POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Improving the Infrastructures of AIOps by Bridging the Data Gap

ROOZBEH AGHILI
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Août 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Improving the Infrastructures of AIOps by Bridging the Data Gap

présentée par **Roozbeh AGHILI**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

DEDICATION

To all the scientists, explorers, and innovators who dedicated their lives to advancing human knowledge, improving our understanding of the Earth and the universe, and developing the technologies that move humanity forward.

ACKNOWLEDGEMENTS

Over the course of my doctoral studies, I had the chance to meet some incredible human beings. I am deeply grateful for their company, support, and moments of joy we shared during the past four years.

My sincere gratitude goes to my supervisor, Heng Li, truly the best advisor I could have asked for. Heng is not only a dedicated and knowledgeable researcher, but also a kind human being. He supported me both academically and personally, making this journey far smoother than I have expected. He even became (and still is!) the guarantor of my apartment, a gesture I will never forget. I would like to thank him for his constant support, for teaching me how to write, and above all, teaching me patience and persistency. He was always there for me, and consistently encouraged me at every step.

I would also like to extend my thanks to my co-supervisor, Foutse Khomh. His depth of knowledge, valuable resources, and broad perspective opened my eyes to the larger landscape of research and academic life. I am thankful not only for his technical knowledge and advice, but also for the interesting conversations we had that helped shape many of my decisions throughout this journey and beyond.

I am thankful to Qiaolin Qin and Xingfang Wu, my dear co-authors and lab mates, whose collaboration made research both enjoyable and productive. I am also grateful to Mehdi Morovati and Amin Ghadesi. Mehdi, our senior lab member, generously shared his experience and helped me navigate many aspects of research life. Amin was my close companion during coursework; we studied together and successfully made it through every class as a team.

I am also thankful to all the wonderful friends I met along the way. The memories we created, on weekends and even on busy weekdays, brought laughter and fun to this journey. May our friendships last long.

A very special thanks goes to Pegah, for her persistent support in all my choices, for always being by my side, and for the endless conversations and wonderful moments we continue to share.

And finally to my parents, Atiyeh and Alireza, my brother, Rambod, and my grandparents, Hoori and Ahmad, for their love, unconditional support, and amazing memories me have shared. They thought me how to think and shaped me into the person I am today, and for that, I will be forever grateful.

RÉSUMÉ

Pour faire face à la complexité et à l'échelle croissantes des environnements informatiques modernes, les organisations se tournent de plus en plus vers des solutions automatisées et axées sur les données pour gérer et améliorer la disponibilité et les performances de leurs systèmes logiciels. Ces dernières années, le concept d'*Intelligence Artificielle pour les Opérations Informatiques (AIOps)* a fait l'objet d'une attention particulière en tant qu'approche prometteuse et transformatrice. AIOps combine les technologies big data avec des techniques avancées d'*Intelligence Artificielle (IA)* et d'*Apprentissage Automatique (AA)* pour automatiser et améliorer les aspects clés des opérations informatiques. En minimisant les efforts manuels et en offrant une meilleure visibilité sur le comportement des systèmes, l'AIOps permet aux équipes informatiques de gérer plus efficacement des environnements complexes et distribués, tout en réduisant les coûts opérationnels et en améliorant la santé globale des systèmes.

Comme pour tout système basé sur l'IA ou le AA, l'efficacité des solutions AIOps dépend fortement de l'accès à des données de haute qualité, pertinentes et diversifiées. Ces données sont essentielles pour entraîner les modèles, évaluer les algorithmes et s'assurer que les solutions AIOps peuvent gérer la complexité et la variabilité des environnements réels. Cependant, malgré son importance, il y a une pénurie notable d'ensembles de données accessibles au public provenant d'opérations informatiques réelles, et un nombre croissant de recherches a mis en évidence les défis critiques liés aux données qui entravent les progrès de l'AIOps. Essentiellement, les données AIOps peuvent être obtenues par deux approches principales : (1) Générer des données synthétiques en simulant des scénarios du monde réel, ou (2) Acquérir des données opérationnelles du monde réel auprès d'organisations informatiques. Ces deux méthodes s'accompagnent de défis et de limites qui leur sont propres.

Cette thèse vise à remédier à la pénurie d'ensembles de données publiquement disponibles pour les AIOps en contribuant à la fois à la génération de données synthétiques et au partage de données préservant la vie privée. Tout d'abord, elle effectue une analyse empirique de la manière dont les données opérationnelles sont actuellement utilisées dans les solutions AIOps, en caractérisant les pipelines AIOps et en identifiant les lacunes. Deuxièmement, il caractérise les modèles et structures temporels communs dans les données télémétriques du monde réel que les travaux futurs sur la génération de données synthétiques réalistes pourront utiliser et reproduire. Troisièmement, il formalise une définition des données sensibles dans les journaux logiciels en se basant sur des perspectives réglementaires, universitaires et industrielles, contribuant ainsi à des pratiques respectueuses de la vie privée dans les opérations informatiques. Sur cette base, il présente SDLog, un cadre basé sur l'apprentissage profond qui surpasse les expressions régulières traditionnelles dans la détection des attributs sensibles, améliorant de manière significative la sécurité du partage des données et de l'analyse externe.

Collectivement, ces contributions ne font pas seulement progresser l'état de la recherche sur les AIOps, mais fournissent également des outils et des connaissances pratiques que les chercheurs et les praticiens peuvent adopter pour générer, partager et analyser les données AIOps de manière plus efficace et plus sûre.

ABSTRACT

To address the growing complexity and scale of modern IT environments, organizations are increasingly turning to automated, data-driven solutions to manage and improve the availability and performance of their software systems. In recent years, the concept of *Artificial Intelligence for IT Operations (AIOps)* has gained significant attention as a promising and transformative approach. AIOps combines big data technologies with advanced techniques in *Artificial Intelligence (AI)* and *Machine Learning (ML)* to automate and improve key aspects of IT operations. By minimizing manual effort and providing deeper visibility into system behavior, AIOps empowers IT teams to manage complex, distributed environments more efficiently while reducing operational overhead and improving overall system health.

As with any AI- or ML-based system, the effectiveness of AIOps solutions heavily depends on access to high-quality, relevant, and diverse data. This data is essential for training models, evaluating algorithms, and ensuring that AIOps solutions can handle the complexity and variability of real-world environments. However, despite its importance, there is a notable shortage of publicly available datasets drawn from actual IT operations, and a growing body of research has highlighted the critical data-related challenges hindering the progress of AIOps. In essence, AIOps data can be obtained through two main approaches: (1) Generating synthetic data by simulating real-world scenarios, or (2) Acquiring real-world operational data from IT organizations. Both methods come with their own unique challenges and limitations.

This thesis aims to address the shortage of publicly available datasets for AIOps by contributing to both synthetic data generation and privacy-preserving data sharing. First, it performs an empirical analysis of how operational data is currently used in AIOps solutions, characterizing AIOps pipelines and identifying gaps. Second, it characterizes common temporal patterns and structures in real-world telemetry data that future work in realistic synthetic data generation can use and replicate. Third, it formalizes a definition of sensitive data in software logs based on regulatory, academic, and industry perspectives, contributing to privacy-aware practices in IT operations. Building on this, it introduces SDLog, a deep learning-based framework that outperforms traditional regular expressions in detecting sensitive attributes, significantly improving the safety of data sharing and external analysis.

