection in DevOps Toolchain. In Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 37–51. Springer International Publishing, 2020.

[20] Antonio Capizzi, Salvatore Distefano, and Manuel Mazzara. From DevOps to DevDataOps: Data management in devops processes. In Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 52–62. Springer International Publishing, 2020.

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

[22] Bhavana Chaurasia, Anshul Verma, and Pradeepika Verma. An in-depth and insightful exploration of failure detection in distributed systems. *Computer Networks*, 247, 2024.

[23] Hongyang Chen, Pengfei Chen, and Guangba Yu. A Framework of Virtual War Room and Matrix Sketch-Based Streaming Anomaly Detection for Microservice Systems. *IEEE Access*, 8:43413–43426, 2020.

[24] Jian Chen, Fagui Liu, Jun Jiang, Guoxiang Zhong, Dishi Xu, Zhuanglun Tan, and Shangsong Shi. TraceGra: A trace-based anomaly detection for microservice using graph deep learning. *Computer Communications*, 204:109–117, April 2023.

[25] Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xiao Ling, Yongqiang Yang, and Michael R. Lyu. Adaptive performance anomaly detection for online service systems via pattern sketching. In *Proceedings of the 44th International Conference on Software Engineering*, pages 61–72, Pittsburgh Pennsylvania, May 2022. ACM.

[26] Antoine Chevrot, Alexandre Vernotte, and Bruno Legeard. DAE : Discriminatory Auto-Encoder for multivariate time-series anomaly detection in air transportation. *arXiv:2109.04247 [cs]*, September 2021.

[27] Kukjin Choi, Jihun Yi, Changhwa Park, and Sungroh Yoon. Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines. *IEEE Access*, 9:120043–120065, 2021.

[28] Marcello Cinque, Raffaele Della Corte, and Antonio Pecchia. Advancing monitoring in microservices systems. In *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 122–123, Berlin, Germany, 2019. IEEE.

[29] Jürgen Cito, Philipp Leitner, Harald C. Gall, Aryan Dadashi, Anne Keller, and Andreas Roth. Runtime metric meets developer: Building better cloud applications using feedback. In *ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 14–27, Pittsburgh, PA, USA, 2015. ACM Press.

[30] Jürgen Cito, Johannes Wettinger, Lucy Ellen Lwakatare, Markus Borg, and Fei Li. Feedback from Operations to Software Development—A DevOps Perspective on Runtime Metrics and Logs. In Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, volume 11350, pages 184–195. Springer International Publishing, 2019.

[31] Andrew A. Cook, Göksel Mısırlı, and Zhong Fan. Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2020.

[32] Breno Costa, João Bachiega, Leonardo Rebouças Carvalho, Michel Rosa, and Aleteia Araujo. Monitoring fog computing: A review, taxonomy and open challenges. *Computer Networks*, 215:109189, October 2022.

[33] Domenico Cotroneo, Luigi De Simone, Pietro Liguori, and Roberto Natella. Enhancing the analysis of software failures in cloud computing systems with deep learning. *Journal of Systems and Software*, 181:111043, November 2021.

[34] Domenico Cotroneo, Luigi De Simone, Pietro Liguori, Roberto Natella, and Nematollah Bidokhti. Enhancing Failure Propagation Analysis in Cloud Computing Systems. In *IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 139–150, Berlin, Germany, October 2019. IEEE.

[35] Adam R. Currie, Sally I. McClean, Philip Morrow, Gerard P. Parr, and Kashaf Khan. Using Correlations for Application Monitoring in Cloud Computing. In *14th International Symposium on Pervasive Systems, Algorithms and Networks & 11th International Conference on Frontier of Computer Science and Technology & Third International Symposium of Creative Computing (ISPAN-FCST-ISCC)*, pages 211–217, Exeter, June 2017. IEEE.

[36] Jeanderson Cândido, Maurício Aniche, and Arie van Deursen. Log-based software monitoring: a systematic mapping study. *PeerJ Computer Science*, 7:e489, May 2021. Publisher: PeerJ Inc.

[37] Yingnong Dang, Qingwei Lin, and Peng Huang. AIOps: Real-World Challenges and Research Innovations. In *IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5, Montreal, QC, Canada, May 2019. IEEE.

[38] Salvador DeCelles, Tingshan Huang, Matthew C. Stamm, and Nagarajan Kandasamy. Detecting Incipient Faults in Software Systems: A Compressed Sampling-Based Approach. In *IEEE 9th International Conference*

on Cloud Computing (CLOUD), pages 303–310, San Francisco, CA, USA, June 2016. IEEE.

[39] Pratyush Kr. Deka, Yash Verma, Adil Bin Bhutto, Erik Elmroth, and Monowar Bhuyan. Semi-supervised range-based anomaly detection for cloud systems. *IEEE Transactions on Network and Service Management*, 20(2):1290–1304, 2023.

[40] Pratyush Kr. Deka, Yash Verma, Adil Bin Bhutto, Erik Elmroth, and Monowar Bhuyan. Semi-Supervised Range-Based Anomaly Detection for Cloud Systems. *IEEE Transactions on Network and Service Management*, 20(2):1290–1304, June 2023.

[41] Pierangelo Di Sanzo, Alessandro Pellegrini, and Dimiter R. Avresky. Machine Learning for Achieving Self-Properties and Seamless Execution of Applications in the Cloud. In *IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA)*, pages 51–58, Munich, Germany, June 2015. IEEE.

[42] Oscar Dieste, Anna Grimán, and Natalia Juristo. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*, 14(5):513–539, October 2009.

[43] Nan Ding, Huanbo Gao, Hongyu Bu, Haoxuan Ma, and Huaiwei Si. Multivariate-Time-Series-Driven Real-time Anomaly Detection Based on Bayesian Network. *Sensors*, 18(10):3367, October 2018.

[44] Nan Ding, HaoXuan Ma, Huanbo Gao, YanHua Ma, and GuoZhen Tan. Real-time anomaly detection based on long short-term memory and gaussian mixture model. *Computers & Electrical Engineering*, 79:106458, 2019.

[45] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: Yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer International Publishing, 2017.

[46] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, Dallas Texas USA, October 2017. ACM.

[47] Qingfeng Du, Tiandi Xie, and Yu He. Anomaly Detection and Diagnosis for Container-Based Microservices with Performance Monitoring. In Jaideep

Vaidya and Jin Li, editors, *Algorithms and Architectures for Parallel Processing*, volume 11337, pages 560–572. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[48] Walid El-Shafai, Iman Almomani, and Aala AlKhayer. Visualized malware multi-classification framework using fine-tuned cnn-based transfer learning models. *Applied Sciences*, 11(14), 2021.

[49] Emelie Engström, Margaret-Anne Storey, Per Runeson, Martin Höst, and Maria Teresa Baldassarre. How software engineering research aligns with design science: a review. *Empirical Software Engineering*, 25(4):2630–2660, July 2020.

[50] Tolga Ergen and Suleyman Serdar Kozat. Unsupervised Anomaly Detection With LSTM Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):3127–3141, August 2020.