Collectively, these contributions not only advance the state of research in AIOps but also provide practical tools and knowledge that researchers and practitioners can adopt to generate, share, and analyze AIOps data more effectively and securely.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

AI	Artificial Intelligence
AIOps	Artificial Intelligence for IT Operations
IT	Information Technology
IT Ops	IT Operations
ML	Machine Learning
DL	Deep Learning
NLP	Natural Language Processing
CI/CD	Continuous Integration and Continuous Delivery
MTTD	Mean Time To Detect
MTTR	Mean Time To Resolution
SRE	Site Reliability Engineering
PII	Personally Identifiable Information
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act
CCPA	California Consumer Privacy Act
SLR	Systematic Literature Review
DevOps	Development and Operations
ANN	Artificial Neural Networks
GPU	Graphics Processing Units
BERT	Bidirectional Encoder Representations from Transformers
ITSM	Information Technology Service Management
SLA	Service Level Agreements

QoS	Quality of Service
CPU	Central Processing Unit
I/O	Input / Output
KPI	Key Performance Indicators
GMM	Gaussian Mixture Model
KDE	Kernel Density Estimator
GA	Genetic Algorithm
AE	Autoencoder
VAE	Variational Autoencoder
GAN	Generative Adversarial Network
PIPL	Personal Information Protection Law
DSL	Data Security Law
PIPEDA	Personal Information Protection and Electronic Documents Act
LGPD	General Data Protection Law
URL	Uniform Resource Locator
IP	Internet Protocol
NER	Named Entity Recognition
HMM	Hidden Markov Models
ME	Maximum Entropy
CRF	Conditional Random Fields
LSTM	Long Short-Term Memory
CNN	Convolutional neural network
MRC	Machine Reading Comprehension
PTM	Pre-trained Transformer-based Model

CD	Continuous Deployment
IoT	Internet of Things
MLOps	Machine Learning Operations
ITA	Internet Traffic Archive
TTL	Time-To-Live
CVE	Common Vulnerabilities and Exposures
PTM	Pre-trained Transformer-based Model
RQ	Research Question
QA	Question and Answer
SVM	Support Vector Machine
SOM	Self-Organizing Map
ARIMA	AutoRegressive Integrated Moving Average
EWMA	Exponentially Weighted Moving Average
DTR	Decision Tree Regression
WMA	Weighted Moving Average
DRL	Deep Reinforcement Learning
DE	Differential Evolution
KNN	K-Nearest Neighbors
PID	Proportional Integrative Derivative
CV	Coefficient of Variation
DTW	Dynamic Time Warping
IMDb	Internet Movie Database
EU	European Union
EEA	European Economic Area

ISMS	Information Security Management Systems
TCP	Transmission Control Protocol
TTL	Time To Live
MAC	Media Access Control
ICMP	Internet Control Message Protocol
TOS	Type of Service
PET	Privacy Enhancing Technologie
CAM	Clustering-based Anonymization Mechanism
LSH	Locality Sensitive Hashing
CTO	Chief Technology Officer
IOB	Inside-Outside-Beginning
LOOCV	Leave-One-Out Cross-Validation

CHAPTER 1 INTRODUCTION

1.1 Research Context

The field of software engineering has been rapidly evolving in recent years. Modern software systems have become large, highly complex, and distributed. Architectural paradigms such as microservices, cloud-native systems, and container orchestration platforms have enabled scalability, flexibility, and fault tolerance, but at the cost of increased system complexity. This complexity, in turn, has driven the need for continuous integration and deployment (*Continuous Integration and Continuous Delivery (CI/CD)*), real-time monitoring, and robust security practices to ensure reliability and maintainability. These changes have also led to a sharp rise in the number of software components and services that interact with each other, with each part handling specific tasks and often running in different environments. As a result, software systems now produce massive operational data, from system logs and performance metrics to network traces and user activity records. This data is generated at very high speeds, often in the range of millions of events per minute [1], and comes in many different formats and structures [2].

While this explosion of operational data provides many opportunities to improve the comprehension, reliability, and performance of software systems, it also creates serious challenges for *Information Technology (IT)* operations teams. Manual methods and traditional tools for monitoring, troubleshooting, and optimizing performance are no longer practical when systems generate so much data across thousands of components [3]. Using manual methods, tasks such as log analysis, anomaly detection, and predicting the performance of systems become inefficient and error-prone [4, 5]. This makes it harder to quickly detect issues, identify root causes, or prevent slowdowns before they affect users. In addition to the overwhelming volume of data, the heterogeneous and semi-structured nature of operational data adds another layer of difficulty [4]. For example, software logs often contain a mix of unstructured text and structured fields, with formats that differ significantly across applications and components. This lack of consistency makes it difficult to parse, understand, and analyze the data in a reliable and efficient way. The challenge is further amplified by the fact that software systems are constantly evolving; log formats and data schemas may change frequently, which can quickly make static analysis tools outdated [6]. As a result, monitoring and diagnostic systems require ongoing updates to keep pace with these changes. These factors make it increasingly difficult for IT teams to extract timely and accurate insights into the system's performance, reliability, and security.

To address the growing complexity and scale of modern IT environments, organizations are increasingly turning to automated, data-driven solutions to manage and improve the availability and performance of their software systems. In recent years, the concept of *AIOps* has gained significant attention as a promising and transformative approach. AIOps combines big data technologies with advanced techniques in *Artificial Intelligence (AI)* and *Machine Learning (ML)* to automate and improve key aspects of IT operations [7, 8]. Rather than relying on static rules or manual intervention, AIOps platforms are designed to continuously ingest, monitor, and analyze massive volumes of heterogeneous operational data, including software logs, traces, performance metrics, and events, in near real-time. A central goal of AIOps is to transit from traditional reactive operations towards a more proactive and predictive schema. This shift involves automatically detecting anomalies [9, 10], forecasting incidents before they impact users [11,12], identifying the root causes of performance or stability issues [13, 14], and recommending or executing corrective actions [15,16]. Modern AIOps solutions can correlate signals from multiple sources, filter out noise, recognize patterns over time, and adapt to changes in system behavior without constant reconfiguration [17]. In doing so, they help organizations reduce *Mean Time To Detect (MTTD)* and *Mean Time To Resolution (MTTR)*, streamline workflows, and improve the availability and reliability of services [17]. Moreover, AIOps is increasingly being integrated with *Development and Operations (DevOps)* and *Site Reliability Engineering (SRE)* practices, enabling closed-loop automation that not only alerts teams about issues but can also trigger self-healing workflows or update system configurations dynamically [4]. By minimizing manual effort and providing deeper visibility into system behavior, AIOps empowers IT teams to manage complex, distributed environments more efficiently while reducing operational overhead and improving overall system health.

As with any AI- or ML-based system, the effectiveness of AIOps solutions heavily depends on access to high-quality, relevant, and diverse data. This data is essential for training models, evaluating algorithms, and ensuring that AIOps solutions can handle the complexity and variability of real-world environments. However, despite its importance, there is a notable shortage of publicly available datasets drawn from actual IT operations. This lack of public data presents a major obstacle to the development and validation of AIOps techniques [18,19]. The core of this issue lies in the sensitive and proprietary nature of operational data. Software logs, traces, and performance metrics often contain confidential business information, system configurations, and user-related data such as *Internet Protocol (IP)* addresses, access credentials, or *Personally Identifiable Information (PII)* [20]. As a result, organizations are reluctant to share this data publicly, even in anonymized form. Concerns about security, privacy, and competitive advantage further discourage companies from contributing to shared datasets. This data scarcity creates a significant gap between academic research and industrial application. Researchers are often forced to rely on synthetic data small-scale testbeds that do not reflect the complexity, scale, or heterogeneity of production environments [4, 21]. Consequently, many proposed AIOps methods remain untested in realistic settings, limiting their generalizability and practical impact. The absence of standardized, benchmark datasets also makes it difficult to perform fair and reproducible evaluations across different approaches. Addressing this challenge requires a collaborative effort from both academia and industry to explore alternative solutions, such as privacy-preserving data sharing or the development of standardized simulation environments, that can facilitate research while protecting sensitive information. Bridging this data gap is essential for accelerating innovation and enabling robust, real-world deployments of AIOps technologies.

1.2 Problem Statement

A growing body of research has highlighted the critical data-related challenges hindering the progress of AIOps. Bogatinovski et al. [18] emphasize the absence of comprehensive and publicly available AIOps benchmarks, which prevents consistent evaluation and comparison of different approaches. Similarly, Dang et al. [7] point out that although major cloud providers collect vast amounts of telemetry data, often reaching terabytes or even petabytes each day/month, there remains a notable shortage of high-quality, diverse, and representative datasets suitable for developing and evaluating AIOps techniques. Several systematic studies and literature reviews further confirm that this scarcity of operational datasets is a widely acknowledged barrier in the field [4, 19, 21–24]. In particular, public datasets that reflect the complexity, variability, and scale of enterprise IT systems are rare, and those that do exist (e.g., Yahoo benchmark [25] or *Internet Traffic Archive (ITA)* [26]) are often limited in scope (i.e., are specific to one task) or outdated.

In essence, operational AIOps datasets can be obtained through two main approaches: (1) Generating synthetic data by simulating real-world scenarios, or (2) Acquiring real-world operational data from IT organizations. Both methods come with their own unique challenges and limitations.

Generating synthetic data. To address the limited availability of real-world datasets, researchers often rely on synthetic data generation [27, 28]. Synthetic datasets offer several advantages, such as enabling controlled and repeatable experiments, allowing researchers to simulate specific events or failure conditions, providing cost-effective access to large volumes of labeled samples, reducing privacy concerns, and reducing historical biases in the data [29, 30]. However, despite these benefits, synthetic data has notable limitations when it comes to reflecting the full complexity of real IT operational environments. Modern distributed systems involve complex dependencies, dynamic behaviors, and a mix of structured and unstructured telemetry sources [31]. Capturing this variety and variability through simulation is extremely challenging. Most synthetic datasets are generated using oversimplified models or assumptions, which often overlook key aspects of real-world systems [32]. As a result, they may exhibit biased data distributions, lack completeness or accuracy, be inconsistent, and fail to capture real-world noise level [33]. Although synthetic data might reproduce some high-level characteristics of operational systems, it typically lacks the temporal patterns, contextual dependencies, and edge cases needed for training robust AIOps models [34]. This creates a significant gap, where models that perform well in simulated settings often struggle to generalize to real-world systems, limiting their reliability and adoption potential [34, 35]. An additional limitation of synthetic data is the difficulty in evaluating its fidelity to real-world operational data [36]. Without access to comprehensive, high-fidelity real datasets for comparison, it becomes challenging to assess how well the simulated data captures the actual behaviors, anomalies, and interactions present in production systems. This lack of a clear reference standard makes it hard to quantify the relevance of synthetic datasets, raising concerns about their validity for training or benchmarking AIOps models.

Sharing real-world data. Alternatively, using real-world operational data is the most effective approach for developing reliable and broadly applicable AIOps solutions, since it reflects actual system behaviors, failures, and usage patterns. Nonetheless, sharing this data comes with significant challenges. One of the key issues is the sensitivity of IT telemetry [37, 38]. Software logs and other operational data often contain PII, such as IP addresses, access credentials, and user-related data, as well as confidential business information, such as internal architectures, deployment details, and system configurations [20]. Because of legal, ethical, and competitive concerns, companies are generally reluctant to share their operational data. Data protection laws such as *General Data Protection Regulation (GDPR)* [39], *Health Insurance Portability and Accountability Act (HIPAA)* [40], and *California Consumer Privacy Act (CCPA)* [41] enforce strict rules on how sensitive data should be stored, processed, and shared. Even when organizations anonymize their data, there is still a risk that individuals or systems could be re-identified [42–44]. Furthermore, effective anonymization is difficult, and the process can significantly reduce the value of the data. When anonymized, datasets frequently lose important structural elements, contextual information, or temporal accuracy, limiting the usefulness of the data for building and testing AIOps models. For instance, masking IP addresses or removing timestamps can destroy important patterns needed for tasks such as anomaly detection or root cause analysis [45]. As a result, most publicly available datasets are small in size, lack diversity, and do not accurately reflect the complexity of real IT environments. This shortage of high-quality, representative datasets remains a major barrier to both academic research and industry development in the AIOps field [4, 18].

1.3 Thesis Statement

This thesis aims to address the shortage of publicly available datasets for AIOps by contributing to both synthetic data generation and privacy-preserving data sharing. First, it provides an empirical analysis of how operational data is currently used in AIOps solutions, characterizing AIOps pipelines and identifying gaps. Second, it characterizes common patterns and structures in real-world telemetry data, paving the way for realistic synthetic data generation. Then it provides a definition of sensitive information in operational data, and finally, it proposes a deep-learning based technique for detecting those sensitive attributes, enabling safer data sharing.

1.4 Thesis Overview

1. *Studying data usage in AIOps pipelines.* To understand how data is used in AIOps solutions, we begin by analyzing AIOps practices in real-world projects. We characterize 119 AIOps projects and compare them with ML-based and General-based projects to identify the distinct traits of AIOps practices. In particular, we investigate the types and sources of input data, the methods used to process and analyze this data, and the associated tasks, providing a clearer picture of how AIOps pipelines operate in practice.

2. *Analyzing web application workloads for a better synthetic data generation.* In the next step, we investigate gathering AIOps data by simulating real-world scenarios using synthetic data. We perform a *Systematic Literature Review (SLR)* to identify how web application workloads, one of the primary sources for simulating synthetic data in AIOps, are used in research. By analyzing 78 articles, we gather a list of 12 real-world datasets and analyze the techniques and objectives of the studies using them. Furthermore, we characterize these workloads, analyzing distinct patterns in daily and weekly granularities. These patterns could be used in practice or future work for realistic workload generation.

3. *Understanding sensitive information in software logs.* We then investigate acquiring AIOps datasets from sharing real-world IT data. We analyze 25 public datasets to identify the common attributes in software logs, the most common data type of AIOps pipelines. Next, we investigate the definition of sensitivity from different perspectives of

privacy regulations (by analyzing five key privacy regulations), research literature (by performing an SLR and analyzing 58 articles), and industry practice (by performing a survey of 45 professional participants). We integrate our findings from these different perspectives and come up with a set of attributes that are generally sensitive and need to be anonymized before sharing.