[51] Matthew E. Falagas, Eleni I. Pitsouni, George A. Malietzis, and Georgios Pappas. Comparison of pubmed, scopus, web of science, and google scholar: strengths and weaknesses. *The FASEB Journal*, 22(2):338–342, 2008.

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

[53] Michael Felderer, Barbara Russo, and Florian Auer. On testing data-intensive software systems. In Stefan Biffl, Matthias Eckhart, Arndt Lüder, and Edgar R. Weippl, editors, *Security and Quality in Cyber-Physical Systems Engineering*, pages 129–148. Springer, 2019.

[54] Robert Feldt and Ana Magazinius. Validity threats in empirical software engineering research - an initial survey. In *SEKE - Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering*, page 374 – 379, 2010.

[55] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, January 2017.

[56] Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916 – 954, 2008.

[57] Senbo Fu, Hyong Kim, and Rui Prior. FlowBox: Anomaly Detection Using Flow Analysis in Cloud Applications. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, San Diego, CA, USA, December 2015. IEEE.

[58] Senbo Fu, Rui Prior, and Hyong Kim. DMFD: Non-Intrusive Dependency Inference and Flow Ratio Model for Performance Anomaly Detection in Multi-Tier Cloud Applications. In *IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 164–173, Milan, Italy, July 2019. IEEE.

[59] Xiaoyu Fu, Rui Ren, Sally A. McKee, Jianfeng Zhan, and Ninghui Sun. Digging deeper into cluster system logs for failure prediction and root cause diagnosis. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 103–112, Madrid, Spain, September 2014. IEEE.

[60] Michael Gall and Federico Pigni. Taking devops mainstream: a critical review and conceptual framework. *European Journal of Information Systems*, 31(5):548–567, 2022.

[61] Florian Gardin, Ronan Gautier, Nicolas Goix, Bibi Ndiaye, and Jean-Matthieu Schertzer. Machine learning with logical rules in Python. `https://github.com/scikit-learn-contrib/skope-rules`, 2020.

[62] Rameshwar Garg, Chandana Kiran Ambekar, Kaustuv Saha, and Girish Rao Salanke N S. PROFCAD: An Algorithm to Detect Anomalies in Cloud Applications for KPI Monitoring Systems. In *IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, pages 1–7, Bangalore, India, December 2021. IEEE.

[63] Radoslav Gatev. Observability: Logs, metrics, and traces. *Introducing distributed application runtime (dapr) simplifying microservices applications development through proven and reusable patterns and practices*, pages 233–252, 2021.

[64] Alexander Geiger, Dongyu Liu, Sarah Alnegheimish, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks. In *IEEE International Conference on Big Data (Big Data)*, pages 33–43, Atlanta, GA, USA, December 2020. IEEE.

[65] L. Giamattei, A. Guerriero, R. Pietrantuono, S. Russo, I. Malavolta, T. Islam, M. Dînga, A. Koziolek, S. Singh, M. Armbruster, J.M. Gutierrez-Martinez, S. Caro-Alvaro, D. Rodriguez, S. Weber, J. Henss, E. Fernandez Vogelin, and F. Simon Panojo. Monitoring tools for DevOps and microservices: A systematic grey literature review. *Journal of Systems and Software*, 208:111906, February 2024.

[66] Asif Qumer Gill and Sarhang Hevary. Cloud Monitoring Data Challenges: A Systematic Review. In *Neural Information Processing*, volume 9947, pages 72–79. Springer International Publishing, 2016.

[67] L. Girish and Sridhar K. N. Rao. Anomaly detection in cloud environment using artificial intelligence techniques. *Computing*, 105(3):675–688, March 2023.

[68] Qiang Guan, Chi-Chen Chiu, Ziming Zhang, and Song Fu. Efficient and Accurate Anomaly Identification Using Reduced Metric Space in Utility Clouds. In *IEEE Seventh International Conference on Networking, Architecture, and Storage*, pages 207–216, Xiamen, China, June 2012. IEEE.

[69] Qiang Guan and Song Fu. Adaptive Anomaly Identification by Exploring Metric Subspace in Cloud Computing Infrastructures. In *IEEE 32nd International Symposium on Reliable Distributed Systems*, pages 205–214, Braga, Portugal, September 2013. IEEE.

[70] Qiang Guan, Song Fu, Nathan DeBardeleben, and Sean Blanchard. Exploring Time and Frequency Domains for Accurate and Automated Anomaly Detection in Cloud Computing Systems. In *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, pages 196–205, Vancouver, BC, Canada, December 2013. IEEE.

[71] Qiang Guan, Ziming Zhang, and Song Fu. Proactive Failure Management by Integrated Unsupervised and Semi-Supervised Learning for Dependable Cloud Systems. In *Sixth International Conference on Availability, Reliability and Security*, pages 83–90, Vienna, Austria, August 2011. IEEE.

[72] Qiang Guan, Ziming Zhang, and Song Fu. Ensemble of Bayesian Predictors and Decision Trees for Proactive Failure Management in Cloud Computing Systems. *Journal of Communications*, 7(1):52–61, January 2012.

[73] Zijie Guan, Jinjin Lin, and Pengfei Chen. On Anomaly Detection and Root Cause Analysis of Microservice Systems. In *Service-Oriented Computing – ICSOC 2018 Workshops*, volume 11434, pages 465–469. Springer International Publishing, 2019. Series Title: Lecture Notes in Computer Science.

[74] Tanja Hagemann and Katerina Katsarou. Reconstruction-based anomaly detection for the cloud: A comparison on the Yahoo! Webscope S5 dataset. In *Proceedings of the 2020 4th International Conference on Cloud and Big Data Computing*, pages 68–75, Virtual United Kingdom, August 2020. ACM.

[75] Tanja Hagemann and Katerina Katsarou. A Systematic Review on Anomaly Detection for Cloud Computing Environments. In *3rd Artificial Intelligence and Cloud Computing Conference*, AICCC 2020, pages 83–96, New York, NY, USA, March 2021. Association for Computing Machinery.

[76] Ahmed Hany Fawzy, Khaled Wassif, and Hanan Moussa. Framework for automatic detection of anomalies in DevOps. *Journal of King Saud University - Computer and Information Sciences*, 35(3):8–19, March 2023.

[77] Wilhelm Hasselbring. Software architecture: Past, present, future. In *The Essence of Software Engineering*, pages 169–184. Springer International Publishing, 2018.

[78] Jingzhu He, Yuhang Lin, Xiaohui Gu, Chin-Chia Michael Yeh, and Zhongfang Zhuang. PerfSig: extracting performance bug signatures via multi-modality causal analysis. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1669–1680, Pittsburgh Pennsylvania, May 2022. ACM.

[79] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. Identifying impactful service system problems via log analysis. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, pages 60–70, Lake Buena Vista, FL, USA, 2018. ACM Press.

[80] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. Experience Report: System Log Analysis for Anomaly Detection. In *IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218, Ottawa, ON, Canada, October 2016. IEEE.

[81] Zilong He, Pengfei Chen, Xiaoyun Li, Yongfeng Wang, Guangba Yu, Cailin Chen, Xinrui Li, and Zibin Zheng. A Spatiotemporal Deep Learning Approach for Unsupervised Anomaly Detection in Cloud Systems. *IEEE Transactions on Neural Networks and Learning Systems*, 34(4):1705–1719, April 2023.

[82] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[83] Jonny Holmström. From AI to digital transformation: The AI readiness framework. *Business Horizons*, 65(3):329–339, May 2022.

[84] Bin Hong, Yazhou Hu, Fuyang Peng, and Bo Deng. Distributed State Monitoring for IaaS Cloud with Continuous Observation Sequence. In *IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and IEEE 12th Intl Conf on Autonomic and Trusted Computing and IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 1037–1042, Beijing, August 2015. IEEE.

[85] Bin Hong, Fuyang Peng, Bo Deng, Yazhou Hu, and Dongxia Wang. DAC-Hmm: detecting anomaly in cloud systems with hidden Markov models. *Concurrency and Computation: Practice and Experience*, 27(18):5749–5764, December 2015.

[86] Bin Hong, Fuyang Peng, Bo Deng, and Yuchao Zhang. O-MAP: A per-component online anomaly predicting method for Cloud infrastructure. In *IEEE International Conference on Information and Automation*, pages 3026–3031, Lijiang, China, August 2015. IEEE.

[87] A. Hrusto, E. Engstrom, and P. Runeson. Optimization of anomaly detection in a microservice system through continuous feedback from development. In *IEEE/ACM 10th International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems (SESoS)*, pages 13–20, Los Alamitos, CA, USA, may 2022. IEEE Computer Society.

[88] Adha Hrusto, Emelie Engström, and Per Runeson. Towards optimization of anomaly detection in DevOps. *Information and Software Technology*, 160:107241, August 2023.

[89] Adha Hrusto, Per Runeson, and Emelie Engström. Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations. *SN Computer Science*, 2(6):447, August 2021.

[90] Adha Hrusto, Per Runeson, and Magnus C Ohlsson. Autonomous monitors for detecting failures early and reporting interpretable alerts in cloud operations. In *IEEE/ACM 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2024.

[91] Ruei-Jie Hsieh, Jerry Chou, and Chih-Hsiang Ho. Unsupervised Online Anomaly Detection on Multivariate Sensing Time Series Data for Smart Manufacturing. In *IEEE 12th SOCA Conference*, pages 90–97, Kaohsiung, Taiwan, November 2019. IEEE.

[92] Chengqiang Huang, Geyong Min, Yulei Wu, Yiming Ying, Ke Pei, and Zuochang Xiang. Time Series Anomaly Detection for Trustworthy Services in Cloud Computing Systems. *IEEE Transactions on Big Data*, 8(1):60–72, February 2022.

[93] Shaohan Huang, Carol Fung, Chang Liu, Shupeng Zhang, Guang Wei, Zhongzhi Luan, and Depei Qian. Arena: Adaptive real-time update anomaly prediction in cloud systems. In *13th International Conference on Network and Service Management (CNSM)*, pages 1–9, Tokyo, November 2017. IEEE.

[94] Tao Huang, Pengfei Chen, and Ruipeng Li. A Semi-Supervised VAE Based Active Anomaly Detection Framework in Multivariate Time Series for Online Systems. In *Proceedings of the ACM Web Conference 2022*, pages 1797–1806, Virtual Event, Lyon France, April 2022. ACM.

[95] Fabian Huch, Mojdeh Golagha, Ana Petrovska, and Alexander Krauss. Machine learning-based run-time anomaly detection in software systems: An industrial evaluation. In *IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, pages 13–18, March 2018.

[96] Mohammad S. Islam, William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. Anomaly detection in a large-scale cloud platform. In *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '21, page 150–159. IEEE Press, 2021.

[97] Mohammad Saiful Islam and Andriy Miranskyy. Anomaly detection in cloud components. In *IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 1–3, 2020.

[98] Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski. Performance Monitoring and Root Cause Analysis for Cloud-hosted Web Applications. In *Proceedings of the 26th International Conference on World Wide Web*, pages 469–478, Perth Australia, April 2017. International World Wide Web Conferences Steering Committee.

[99] Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski. Detecting Performance Anomalies in Cloud Platform Applications. *IEEE Transactions on Cloud Computing*, 8(3):764–777, July 2020.

[100] M.P.G.K. Jayaweera, W.M.C.J.T. Kithulwatta, and R.M.K.T. Rathnayaka. Detect anomalies in cloud platforms by using network data: a review. *Cluster Computing*, 26(5):3279 – 3289, 2023.

[101] Tong Jia, Yifan Wu, Chuanjia Hou, and Ying Li. LogFlash: Real-time Streaming Anomaly Detection and Diagnosis from System Logs for Large-scale Software Systems. In *IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 80–90, Wuhan, China, October 2021. IEEE.

[102] Tong Jia, Lin Yang, Pengfei Chen, Ying Li, Fanjing Meng, and Jingmin Xu. LogSed: Anomaly Diagnosis through Mining Time-Weighted Control Flow Graph in Logs. In *IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 447–455, Honolulu, CA, USA, June 2017. IEEE.

[103] Wenqian Jiang, Yang Hong, Beitong Zhou, Xin He, and Cheng Cheng. A gan-based anomaly detection approach for imbalanced industrial time series. *IEEE Access*, 7:143608–143619, 2019.

[104] Ying Jiang, Na Zhang, and Zheng Ren. Research on intelligent monitoring scheme for microservice application systems. In *International Conference on Intelligent Transportation, Big Data Smart City (ICITBS)*, pages 791–794, Vientiane, Laos, 2020. IEEE.

[105] Javier Jose Diaz Rivera, Talha Ahmed Khan, Waleed Akbar, Muhammad Afaq, and Wang-Cheol Song. An ML Based Anomaly Detection System in real-time data streams. In *International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1329–1334, Las Vegas, NV, USA, December 2021. IEEE.

[106] Parisa Sadat Kalaki, Alireza Shameli-Sendi, and Behzad Khalaji Emamzadeh Abbasi. Anomaly detection on OpenStack logs based on an improved robust principal component analysis model and its projection onto column space. *Software: Practice and Experience*, 53(3):665–681, March 2023.

[107] Yasuhiko Kanemasa, Shuji Suzuki, Atsushi Kubota, and Junichi Higuchi. Single-View Performance Monitoring of On-Line Applications Running on a Cloud. In *IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 350–358, Honolulu, CA, USA, June 2017. IEEE.

[108] Phanindra Reddy Kannari, Noorullah Shariff Chowdary, and Rajkumar Laxmikanth Biradar. An anomaly-based intrusion detection system using recursive feature elimination technique for improved attack detection. *Theoretical Computer Science*, 931:56–64, 2022.