4. *Detecting sensitive information in software logs.* With the knowledge about the categories of sensitive attributes in software logs, the next step is to locate these attributes across large volumes of data, ranging from gigabytes to terabytes. Historically, in practice, log anonymization techniques primarily rely on regular expressions (regex), which involve manually crafting rules to identify and replace sensitive information. However, these regex-based approaches suffer from significant limitations, such as extensive manual efforts and poor generalizability across diverse log formats and datasets. Hence, we introduce a deep learning-based framework designed to identify sensitive information in software logs, achieving much better results than regular expressions. With only 100 fine-tuning samples from the target dataset, our framework can correctly identify 99.5% of sensitive attributes and achieves an F1-score of 98.4%.

1.5 Thesis Contributions

This thesis makes the following original contributions.

- Conducted an empirical study on AIOps practices in real-world projects and analyzed the characteristics of 119 AIOps projects, including their input data, analysis techniques, and tasks. (Presented in Chapter 4)
- Performed an SLR on web application workloads to provide a thorough understanding of how these workloads are used in research and identified a total of 12 publicly available web application workload datasets that can be used for simulating synthetic data. (Presented in Chapter 5)
- Identified daily and weekly workload patterns across different web applications and presented their distinct characteristics. (Presented in Chapter 5)
- Analyzed 25 log datasets to understand commonly collected information in software logs. (Presented in Chapter 6)
- Analyzed 5 key privacy regulations to understand the privacy requirements in IT data.

- Performed an SLR of the existing practices and tools in the domain of log privacy. (Presented in Chapter 6)
- Conducted a survey of 45 industry professionals to uncover practical insights into the real-world challenges and strategies involved in log anonymization. (Presented in Chapter 6)
- Provided a systematic analysis of the effectiveness of regex-based approaches in detecting sensitive attributes in software logs. (Presented in Chapter 7)
- Annotated a dataset of 32,000 log entries labeled with sensitive information categories. (Presented in Chapter 7)
- Introduced a novel method for identifying sensitive attributes in software logs. (Presented in Chapter 7)

This thesis led to the publication/submission of the following research papers.

- **Roozbeh Aghili**, Heng Li, and Foutse Khomh. "Studying the characteristics of AIOps projects on GitHub." Empirical Software Engineering 28.6, 143, 2023, doi: 10.1007/s10664-023-10382-z.
- **Roozbeh Aghili**, Qiaolin Qin, Heng Li, and Foutse Khomh. "Understanding Web Application Workloads and Their Applications: Systematic Literature Review and Characterization." IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, pp. 474-486, 2024, doi: 10.1109/ICSME58944.2024.00050.
- **Roozbeh Aghili**, Heng Li, and Foutse Khomh. "Protecting Privacy in Software Logs: What Should Be Anonymized?." Proceedings of the ACM on Foundations of Software Engineering (FSE), 2025, "accepted".
- **Roozbeh Aghili**, Xingfang Wu, Foutse Khomh, and Heng Li. "SDLog: A Deep Learning Framework for Detecting Sensitive Information in Software Logs." ACM Transactions on Software Engineering and Methodology (TOSEM), 2025, "submitted".

The following publications are not directly related to the material of this thesis but were produced in parallel and address similar topics.

- Mohamed Amine Batoun, Mohammed Sayagh, **Roozbeh Aghili**, Ali Ouni, and Heng Li. "A literature review and existing challenges on software logging practices: From the creation to the analysis of software logs." Empirical Software Engineering 29.4, 103, 2024, doi: 10.1007/s10664-024-10452-w.

- Qiaolin Qin, **Roozbeh Aghili**, Heng Li, and Ettore Merlo. "Preprocessing is All You Need: Boosting the Performance of Log Parsers With a General Preprocessing Framework." IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, pp. 310-320, 2024, doi: 10.1109/SANER64311.2025.00036.

1.6 Thesis Organization

The reminder of this thesis is organized as follows. Chapter 2 provides the background information of this thesis. Chapter 3 outlines the literature review of related work. Chapter 4 presents an empirical study of AIOps projects to understand their different characteristics, including the types of input data and the methods used to process and analyze this data. Chapter 5 provides an SLR on web application workloads and characterizes 12 real-world datasets in order to help future work in realistic workload generation. Chapter 6 investigates the definition of sensitivity in software logs from different perspectives of privacy regulations, research literature, and industry practice. Chapter 7 introduces a deep learning-based approach to identify sensitive information in software logs. Chapter 8 provides a summary and conclusion of the thesis, addressing its limitations and proposing directions for future research.

CHAPTER 2 BACKGROUND

This Chapter provides an overview of the topics relevant to this thesis. It begins with a review of the field of information technology and its operational aspects, followed by an explanation of artificial intelligence. The discussion then moves to the concept of AIOps and the various types of data it involves.

2.1 IT Operations (IT Ops)

Information Technology (IT) is the use of computers, software, networks, and other digital technologies to process, store, retrieve, and transmit information. IT has become an essential part of many organizations, providing the tools and systems to support a wide range of business functions such as communication, data management, and decision-making. *IT Operations (IT Ops)* refers to the set of activities and processes involved in managing and maintaining an organization's IT systems. Gartner defines IT Ops as "the people and management processes associated with IT service management to deliver the right set of services at the right quality at competitive costs for customers" [46]. Some typical IT Ops roles include system administrators, network administrators, database administrators, security analysts, and help desk technicians. These professionals perform day-to-day IT Ops tasks, troubleshooting and resolving issues. Complementing IT Ops, *Information Technology Service Management (ITSM)* is defined as a set of practices and processes used to manage and deliver IT services to an organization's customers by focusing on functionalities and *Service Level Agreements (SLA)* [47]. An SLA is a contract or agreement between a service provider and its customers that defines the level of services that will be provided [48]. It typically includes a set of performance metrics, service standards, and penalties for failing to meet the agreed-upon service levels. ITSM practices rely on such agreements to ensure accountability and consistency, helping organizations to manage their IT services more effectively by providing a structured framework for planning, delivering, and supporting various services.

As computing becomes increasingly integral to modern life, IT systems are expanding in scale and complexity, requiring more resources for deployment and maintenance [49]. Over the past decade, advances in cloud computing and infrastructure virtualization have significantly reshaped the development, deployment, and management of enterprise applications [50]. Key technologies such as containers and virtualization have enabled the encapsulation of complex applications within isolated, portable environments. This abstraction not only simplifies the development process but also improves scalability, cost efficiency, and deployment flexibility.

In addition to containerization and virtualization, microservice architectures have emerged as a powerful design paradigm, allowing large applications to be broken down into smaller, modular components that can be developed, deployed, and maintained independently by distributed teams. Together, these advancements have driven a shift away from monolithic and static infrastructures toward more dynamic, scalable, and resilient cloud-native environments [50].

The main objective of IT operations and maintenance is to ensure high *Quality of Service (QoS)* by delivering continuous, uninterrupted system functionality [51]. Service outages caused by hardware failures, software bugs, or misconfigurations can lead to substantial financial losses [51]. For example, the estimated cost of one hour of server downtime for an organization ranges between $300,000 and $400,000 [52]. Another example is the seven-hour outage of Facebook in October 2021 which resulted in a nearly 5% decline in its stock value and an estimated loss of at least $60 million in advertising revenue [53]. These considerable losses emphasize the importance of identifying and mitigating the root causes of system failures [54]. To minimize disruptions and maintain service continuity, many organizations aim for extremely high system availability targets, often expressed in terms of *nines*. For example, four nines (i.e., 99.99%) availability allows for just over 52 minutes of downtime per year, while five nines (i.e., 99.999%) restricts downtime to approximately 5 minutes annually [55]. Achieving such high availability levels requires robust fault tolerance, automated recovery mechanisms, and proactive monitoring strategies.