[109] Yıldız Karadayı, Mehmet N. Aydin, and A. Selçuk Öğrenci. A Hybrid Deep Learning Framework for Unsupervised Anomaly Detection in Multivariate Spatio-Temporal Data. *Applied Sciences*, 10(15):5191, July 2020.

[110] George K. Karagiannidis and Athanasios S. Lioumpas. An improved approximation for the gaussian q-function. *IEEE Communications Letters*, 11(8):644–646, 2007.

[111] Sara Kardani-Moghaddam, Rajkumar Buyya, and Kotagiri Ramamohanarao. Performance anomaly detection using isolation-trees in heterogeneous workloads of web applications in computing clouds. *Concurrency and Computation: Practice and Experience*, 31(20):e5306, October 2019.

[112] Mark Kasunic. Designing an effective survey. Technical Report CMU/SEI-2005-HB-004, Carnegie Mellon Software Engineering Institute, 2005.

[113] Mohammad Khanahmadi, Alireza Shameli-Sendi, Masoume Jabbarifar, Quentin Fournier, and Michel Dagenais. Detection of microservice-based software anomalies based on OpenTracing in cloud. *Software: Practice and Experience*, 53(8):1681–1699, August 2023.

[114] Farzaneh Khoshnevisan, Zhewen Fan, and Vitor R. Carvalho. Improving robustness on seasonality-heavy multivariate time series anomaly detection. *CoRR*, abs/2007.14254, 2020.

[115] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. Outlier Detection for Time Series with Recurrent Autoencoder Ensembles. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 2725–2732, Macao, China, August 2019. International Joint Conferences on Artificial Intelligence Organization.

[116] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. *Evidence-Based Software Engineering and Systematic Reviews*. Chapman & Hall/CRC, 2015.

[117] Iman Kohyarnejadfard, Daniel Aloise, Seyed Vahid Azhari, and Michel R. Dagenais. Anomaly detection in microservice environments using distributed tracing data analysis and NLP. *Journal of Cloud Computing*, 11(1):25, August 2022.

[118] Joanna Kosińska, Bartosz Baliś, Marek Konieczny, Maciej Malawski, and Sławomir Zieliński. Toward the observability of cloud-native applications: The overview of the state-of-the-art. *IEEE Access*, 11:73036–73052, 2023.

[119] Nane Kratzke and Peter-Christian Quint. Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study. *Journal of Systems and Software*, 126:1–16, 2017.

[120] Abhishek Kumar, Tristan Braud, Sasu Tarkoma, and Pan Hui. Trustworthy ai in the age of pervasive computing and big data. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 1–6, Austin, TX, USA, 2020. IEEE.

[121] Giacomo Lanciano, Antonio Ritacco, Fabio Brau, Tommaso Cucinotta, Marco Vannucci, Antonino Artale, Joao Barata, and Enrica Sposato. Using Self-Organizing Maps for the Behavioral Analysis of Virtualized Network Functions. In Donald Ferguson, Claus Pahl, and Markus Helfert, editors, *Cloud Computing and Services Science*, volume 1399, pages 153–177. Springer International Publishing, 2021. Series Title: Communications in Computer and Information Science.

[122] Eero Laukkanen, Juha Itkonen, and Casper Lassenius. Problems, causes and solutions when adopting continuous delivery—A systematic literature review. *Information and Software Technology*, 82:55–79, February 2017.

[123] Chang-Ki Lee, Yu-Jeong Cheon, and Wook-Yeon Hwang. Studies on the GAN-Based Anomaly Detection Methods for the Time Series Data. *IEEE Access*, 9:73201–73215, 2021.

[124] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Yongqiang Yang, and Michael R. Lyu. Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention. In *IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1724–1736, 2023.

[125] Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In *Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series*, pages 703–716. Springer, 2019.

[126] Min Li, Dingyong Tang, Zepeng Wen, and Yunchang Cheng. Universal Anomaly Detection Method Based on Massive Monitoring Indicators of Cloud Platform. In *IEEE International Conference on Software Engineering and Artificial Intelligence (SEAI)*, pages 23–29, Xiamen, China, June 2021. IEEE.

[127] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. SwissLog: Robust Anomaly Detection and Localization for Interleaved Unstructured Logs. *IEEE Transactions on Dependable and Secure Computing*, 20(4):2762–2780, July 2023.

[128] Yuewei Li, Yan Lu, Jingyu Wang, Qi Qi, Jing Wang, Yingying Wang, and Jianxin Liao. TADL: Fault Localization with Transformer-based Anomaly Detection for Dynamic Microservice Systems. In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 718–722, Taipa, Macao, March 2023. IEEE.

[129] Yufeng Li, Guangba Yu, Pengfei Chen, Chuanfu Zhang, and Zibin Zheng. MicroSketch: Lightweight and Adaptive Sketch Based Performance Issue Detection and Localization in Microservice Systems. In *Service-Oriented Computing*, volume 13740, pages 219–236. Springer Nature Switzerland, 2022. Series Title: Lecture Notes in Computer Science.

[130] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. Copod: Copula-based outlier detection. In *IEEE International Conference on Data Mining (ICDM)*, pages 1118–1123. IEEE, 09 2020.

[131] Derek Lin, Rashmi Raghu, Vivek Ramamurthy, Jin Yu, Regunathan Radhakrishnan, and Joseph Fernandez. Unveiling clusters of events for alert and incident management in large-scale enterprise it. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '14*, pages 1630–1639, New York, New York, USA, 2014. ACM Press.

[132] Mingwei Lin and Zhiqiang Yao. Toward Anomaly Detection in IaaS Cloud Computing Platforms. *International Journal of Security and Its Applications*, 9(12):175–188, December 2015.

[133] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16*, pages 102–111, Austin, Texas, 2016. ACM Press.

[134] Ying Lin, Zhengzhang Chen, Cheng Cao, Lu-An Tang, Kai Zhang, Wei Cheng, and Zhichun Li. Collaborative Alert Ranking for Anomaly Detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1987–1995, Torino Italy, October 2018. ACM.

[135] Benjamin Lindemann, Benjamin Maschler, Nada Sahlab, and Michael Weyrich. A survey on anomaly detection for technical systems using lstm networks. *Computers in Industry*, 131:103498, 2021.

[136] Johan Linåker, Sardar Sulaman, Martin Host, and Rafael de Mello. Guidelines for conducting surveys in software engineering. Technical report, Lund University, May 2015.

[137] Dapeng Liu, Dan Pei, and Youjian Zhao. Application-aware latency monitoring for cloud tenants via CloudWatch+. In *10th International Conference on Network and Service Management (CNSM) and Workshop*, pages 73–81, Rio de Janeiro, Brazil, November 2014. IEEE.

[138] Jianwei Liu, Hongwei Zhu, Yongxia Liu, Haobo Wu, Yunsheng Lan, and Xinyu Zhang. Anomaly detection for time series using temporal convolutional networks and Gaussian mixture model. *Journal of Physics: Conference Series*, 1187(4):042111, April 2019.