Similar to software availability, software reliability is another critical dimension of QoS. It addresses the probability of software failures resulting from defects or errors in the code, which prevents the system from delivering the expected functionality [56]. To mitigate such failures and enhance the robustness of systems, the discipline of SRE has emerged at the intersection of software engineering and IT operations [57]. Originally introduced by Google in the early 2000s, SRE is designed to address the limitations of traditional operations teams in managing the complexity and velocity of modern software delivery [58]. It emphasizes the use of automation, observability, and software engineering principles to maintain reliable and scalable systems. SRE teams are not only responsible for responding to incidents and minimizing downtime but also play a proactive role in ensuring the health of systems [59]. In today’s cloud-native environments, where microservices and containerized architectures dominate, the role of SRE has become increasingly essential for managing operational complexity and supporting rapid, reliable software delivery [57, 59].

2.2 Artificial Intelligence (AI)

The term *Artificial Intelligence (AI)* was first used in 1955, when John McCarthy (Dartmouth College), Marvin L. Minsky (MIT), Nathaniel Rochester (IBM), and Claude Shannon (Bell Laboratories) applied for a grant to organize a summer research project at Dartmouth College [60]. In 1956, the Dartmouth Summer Research Project on Artificial Intelligence marked the official founding of AI as a research discipline, introducing its core mission: to formalize the processes of intelligence so precisely that they could be simulated by machines [60]. Early AI research focused on identifying formal methods that could replicate intelligent human behavior in areas such as medical diagnosis, mathematics, chess, and language processing, with the goal of automating these behaviors [61]. Probably one of the most notable milestones in AI history is the development of Deep Blue, a chess-playing supercomputer built by IBM that could evaluate up to 200 million positions per second [62]. In 1997, Deep Blue defeated Garry Kasparov, becoming the first computer system to beat a world champion in a match played under standard tournament time controls [62].

Generally speaking, AI refers to the techniques used to develop intelligent machines, particularly computer programs, designed for learning, reasoning, and perception [63]. Nilsson et al. [64] define AI as "that activity devoted to making machines intelligent... [where] intelligence is that quality that enables an entity to function correctly and with foresight in its environment". AI has been employed heavily in software systems in the last decade and has many applications in various industries, such as healthcare, finance, transportation, and entertainment. AI can be divided into different branches, including *ML*, *Deep Learning (DL)*, and *Natural Language Processing (NLP)*. We briefly introduce these branches, as they are used in AIOps techniques.

ML is a branch of AI that allows computers to learn from data and make decisions or predictions on their own [65]. Instead of being programmed with specific instructions, ML systems learn patterns and behaviors from data through a training process, allowing them to make decisions or predictions based on what they have learned [66]. By identifying patterns in data using mathematical and statistical techniques, ML systems can adapt and improve their performance over time. Given the central role of data in ML systems, they can be categorized into three main types [67]; 1) Supervised learning: The most widely used approach which relies on labeled data to train models by mapping inputs to known outputs; 2) Unsupervised learning: Works with unlabeled data to discover hidden patterns or structures; and 3) *Reinforcement Learning (RL)*: These are the applications where learning is driven by sequences of observations, actions, and rewards. In RL, systems learn through interaction with an environment, aiming to take actions that maximize cumulative rewards over time.

DL is a branch of ML and involves building and training *Artificial Neural Networks (ANN)*, which are algorithms inspired by the structure of the human brain. ANN consist of layers of interconnected nodes or neurons, processing information and make predictions based on that information. DL algorithms use multiple layers of these neural networks to learn and extract hierarchical representations of data, allowing them to identify complex patterns and relationships in the data [68]. DL has various use cases in different domains of NLP, CV, speech recognition, recommendation systems, and generative AI. DL techniques have received significant attention, especially after technological improvements and the invention of *Graphics Processing Units (GPU)*.

NLP is a branch of AI that deals with natural language, typically in the form of text. It involves the development of computational models and algorithms that enable machines to understand, interpret, generate, and manipulate natural language in a linguistically and semantically meaningful way [69]. The introduction of the Transformer architecture [70] has been particularly transformative, establishing a new paradigm for NLP research and enabling the development of models with outstanding language understanding and generation capabilities. Recent advances in NLP have been largely driven by the availability of large-scale textual datasets and the growing demand for systems capable of sophisticated and human-like language understanding. NLP systems employ a variety of ML and DL techniques, with significant progress marked by the introduction of large-scale pre-trained language models such as *Bidirectional Encoder Representations from Transformers (BERT)* [71]. In recent years, the field has seen a rapid acceleration in model development, with new language models being released every now and then by major organizations, such as Google, Microsoft, OpenAI, and Anthropic. These models have demonstrated state-of-the-art performance across a wide range of language tasks, including sentiment analysis [72], machine translation [73], and question answering [74]. NLP continues to play a critical role in numerous real-world applications, including healthcare, education, and customer service, highlighting its growing importance as a field of study.

2.3 Artificial Intelligence for IT Operations (AIOps)

Modern IT environments, driven by cloud computing, microservices, and continuous deployment, generate overwhelming volumes of telemetry data and alerts, making manual monitoring and incident resolution increasingly impractical [75]. In response, *Artificial Intelligence for IT Operations (AIOps)* has emerged as a transformative approach that leverages big data analytics and AI techniques to automate key operational tasks such as anomaly detection, failure prediction, and root cause analysis. Introduced by Gartner [8], AIOps integrates AI capabilities into the operational workflow, enabling software engineers, SREs, and IT Ops teams to manage scalable, reliable services more effectively [7].

AIOps enhances service quality and operational resilience by applying intelligent algorithms to real-time telemetry data [17]. Its primary goals include improving monitoring precision, forecasting performance degradations, increasing system availability, and supporting self-healing behaviors. For example, an AIOps-enabled system can detect early signs of resource strain, predict future performance based on historical patterns, and trigger self healing procedures without human intervention [7]. This accelerates incident response and reduces the reliance on specialized personnel for routine maintenance [4].

Beyond reactive responses, AIOps also enables proactive system optimization by identifying inefficiencies, forecasting workload trends, and recommending configuration adjustments [17]. By automating these processes, it frees IT teams from repetitive, manual diagnostics and allows them to focus on more strategic engineering efforts. The ultimate vision of AIOps is to build self-adaptive systems capable of maintaining high availability and performance with minimal human oversight [17].

Adopting AIOps represents a paradigm shift away from traditional operations models centered on manual debugging and ad hoc problem-solving-approaches that are neither scalable nor sustainable in large, distributed environments [7]. Although still an emerging field, AIOps is gaining widespread recognition for its potential to address long-standing challenges related to scalability, standardization, and human error [4]. Ultimately, AIOps enables robust, autonomous IT operations by automating the detection, diagnosis, and resolution of incidents. As a cross-disciplinary domain, it draws on insights from software engineering, AI, distributed computing, and data visualization to meet the demands of today's complex operational landscape [17].

2.4 Operational Data in AIOps

Similar to most AI-related concepts, data plays a critical role in all AIOps capabilities [49,76]. As inherently data-driven systems, AIOps platforms rely on the continuous ingestion and analysis of large-scale telemetry to generate actionable insights. The performance and accuracy of any AIOps solution are critically dependent on the quality, relevance, and heterogeneity of the operational data it processes. AIOps solutions depend on a wide range of telemetry sources that collectively offer a multidimensional view of system behavior and health. Some of these data types are introduced below and discussed in more detail in Chapter 4.