[139] Jinyang Liu, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Cong Feng, Zengyin Yang, and Michael R. Lyu. Practical Anomaly Detection over Multivariate Monitoring Metrics for Online Services, August 2023.

[140] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. In *IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 48–58, Coimbra, Portugal, October 2020. IEEE.

[141] Shirong Liu, Xiong Chen, Xingxiong Peng, and Ruliang Xiao. Network log anomaly detection based on gru and svdd. In *IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 1244–1249, 2019.

[142] Yuecan Liu, Jiangang Sun, Yuzhu Chang, Qingfu Yang, Linwei Yang, and Jing Li. Service-based cloud platform application monitoring analysis. In *IEEE 6th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 799–803, Beijing, China, October 2022. IEEE.

[143] Zhuo Long, Xiaofei Zhang, Li Zhang, Guojun Qin, Shoudao Huang, Dianyi Song, Haidong Shao, and Gongping Wu. Motor fault diagnosis using attention mechanism and improved adaboost driven by multi-sensor information. *Measurement*, 170:108718, 2021.

[144] Matthieu Lucke, Moncef Chioua, Chriss Grimholt, Martin Hollender, and Nina F. Thornhill. Integration of alarm design in fault detection and diagnosis through alarm-range normalization. *Control Engineering Practice*, 98:104388, May 2020.

[145] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. Dimensions of DevOps. In Casper Lassenius, Torgeir Dingsøyr, and Maria Paasivaara, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 212–217. Springer International Publishing, 2015.

[146] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. An exploratory study of DevOps: Extending the dimensions of devops with practices. In *The Eleventh International Conference on Software Engineering Advances (ICSEA)*, Rome, Italy, 08 2016.

[147] Joao Paulo Magalhaes and Luis Moura Silva. A Framework for Self-Healing and Self-Adaptation of Cloud-Hosted Web-Based Applications. In *IEEE 5th International Conference on Cloud Computing Technology and Science*, pages 555–564, Bristol, United Kingdom, December 2013. IEEE.

[148] Sepehr Maleki, Sasan Maleki, and Nicholas R. Jennings. Unsupervised anomaly detection with LSTM autoencoders using statistical data-filtering. *Applied Soft Computing*, 108:107443, September 2021.

[149] Leonardo Mariani, Mauro Pezzè, Oliviero Riganelli, and Rui Xin. Predicting failures in multi-tier distributed systems. *Journal of Systems and Software*, 161:110464, March 2020.

[150] Lun Meng, Feng Ji, Yao Sun, and Tao Wang. Detecting anomalies in microservices with execution trace comparison. *Future Generation Computer Systems*, 116:291–301, March 2021.

[151] Haibo Mi, Huaimin Wang, Zhenbang Chen, and Yangfan Zhou. Automatic Detecting Performance Bugs in Cloud Computing Systems via Learning Latency Specification Model. In *IEEE 8th International Symposium on Service Oriented System Engineering*, pages 302–307, Oxford, United Kingdom, April 2014. IEEE.

[152] HaiBo Mi, HuaiMin Wang, YangFan Zhou, Michael R. Lyu, and Hua Cai. Localizing root causes of performance anomalies in cloud computing systems by analyzing request trace logs. *Science China Information Sciences*, 55(12):2757–2773, December 2012.

[153] Alok Mishra and Ziadoon Otaiwi. Devops and software quality: A systematic mapping. *Comput. Sci. Rev.*, 38:100308, 2020.

[154] Sara Kardani Moghaddam, Rajkumar Buyya, and Kotagiri Ramamohanarao. ACAS: An anomaly-based cause aware auto-scaling framework for clouds. *Journal of Parallel and Distributed Computing*, 126:107–120, April 2019.

[155] Jefferson Seide Molléri, Kai Petersen, and Emilia Mendes. An empirically evaluated checklist for surveys in software engineering. *Information and Software Technology*, 119:106240, March 2020.

[156] Saad Mubeen, Sara Abbaspour Asadollah, Alessandro Vittorio Papadopoulos, Mohammad Ashjaei, Hongyu Pei-Breivold, and Moris Behnam. Management of Service Level Agreements for Cloud Services in IoT: A Systematic Mapping Study. *IEEE Access*, 6:30184–30207, 2018.

[157] Joydeep Mukherjee, Alexandru Baluta, Marin Litoiu, and Diwakar Krishnamurthy. RAD: Detecting Performance Anomalies in Cloud-based Web Services. In *IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 493–501, Beijing, China, October 2020. IEEE.

[158] Sankha Subhra Mullick, Shounak Datta, and Swagatam Das. Adaptive learning-based k -nearest neighbor classifiers with resilience to class imbalance. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5713–5725, 2018.

[159] Mohsin Munir, Shoaib Ahmed Siddiqui, Andreas Dengel, and Sheraz Ahmed. DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. *IEEE Access*, 7:1991–2005, 2019.

[160] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. Anomaly Detection from System Tracing Data Using Multimodal Deep Learning. In *IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 179–186, Milan, Italy, July 2019. IEEE.

[161] Isaac Odun-Ayo, Toro-Abasi Williams, and Jamaiah Yahaya. Cloud management and monitoring - a systematic mapping study. *Indonesian Journal of Electrical Engineering and Computer Science*, 21(3):1648–1662, March 2021. Number: 3.

[162] OpenAI. Gpt-3: Language models are few-shot learners, 2020.

[163] Alina Oprea, Zhou Li, Ting-Fang Yen, Sang H. Chin, and Sumayah Alrwais. Detection of early-stage enterprise infection by mining large-scale log data. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 45–56, 2015.

[164] Alessandro Orso, Donglin Liang, Mary Jean Harrold, and Richard Lipton. Gamma system: Continuous evolution of software after deployment. *SIGSOFT Softw. Eng. Notes*, 27(4):65–69, July 2002.

[165] David O'Shea, Vincent C. Emeakaroha, Neil Cafferkey, John P. Morrison, and Theo Lynn. Detecting Anomaly in Cloud Platforms Using a Wavelet-Based Framework. In Markus Helfert, Donald Ferguson, Victor Méndez Muñoz, and Jorge Cardoso, editors, *Cloud Computing and Services Science*, volume 740, pages 131–150. Springer International Publishing, 2017. Series Title: Communications in Computer and Information Science.

[166] Yicheng Pan, Meng Ma, Xinrui Jiang, and Ping Wang. Faster, deeper, easier: crowdsourcing diagnosis of microservice kernel failure from user space. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 646–657, Virtual Denmark, July 2021. ACM.

[167] Husanbir S. Pannu, Jianguo Liu, and Song Fu. AAD: Adaptive Anomaly Detection System for Cloud Computing Infrastructures. In *IEEE 31st Symposium on Reliable Distributed Systems*, pages 396–397, Irvine, CA, USA, October 2012. IEEE.

[168] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[169] Manjula Peiris, James H. Hill, Jorgen Thelin, Sergey Bykov, Gabriel Kliot, and Christian Konig. PAD: Performance Anomaly Detection in Multi-server Distributed Systems. In *IEEE 7th International Conference on Cloud Computing*, pages 769–776, Anchorage, AK, USA, June 2014. IEEE.

[170] Roberto Pietrantuono, Antonia Bertolino, Guglielmo De Angelis, Breno Miranda, and Stefano Russo. Towards Continuous Software Reliability Testing in DevOps. In *IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, pages 21–27, Montreal, QC, Canada, May 2019. IEEE.

[171] William Pourmajidi, John Steinbacher, Tony Erwin, and Andriy Miranskyy. On challenges of cloud monitoring. In *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering*, CASCON '17, page 259–265, USA, 2017. IBM Corp.

[172] Jiaxing Qi, Zhongzhi Luan, Shaohan Huang, Carol Fung, Hailong Yang, Hanlu Li, Danfeng Zhu, and Depei Qian. LogEncoder: Log-Based Contrastive Representation Learning for Anomaly Detection. *IEEE Transactions on Network and Service Management*, 20(2):1378–1391, June 2023.

[173] Juan Qiu, Qingfeng Du, and Chongshu Qian. KPI-TSAD: A Time-Series Anomaly Detector for KPI Monitoring in Cloud Applications. *Symmetry*, 11(11):1350, November 2019.

[174] Rakesh Rana, Miroslaw Staron, Jörgen Hansson, Martin Nilsson, and Wilhelm Meding. A Framework for Adoption of Machine Learning in Industry for Software Defect Prediction. In *Proceedings of the 9th International Conference on Software Engineering and Applications*, pages 383–392, Vienna, Austria, 2014. SCITEPRESS - Science and and Technology Publications.

[175] Rui Ren, Yang Wang, Fengrui Liu, Zhenyu Li, and Gaogang Xie. Triple:the interpretable deep learning anomaly detection framework based on trace-metric-log of microservice. In *IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2023.

[176] Oliviero Riganelli, Paolo Saltarel, Alessandro Tundo, Marco Mobilio, and Leonardo Mariani. Cloud Failure Prediction with Hierarchical Temporal Memory: An Empirical Assessment. In *20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 785–790, Pasadena, CA, USA, December 2021. IEEE.

[177] Pilar Rodríguez, Alireza Haghighatkhah, Lucy Ellen Lwakatare, Susanna Teppola, Tanja Suomalainen, Juho Eskeli, Teemu Karvonen, Pasi Kuvaja, June M. Verner, and Markku Oivo. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, 123:263–291, January 2017.

[178] Derrick Rountree and Ileana Castrillo. Chapter 3 - cloud deployment models. In Derrick Rountree and Ileana Castrillo, editors, *The Basics of Cloud Computing*, pages 35–47. Syngress, Boston, 2014.

[179] Sudip Roy, Arnd Christian Konig, Igor Dvorkin, and Manish Kumar. PerfAugur: Robust diagnostics for performance anomalies in cloud services. In *IEEE 31st International Conference on Data Engineering*, pages 1167–1178, Seoul, South Korea, April 2015. IEEE.

[180] Shaolun Ruan, Yong Wang, Hailong Jiang, Weijia Xu, and Qiang Guan. BatchLens: A Visualization Approach for Analyzing Batch Jobs in Cloud Systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 108–111, Antwerp, Belgium, March 2022. IEEE.

[181] Lukas Ruff, Jacob R. Kauffmann, Robert A. Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G. Dietterich, and Klaus-Robert Müller. A Unifying Review of Deep and Shallow Anomaly Detection. *Proceedings of the IEEE*, 109(5):756–795, May 2021.

[182] Per Runeson, Emelie Engström, and Margaret-Anne Storey. The design science paradigm as a frame for empirical software engineering. In *Contemporary Empirical Methods in Software Engineering*, pages 127–147. Springer, 2020.

[183] Areeg Samir and Claus Pahl. DLA: Detecting and Localizing Anomalies in Containerized Microservice Architectures Using Markov Models. In *7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 205–213, Istanbul, Turkey, August 2019. IEEE.

[184] Carla Sauvanaud, Mohamed Kaâniche, Karama Kanoun, Kahina Lazri, and Guthemberg Da Silva Silvestre. Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. *Journal of Systems and Software*, 139:84–106, 2018.

[185] Carla Sauvanaud, Mohamed Kaâniche, Karama Kanoun, Kahina Lazri, and Guthemberg Da Silva Silvestre. Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. *Journal of Systems and Software*, 139:84–106, May 2018.

[186] Carla Sauvanaud, Guthemberg Silvestre, Mohamed Kaaniche, and Karama Kanoun. Data Stream Clustering for Online Anomaly Detection in Cloud Applications. In *11th European Dependable Computing Conference (EDCC)*, pages 120–131, Paris, France, September 2015. IEEE.

[187] Florian Schmidt, Florian Suri-Payer, Anton Gulenko, Marcel Wallschlager, Alexander Acker, and Odej Kao. Unsupervised Anomaly Event Detection for VNF Service Monitoring Using Multivariate Online Arima. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 278–283, Nicosia, December 2018. IEEE.

[188] Syed Yousaf Shah, Zengwen Yuan, Songwu Lu, and Petros Zerfos. Dependency analysis of cloud applications for performance monitoring using recurrent neural networks. In *IEEE International Conference on Big Data (Big Data)*, pages 1534–1543, Boston, MA, December 2017. IEEE.

[189] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, 5:3909–3943, 2017.

[190] Lifeng Shen, Zhuocong Li, and James Kwok. Timeseries anomaly detection using temporal hierarchical one-class network. In *Advances in Neural Information Processing Systems*, volume 33, pages 13016–13026. Curran Associates, Inc., 2020.

[191] Jayalaxmi P Shetty and Rajesh Panda. An overview of cloud computing in SMEs. *Journal of Global Entrepreneurship Research*, 11(1):175–188, December 2021.

[192] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. Anomaly Detection in Streams with Extreme Value Theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1067–1075, Halifax NS Canada, August 2017. ACM.

[193] Jonathan Sillito and Esdras Kutomi. Failures and Fixes: A Study of Software System Incident Response. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 185–195, Adelaide, SA, Australia, September 2020. IEEE.

[194] Derek Smith, Qiang Guan, and Song Fu. An Anomaly Detection Framework for Autonomic Management of Compute Cloud Systems. In *IEEE 34th Annual Computer Software and Applications Conference Workshops*, pages 376–381, Seoul, Korea (South), July 2010. IEEE.

[195] Jacopo Soldani and Antonio Brogi. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Computing Surveys*, 55(3):59:1–59:39, February 2022.

[196] Gagan Somashekar, Anurag Dutt, Rohith Vaddavalli, Sai Bhargav Varanasi, and Anshul Gandhi. B-MEG: Bottlenecked-Microservices Extraction Using Graph Neural Networks. In *Companion of the ACM/SPEC International Conference on Performance Engineering*, pages 7–11, Bejing China, July 2022. ACM.