1. Software logs consist of timestamped messages generated by software components dur-

ing run time, capturing events, errors, warnings, and diagnostic details. Software logs often have a mix of structured metadata with unstructured free text, offering granular insight into internal operations.

2. Software traces come in various forms, including kernel-level, application-level, and distributed tracing. For example, distributed traces document the end-to-end execution path of requests through distributed services, providing valuable information about service dependencies, latency sources, and call sequences.

3. Performance metrics are numerical indicators, such as *Central Processing Unit (CPU)* load, memory usage, disk *Input / Output (I/O)*, and response times, typically collected at fixed intervals and used to assess resource usage and performance trends.

4. Network data contains connection patterns, protocol usage, and traffic flow, helping detect anomalies and monitor communication health.

5. *Key Performance Indicators (KPI)* represent abstract, high-level metrics that reflect service-level or business-level performance.

6. Sensor data, originating from hardware components such as servers, power supplies, or environmental controls, provides additional physical-layer observability.

7. Lastly, alerts and alarms from monitoring systems flag anomalies or critical events that need immediate operational response.

Despite their value, these diverse data sources present substantial analytical challenges. The most important one is *volume*: modern IT systems generate telemetry data at massive ranges, from terabytes to petabytes per month in large deployments [7]. Such scale makes manual inspection impractical and requires robust, automated data processing frameworks. Equally important is the high *velocity* at which this data is produced. Many operational environments generate millions of log entries, metrics, or events per minute [1]. To support real-time analysis and enable proactive responses, AIOps platforms must process these high-throughput streams with minimal latency, as delays can compromise the utility of the insights produced. The *variety* of operational data further complicates analysis. Telemetry data has a spectrum of formats and structures, from semi-structured logs and structured performance metrics to hierarchical trace data. In practice, such heterogeneity leads to fragmentation across data sources, hindering the ability to form a comprehensive view of system behavior. Effective AIOps solutions must therefore incorporate sophisticated integration mechanisms to overcome this heterogeneity. In addition, operational data often suffers from limited *veracity*.

Missing fields, inaccurate values, inconsistent timestamps, and noisy signals are common due to various instrumentation issues or temporary failures in monitoring systems. Such imperfections can significantly impact the reliability of models trained on this data, and addressing them is essential to ensure robust inference and diagnosis [52]. Finally, *contextual complexity* of AIOps operational data is of high importance. Individual data points seldom convey meaningful insights. Instead, the real value comes from correlating diverse signals, such as linking an anomaly to a log pattern and a recent code deployment, and to detect repetitive patterns over time. Overcoming these challenges is central to building trustworthy and intelligent AIOps solutions.

While these characteristics present significant processing challenges, acquiring high-quality operational data remains one of the most substantial challenges for AIOps. For any research or development in this field, access to relevant, diverse, and robust datasets is required for tasks ranging from training and validating ML models to enabling real-time analysis and accurate anomaly detection. However, public datasets in the field of AIOps are limited, and especially, public datasets that capture the complexity, diversity, and scale of enterprise IT environments are scarce [4, 7, 18, 19, 21–24]. Those that are available are often narrow in focus, targeting a single task, or no longer reflect current system characteristics. Broadly, AIOps data can be accessed in two ways: generating synthetic data by simulating real-world scenarios or acquiring real-world operational data from IT organizations. However, both methods come with their own distinct set of limitations and challenges.

2.5 Chapter Summary

This Chapter provides the key concepts related to AIOps and more specifically, to the data aspect of AIOps pipelines. In the next Chapter, a literature review of this thesis is presented and different research work in the area of AIOps are discussed.

CHAPTER 3 LITERATURE REVIEW

This Chapter reviews existing studies in the field of AIOps. First, an overview of existing literature on the AIOps domain is provided, including studies that define AIOps and explore its benefits, challenges, and possible future paths. Then, existing AIOps literature reviews are analyzed. These articles perform a literature review or a systematic mapping study on AIOps work or AIOps tasks, aiming to pinpoint current trends and identify gaps in existing studies. Next, articles performing AIOps tasks such as anomaly detection and failure prediction are examined. Following that, existing work that shares public AIOps datasets is introduced. These datasets have been created and shared for the use of researchers and practitioners. The Chapter then explores synthetic data generation and discusses data privacy concerns in IT Ops.

3.1 AIOps domain overviews

There are few papers discussing AIOps in general. They mainly focus on explaining the field of AIOps, pinpointing the advantages and challenges, and showing a roadmap from traditional systems to AIOps solutions.

The first time the terminology AIOps appeared was in the report published by Gartner [8] in 2018. Gartner introduces this terminology due to the excessive attention to AI techniques. According to Gartner, the increase in the volume, variety, and velocity of data in IT Ops needs strategies that leverage ML and big data analysis. Gartner discusses that AIOps can enhance a broad range of IT Ops processes, such as anomaly detection, performance analysis, event correlation, and automation. Gartner defines three main stages for AIOps; observe (using monitoring of real-time data and according to historical data), engage (the technical execution of the tasks such as anomaly detection and performance analysis), and act (try to automate the pipeline, not to need human intervention).

Dang et al. [7] discuss the need for AIOps solutions where it can address the DevOps challenges with AI. The paper mainly pinpoints three advantages of AIOps techniques; higher service intelligence (by self-adaption, auto-healing, and low human intervention), high customer satisfaction (by proactively searching and preventing the coming issues), and high engineering productivity (by removing tedious tasks). The article also lists three challenges of AIOps solutions; gaps in innovation methodologies (missing innovation methodologies that can guide people in different disciplines to build AIOps solutions), the needed engineering changes (building AIOps solutions needs significant engineering efforts and traditional engineering best practices do not fit the needs), and difficulty in building ML models (because of lack of high-quality data and complex dependencies among components [77]).

Bogatinovski et al. [18] discuss that AIOps is a candidate to produce the future standard for IT Ops management. However, it has three main challenges. First, various research branches from other research domains, like SRE, need to be combined. Second, novel modeling techniques are required in different systems and scenarios. Third, a good understanding and interpretability of these models are crucial in order to build trust between the AIOps approach and the domain experts. This article also discusses the lack of a good and public AIOps benchmark where all the data and methodologies can be compared. This benchmark can facilitate the comparison of different techniques under various settings.

Shen et al. [75] explore the evolution from traditional systems to AIOps. They divide the development of IT Ops into five eras; the age of manual, the age of scripts, the age of small systems, the age of platforms, and the age of AIOps. The article then proposes Proton, a novel AIOps system that adopts the layered design with interoperability services between modules. Their design consists of five layers; perception layer (monitoring attributes such as metrics, logs, and traces), storage layer (data processing frameworks or databases such as MySQL, ElasticSearch, and Spark), algorithm layer (primary AI models such as Random Forest or K-Means), business layer (tasks such as self-healing and failure prediction), and finally, interaction layer (with abilities such as face and gesture recognition). Proton has been deployed in a large IT system.

Gulenko et al. [51] discuss some sub-aspects of the AI-governance, focusing on IT Ops management. They define six levels of automation for a manageable and explainable AIOps platform. The first level of this spectrum is that the human operator implements the task and turns it over to the computer to execute. At the final level of the spectrum, the computer can develop new remediation and tuning activities as needed. The paper pinpoints that for some use cases, such as system updates and self-healing systems, no existing solution has yet reached level 6 automation.