[197] Youmei Song, Chaoran Li, Kuoran Zhuang, Tianjiao Ma, and Tianyu Wo. An Automatic Scaling System for Online Application with Microservices Architecture. In *IEEE International Conference on Joint Cloud Computing (JCC)*, pages 73–78, Fremont, CA, USA, August 2022. IEEE.

[198] Song Fu. Performance Metric Selection for Autonomic Anomaly Detection on Cloud Computing Systems. In *IEEE Global Telecommunications Conference - GLOBECOM 2011*, pages 1–5, Houston, TX, USA, December 2011. IEEE.

[199] Daniel Ståhl, Torvald Mårtensson, and Jan Bosch. Continuous practices and devops: Beyond the buzz, what does it all mean? In *43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 440–448, Vienna, September 2017. IEEE.

[200] Margaret-Anne Storey, Neil A. Ernst, Courtney Williams, and Eirini Kalliamvakou. The who, what, how of software engineering research: a socio-technical framework. *Emp. Softw. Eng.*, 25(5):4097–4129, 2020.

[201] Daniel Sun, Min Fu, Liming Zhu, Guoqiang Li, and Qinghua Lu. Non-Intrusive Anomaly Detection With Streaming Performance Metrics and Logs for DevOps in Public Clouds: A Case Study in AWS. *IEEE Transactions on Emerging Topics in Computing*, 4(2):278–289, April 2016.

[202] Sampo Suonsyrjä, Laura Hokkanen, Henri Terho, Kari Systä, and Tommi Mikkonen. Post-Deployment Data: A Recipe for Satisfying Knowledge Needs in Software Development? In *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, pages 139–147, Berlin, Germany, October 2016. IEEE.

[203] Chellammal Surianarayanan and Pethuru Raj Chelliah. *Essentials of Cloud Computing: A Holistic, Cloud-Native Perspective*. Texts in Computer Science. Springer International Publishing, 2023.

[204] Aboozar Taherkhani, Georgina Cosma, and T.M. McGinnity. Adaboost-cnn: An adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning. *Neurocomputing*, 404:351–366, 2020.

[205] Damian A. Tamburri, Marco Miglierina, and Elisabetta Di Nitto. Cloud applications monitoring: An industrial study. *Information and Software Technology*, 127:106376, November 2020.

[206] Yongmin Tan, Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Chitra Venkatramani, and Deepak Rajan. PREPARE: Predictive Performance Anomaly Prevention for Virtualized Cloud Systems. In *IEEE 32nd International Conference on Distributed Computing Systems*, pages 285–294, Macau, China, June 2012. IEEE.

[207] Minaoar Hossain Tanzil, Masud Sarker, Gias Uddin, and Anindya Iqbal. A mixed method study of DevOps challenges. *Information and Software Technology*, 161:107244, September 2023.

[208] FreeWheel Biz-UI Team. Continuous Integration and Continuous Deployment. In *Cloud-Native Application Architecture*, pages 351–382. Springer Nature Singapore, Singapore, 2024.

[209] Pratik Thantharate. IntelligentMonitor: Empowering DevOps Environments with Advanced Monitoring and Observability. In *International Conference on Information Technology (ICIT)*, pages 800–805, Amman, Jordan, August 2023. IEEE.

[210] Stefan Throner, Heiko Hutter, Niklas Sanger, Michael Schneider, Simon Hanselmann, Patrick Petrovic, and Sebastian Abeck. An Advanced DevOps Environment for Microservice-based Applications. In *IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 134–143, Oxford, United Kingdom, August 2021. IEEE.

[211] Deepika Tiwari, Long Zhang, Martin Monperrus, and Benoit Baudry. Production monitoring to improve test suites. *IEEE Transactions on Reliability*, 71(3):1381–1397, 2021.

[212] Laszlo Toka, Gergely Dobreff, David Haja, and Mark Szalay. Predicting cloud-native application failures based on monitoring data of cloud infrastructure. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 842–847, 2021.

[213] Luca Traini and Vittorio Cortellessa. DeLag: Using Multi-Objective Optimization to Enhance the Detection of Latency Degradation Patterns in Service-Based Systems. *IEEE Transactions on Software Engineering*, pages 1–28, 2023.

[214] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. Tranad: deep transformer networks for anomaly detection in multivariate time series data. *Proc. VLDB Endow.*, 15(6):1201–1214, February 2022.

[215] Aline Valente, Maristela Holanda, Ari Melo Mariano, Richard Furuta, and Dilma Da Silva. Analysis of Academic Databases for Literature Review in the Computer Science Education Field. In *IEEE Frontiers in Education Conference (FIE)*, pages 1–7, October 2022. ISSN: 2377-634X.

[216] Owen Vallis, Jordan Hochenbaum, and Arun Kejariwal. A novel technique for Long-Term anomaly detection in the cloud. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, Philadelphia, PA, June 2014. USENIX Association.

[217] Yiannis Verginadis. A review of monitoring probes for cloud computing continuum. In *Advanced Information Networking and Applications*, pages 631–643. Springer, 2023.

[218] Fotios Voutsas, John Violos, and Aris Leivadeas. Filtering Alerts on Cloud Monitoring Systems. In *IEEE International Conference on Joint Cloud Computing (JCC)*, pages 34–37, Athens, Greece, July 2023. IEEE.

[219] Fotios Voutsas, John Violos, and Aris Leivadeas. Mitigating Alert Fatigue in Cloud Monitoring Systems: A Machine Learning Perspective. *Computer Networks*, 250:110543, August 2024.

[220] Chao Wang and Zhongchuan Fu. Quantitative evaluation of fault propagation in a commercial cloud system. *International Journal of Distributed Sensor Networks*, 16:155014772090361, March 2020.

[221] GuiPing Wang, JianXi Yang, and Ren Li. UFKLDA: An unsupervised feature extraction algorithm for anomaly detection under cloud environment. *ETRI Journal*, 41(5):684–695, October 2019.

[222] Ke Wang and Hyong S. Kim. PCAD: Cloud Performance Anomaly Detection with Data Packet Counts. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 106–113, Hong Kong, December 2017. IEEE.

[223] Lu Wang, Yu Xuan Jiang, Zhan Wang, Qi En Huo, Jie Dai, Sheng Long Xie, Rui Li, Ming Tao Feng, Yue Shen Xu, and Zhi Ping Jiang. The operation and maintenance governance of microservices architecture systems: A

systematic literature review. *Journal of Software: Evolution and Process*, 35(10):e2433.

[224] Tao Wang, Jiwei Xu, Wenbo Zhang, Zeyu Gu, and Hua Zhong. Self-adaptive cloud monitoring with online anomaly detection. *Future Generation Computer Systems*, 80:89–101, March 2018.

[225] Tao Wang, Wenbo Zhang, Jun Wei, and Hua Zhong. Fault detection for cloud computing systems with correlation analysis. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 652–658, Ottawa, ON, Canada, May 2015. IEEE.

[226] Tao Wang, Wenbo Zhang, Jiwei Xu, and Zeyu Gu. Workflow-Aware Automatic Fault Diagnosis for Microservice-Based Applications With Statistics. *IEEE Transactions on Network and Service Management*, 17(4):2350–2363, December 2020.