Lyu et al. [52] discuss different ways for a more consistent interpretation of AIOps approaches. According to the study, many AIOps studies violate established practices in ML, such as interpreting models with sub-optimal performance. The article defines three aspects of consistency; internal, external, and time. It also provides guidelines for AIOps practitioners on how to interpret models consistently. For example, it suggests using sliding windows and Full History approaches to update the AIOps models.

Brahmandam [78] investigates the evolving role of intelligent IT Ops, focusing on the integration of AIOps, MLOps, and *Internet of Things (IoT)* to enhance organizational performance, efficiency, and cost-effectiveness. Performing statistical analyses, the paper shows that the adoption of these technologies can lead to improvements in response time, anomaly detection, issue resolution, and resource utilization. Notably, AIOps and MLOps are shown to enable faster, more proactive IT Ops, reducing downtime and operational costs. The study emphasizes that organizations embracing these intelligent systems are better equipped to manage complexity, improve agility, and drive digital transformation.

Dakkak et al. [5] investigates how AIOps supports *Continuous Deployment (CD)* in the context of software-intensive embedded systems, using a case study of a large telecommunications company working on 4G/5G radio access networks. It highlights that unlike traditional cloud-based systems, embedded systems present unique challenges due to their physical nature, configurability, and strict reliability requirements. One key finding is that AIOps platforms are essential for automating post-deployment activities, such as monitoring, root cause analysis, and service optimization, thus enabling faster, more intelligent service delivery despite increasing complexity. The study also identifies critical challenges for AIOps in this context, including data collection from complex ecosystems, correlation of heterogeneous data, and deployment flexibility to meet diverse customer needs.

3.2 AIOps Literature Reviews

Notaro et al. [79] conduct a systematic mapping study to identify research studies in AIOps. They provide a taxonomy of AIOps papers in order to investigate the trends and also a comparison of AIOps papers for specific problems. Their findings demonstrate a growing research interest in the field of AIOps, particularly for tasks such as anomaly detection and root cause analysis. According to them, the majority of studies (more than 60%) are related to failure management (e.g., failure prediction and failure detection). Resource management and scheduling are two other popular areas in AIOps.

To understand the trends in AIOps articles, similarly, Rijal et al. [23] perform a literature review about AIOps works. Their findings support the growing interest in AIOps [79] and count several benefits of AIOps: better monitoring of IT work, efficient time saving, improved human-AI collaborations, proactive IT work, and faster *MTTR*. The development of AIOps solutions also faces a number of difficulties, including low-quality data, doubt about the efficiency of AI and ML, identifying the proper use cases, and traditional engineering approaches. Their paper concludes the need for further research to improve human and AIOps interaction in order to enhance human productivity.

Cheng et al. [4] present a survey paper of AIOps techniques on cloud, focusing on how AI techniques can be applied to automate and optimize IT Ops across the software lifecycle. It categorizes core AIOps tasks, such as anomaly detection and root cause analysis, and analyzes the underlying data and challenges for each. One of the paper's key findings is that while detection and diagnosis have seen significant AI adoption, full automation remains underdeveloped. The authors highlight a growing shift from human-centric to AI-centric operations and emphasize the need for more advanced AI-driven automation solutions to meet the scalability demands of modern IT infrastructures.

Diaz-de-Arcaya et al. [19] present a systematic literature review on AIOps and *Machine Learning Operations (MLOps)*, focusing on their roles in overcoming the operational challenges of deploying data science and AI solutions in real-world environments. It identifies that while MLOps is more mature and widely adopted in traditional industries, AIOps is emerging as a crucial enabler in dynamic and complex IT environments. A notable observation is that AIOps is still underutilized during the deployment phases of the ML lifecycle, even though it offers significant benefits in automating and optimizing operations, minimizing human involvement, and managing the complications of cloud and edge environments. The study also reveals that AIOps is increasingly supported by major industry players (e.g., IBM, AWS, Microsoft) as a strategic tool for observability, system resilience, and performance optimization in large-scale IT ecosystems.

Notaro et al. [17] present a survey of AIOps methods for failure management, a core area in managing modern, large-scale IT systems. The authors review 100 failure management solutions and group them into five main categories and 14 subcategories based on their intervention timing and target problems, offering a structured taxonomy. The study reveals that while AIOps is expanding rapidly, the field lacks standardization and common benchmarks, making comparison and replication difficult. To address this, the authors propose future research directions including formal problem definitions and shared datasets to support reproducibility and advancement in AIOps-driven failure management.

Zhong et al. [80] present a survey of time series anomaly detection methods in the context of AIOps, which deals with monitoring large-scale, complex, and dynamic Internet-based services. The paper discusses the unique challenges of anomaly detection in AIOps, including noisy high-dimensional data, rare anomalies, the lack of labeled data, and frequent pattern shifts due to software/hardware changes. One of the key contributions is a taxonomy that organizes existing detection methods according to their core characteristics, helping practitioners and researchers select suitable techniques for real-world use cases. The paper also reviews public datasets for anomaly detection, evaluation metrics, and provides software resources for reproducibility. It also identifies gaps in the literature, such as the lack of methods tailored to AIOps-specific anomaly types, and highlights promising future directions such as contextual, collective, distributed, and privacy-preserving anomaly detection.

Remil et al. [3] perform a literature review study focused on AIOps solutions in incident management. The paper highlights that modern IT systems generate vast, complex data streams, rendering traditional manual and rule-based management methods inefficient. The study proposes an AIOps terminology and taxonomy, aiming to unify research and industrial contributions that currently lack consistent frameworks for data management, problem definition, and implementation. The paper outlines six fundamental AIOps capabilities: Perception (gathering heterogeneous data sources), Prevention (actively identifying potential failures), Detection (discovering abnormal patterns in data), Location (identifying and analyze potential root causes), Action (conducting reactive triage on problems and prioritizing incidents), and Interaction (human-AIOps system interaction for effective issue solving). It also categorizes existing AIOps research based on these capabilities, application areas, data sources, and technical approaches. The paper also highlights limitations in current AIOps capabilities, noting that incident classification and mitigation receive less attention than detection. Finally, it guides future developments toward more robust, interpretable, and scalable AIOps solutions.

3.3 AIOps tasks

In recent years and with the emergence of AIOps as a research field, more and more studies have been conducted in this domain. Prior works have come up with different solutions for various problems. Anomaly detection [9,10], failure prediction [11,12], ticket management [81, 82], self-healing [15, 16, 83], and issue diagnosis [84] are among the most studied topics in AIOps solutions. We divide AIOps papers by their main objective into four sections and discuss them in more detail.

Anomaly Detection. Anomaly detection is one of the most common tasks in AIOps [18]. Previous studies perform anomaly detection using different approaches; for example, Fu et al. [85] and Sharma et al. [86] use clustering to find the anomalies, Liu et al. [87] and Xu et al. [88] use tree-based models, and Brown et al. [9] and Du et al. [89] use neural networks to perform the task. To perform anomaly detection, different data sources have been used. Even though software logs [9,85,89,90] are the most common data source, other sources such as performance metrics [86,91], network traffic [92,93], and software traces [10,94] have also been employed.