[227] Tao Wang, Wenbo Zhang, Chunyang Ye, Jun Wei, Hua Zhong, and Tao Huang. FD4C: Automatic Fault Diagnosis Framework for Web Applications in Cloud Computing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(1):61–75, January 2016.

[228] Wenlu Wang, Pengfei Chen, Yibin Xu, and Zilong He. Active-MTSAD: Multivariate Time Series Anomaly Detection With Active Learning. In *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 263–274, Baltimore, MD, USA, June 2022. IEEE.

[229] Muhammad Waseem, Peng Liang, Mojtaba Shahin, Amleto Di Salle, and Gastón Márquez. Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, 182:111061, 2021.

[230] Billy Williams and Lester Hoel. Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results. *Journal of Transportation Engineering*, 129:664–672, 11 2003.

[231] Li Wu, Johan Tordsson, Jasmin Bogatinovski, Erik Elmroth, and Odej Kao. MicroDiag: Fine-grained Performance Diagnosis for Microservice Systems. In *IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*, pages 31–36, Madrid, Spain, May 2021. IEEE.

[232] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. MicroRCA: Root Cause Localization of Performance Issues in Microservices. In *NOMS IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, Budapest, Hungary, April 2020. IEEE.

[233] Zhe Xie, Haowen Xu, Wenxiao Chen, Wanxue Li, Huai Jiang, Liangfei Su, Hanzhang Wang, and Dan Pei. Unsupervised Anomaly Detection on Microservice Traces through Graph VAE. In *Proceedings of the ACM Web Conference 2023*, pages 2874–2884, Austin TX USA, April 2023. ACM.

[234] Ruyue Xin, Hongyun Liu, Peng Chen, and Zhiming Zhao. Robust and accurate performance anomaly detection and prediction for cloud applications: a novel ensemble learning-based framework. *Journal of Cloud Computing*, 12(1):7, January 2023.

[235] Jingmin Xu, Yuan Wang, Pengfei Chen, and Ping Wang. Lightweight and Adaptive Service API Performance Monitoring in Highly Dynamic Cloud Environment. In *IEEE International Conference on Services Computing (SCC)*, pages 35–43, Honolulu, HI, USA, June 2017. IEEE.

[236] Ke Xu, Yun Wang, Leni Yang, Yifang Wang, Bo Qiao, Si Qin, Yong Xu, Haidong Zhang, and Huamin Qu. CloudDet: Interactive Visual Analysis of Anomalous Performances in Cloud Computing Systems. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2019.

[237] Xiwei Xu, Liming Zhu, Min Fu, Daniel Sun, An Binh Tran, Paul Rimba, Srini Dwarakanathan, and Len Bass. Crying wolf and meaning it: Reducing false alarms in monitoring of sporadic operations through pod-monitor. *IEEE/ACM 1st International Workshop on Complex Faults & Failures in Large Software Systems (COUFLESS)*, pages 69 – 75, 2015.

[238] Tianyi Yang, Jiacheng Shen, Yuxin Su, Xiaoxue Ren, Yongqiang Yang, and Michael R. Lyu. Characterizing and Mitigating Anti-patterns of Alerts in Industrial Cloud Systems. In *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 393–401, Baltimore, MD, USA, June 2022. IEEE.

[239] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[240] Guangba Yu, Zicheng Huang, and Pengfei Chen. TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process*, 35(10):e2413, October 2023.

[241] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, Qingwei Lin, and Dongmei Zhang. Monitorassistant: Simplifying cloud service monitoring via large language models. In *Companion Proceedings of the*

32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, page 38–49, New York, NY, USA, 2024. Association for Computing Machinery.

[242] Ang Zhang, Xiaoyong Zhao, and Lei Wang. CNN and LSTM based Encoder-Decoder for Anomaly Detection in Multivariate Time Series. In *IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*, pages 571–575, Xi'an, China, October 2021. IEEE.

[243] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V. Chawla. A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:1409–1416, July 2019.

[244] Hancui Zhang, Jun Liu, and Tianshu Wu. Adaptive and incremental-clustering anomaly detection algorithm for vms under cloud platform runtime environment. *IEEE Access*, 6:76984–76992, 2018.

[245] Jing Zhang. Anomaly detecting and ranking of the cloud computing platform by multi-view learning. *Multimedia Tools and Applications*, 78(21):30923–30942, November 2019.

[246] Xiao Zhang, Fanjing Meng, Pengfei Chen, and Jingmin Xu. TaskInsight: A Fine-Grained Performance Anomaly Detection and Problem Locating System. In *IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 917–920, San Francisco, CA, USA, June 2016. IEEE.

[247] Xiao Zhang, Fanjing Meng, and Jingmin Xu. PerfInsight: A Robust Clustering-Based Abnormal Behavior Detection System for Large-Scale Cloud. In *IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 896–899, San Francisco, CA, USA, July 2018. IEEE.

[248] Xu Zhang, Qingwei Lin, Yong Xu, Si Qin, Hongyu Zhang, Bo Qiao, Yingnong Dang, Xinsheng Yang, Qian Cheng, Murali Chintalapati, Youjiang Wu, Ken Hsieh, Kaixin Sui, Xin Meng, Yaohai Xu, Wenchi Zhang, Furao Shen, and Dongmei Zhang. Cross-dataset time series anomaly detection for cloud systems. In *USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1063–1076, Renton, WA, July 2019. USENIX Association.

[249] Yuxin Zhang, Yiqiang Chen, Jindong Wang, and Zhiwen Pan. Unsupervised Deep Anomaly Detection for Multi-Sensor Time-Series Signals. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[250] Zhizhou Zhang, Murali Krishna Ramanathan, Prithvi Raj, Abhishek Parwal, Timothy Sherwood, and Milind Chabbi. CRISP: Critical path analysis of Large-Scale microservice architectures. In *USENIX Annual Technical Conference (USENIX ATC 22)*, pages 655–672, Carlsbad, CA, July 2022. USENIX Association.

[251] Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Multivariate time-series anomaly detection via graph attention network. In *IEEE International Conference on Data Mining (ICDM)*, pages 841–850, 2020.

[252] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, Yong Wu, Fang Zhou, Wenchi Zhang, Kaixin Sui, and Dan Pei. Understanding and handling alert storm for online service systems. In *IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 162–171, 2020.

[253] Nengwen Zhao, Panshi Jin, Lixin Wang, Xiaoqin Yang, Rong Liu, Wenchi Zhang, Kaixin Sui, and Dan Pei. Automatically and Adaptively Identifying Severe Alerts for Online Service Systems. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 2420–2429, Toronto, ON, Canada, July 2020. IEEE.

[254] Peihai Zhao, Xiaoyan Chang, and Mimi Wang. A Novel Multivariate Time-Series Anomaly Detection Approach Using an Unsupervised Deep Neural Network. *IEEE Access*, 9:109025–109041, 2021.

[255] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A Python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.