Failure Prediction. Predicting failures is another popular task in AIOps field [17]. Failure prediction can be divided into two groups of hardware failures and system failures. Compared to anomaly detection task where various data sources have been employed, in failure prediction, usually only the performance metrics of the systems are used. Performance metrics are measurements that help in identifying and analyzing system bottlenecks and diagnosing issues. Some of the most popular performance metrics include CPU usage, memory usage, response time, throughput, I/O, and network latency. Zhao et al. [12] use different system metrics, such as CPU usage and the number of live threads of the system, to predict performance anomalies at run-time. Lin et al. [77] leverage temporal data (e.g., CPU and memory usage metrics, alerts) and spatial data (e.g., rack locations) and construct MING, a deep learning-based approach. MING is able to rank the faulty nodes of the cloud system and is applied to the maintenance of one of the cloud service systems in Microsoft. Li et al. [11] try to enhance the performance of MING, e.g., by enriching the data representing node failures through an oversampling approach. They also discuss the criteria for successful adoption of AIOps solutions, including trust, interpretation, maintenance, scalability, and in-context evaluation. Nguyen et al. [95] introduce a deep learning-based AIOps solution for proactive cybersecurity monitoring in live production environments. They also implement a multivariate forecasting model that operates in an unsupervised manner to model multiple network monitoring channels simultaneously. Unlike traditional threshold-based systems, the proposed model enables baselines, allowing real-time detection of anomalies in evolving network conditions. The work also emphasizes the importance of automation in software deployment through reducing complexity in long-term system management. They propose future directions such as using federated learning to train models in order to reduce the sensitive data sharing.

Root Cause Analysis. Root cause analysis is the technique of identifying the underlying causes of problems or issues in the system. The process involves identifying and analyzing the factors that contributed to the problem, determining the root cause, and developing solutions to address the underlying issue. Ding et al. [13] leverage the generated alarms of a software system and find the root causes of the issues in an online manner. Wang et al. [14] implement a graph-based algorithm and construct GROOT to perform root cause analysis. Their approach uses a combination of different event data (i.e., performance metrics, logs, and developer activities). GROOT is deployed in the production services of eBay. Zhang et al. [96] propose a root cause analysis framework that also leverages different data sources: KPIs, software logs, and topology data. Their primary model is a hierarchical Bayesian network that helps handle novel types of root causes.

Incident Management. Incident management refers to the process of identifying, analyzing, and resolving incidents that happen within a software system. Although the majority of organizations have established unique methods for handling incidents, critical service incidents still happen unexpectedly, and the incident system fails to mitigate them. Chen et al. [97] provide an overview of incident management at Microsoft by analyzing their incident management practices over two years and identifing the distribution of incident severities for each section (e.g., network, database). Saha et Hoi. [98] leverage past incident root cause analysis reports to manage the incidents. They present an incident causation analysis engine that extracts information from previous root cause analysis reports and use that knowledge for new incidents. They employ pre-trained NLP models over reports collected over a few years at Salesforce. Li et al. [99] propose an AIOps framework for incident detection. Their framework consists of four main parts: multi-aspect detection (where it is able to automatically identify the combinations of different data types such as software logs and performance metrics), proactive detection (where it can proactively search for future hardware and software failures), incident refinement (where it can provide a global view of the incident and prioritize high-impact incidents), and incident enrichment (where it can locate the faulty scope).

3.4 Public AIOps Datasets

As research in the domain of AIOps grows, researchers try to enhance the performance of AIOps pipelines in various tasks, such as anomaly detection, failure prediction, and self-healing. These studies usually use private datasets for their experiments. However, the lack of large-scale public datasets hinders fair comparison between different AIOps approaches and limits the community's ability to track progress and guide future research [18]. Besides, the efforts done on private datasets may not be generalizable in real-world scenarios, where these approaches could perform unstably because of different scales and complexity. Therefore, the lack of public large-scale real-world datasets is one significant limitation of AIOps research [4, 7, 18, 19, 21–24].

Although publicly available AIOps data is scarce, several initiatives have attempted to address this gap. Yahoo benchmark [25] is one of the first attempts for an open-source dataset for anomaly detection. The paper introduces EGADS, a generic anomaly detection system that employs different time-series modeling and anomaly detection algorithms. It is implemented at Yahoo for monitoring and alerting purposes, and the main model as well as the generated datasets are open-sourced. Nevertheless, the scale of Yahoo dataset is limited for AIOps applications as it includes only hundreds of data points.

The Numenta benchmark [100] is another dataset for anomaly detection. The generated dataset consists of real-world time-series data and is labeled by hand, marking anomalous points. The dataset contains different metrics, such as CPU utilization and sensor temperature. The dataset contains 58 data files, each with 1,000-22,000 instances. The authors also highlight the unique challenges of anomaly detection in real-time environments, where models must process data sequentially, adapt continuously, and prioritize early detection without human intervention.

One notable effort is LogHub [101], which is a collection of 16 datasets spanning six categories: distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software. LogHub datasets contain a total of around 77 GB raw log data. Among the datasets, six are labeled, enabling supervised learning approaches, while the rest support unsupervised or semi-supervised methods. LogHub is widely adopted in the software engineering community, serves as a key dataset for research in several domains such as log parsing, anomaly detection, and system monitoring [102, 103]. However, despite its widespread usage, parts of the collection are considerably outdated; for instance, the *Apache* and *BGL* datasets date back to 2005. Besides, most of these 16 datasets are generated in a lab settings, not having a real-world production setup.

Similar to LogHub, ITA [26] is another repository that hosts public server logs and packet traces. It includes large, real-world datasets such as *WorldCup98*, which records the 1.3 billion requests made to the servers of FIFA during the 1998 World Cup, or *Calgary*, which records the requests sent to the department of computer science of the University of Calgary in Canada over one year. However, all ITA datasets are relatively dated, having been collected between 1989 and 2007.

Certain cloud providers have also released datasets for public use. One example is the Google cluster workload dataset [104, 105], which contains traces collected from workloads managed by Google's cluster management system, Borg. This dataset has been released in three versions: the initial release in 2009, followed by updates in 2011 and 2019. The traces capture detailed information on job submissions, scheduling decisions, and resource usage associated with jobs executed within Borg clusters.

Microsoft Azure has also released a dataset containing VM traces and Azure Functions traces [106,107]. The VM traces are available in two versions, collected in 2017 and 2019. The 2017 dataset includes data from 2 million virtual machines and 1.2 billion resource utilization records, while the 2019 version has a record of 2.6 million VMs and 1.9 billion readings. Additionally, the Azure Functions traces, gathered in 2019 and 2021, provide insights into applications running on Azure Functions, capturing details such as application and function IDs, execution times, and memory usage. Since 2017, Alibaba has also released a series of cluster trace datasets derived from its production systems, including microservice-based architectures and GPU clusters [108]. This repository is actively maintained, with the most recent update published in April 2025. The cloud traces from Google, Azure, and Alibaba are widely used across various research areas, including job scheduling, workload allocation, and failure prediction [109–111].

Several public datasets are also available in the networking domain. The KDD Cup 1999 dataset [112] was created for a data mining competition, hosted in conjunction with the KDD-99 conference. It serves as a benchmark dataset for evaluating the performance of ML algorithms in the context of network intrusion detection. The dataset simulates a military network environment and has a training size of about four GBs. It consists of compressed binary tcpdump files from seven weeks of network traffic and includes a mixture of normal traffic and 22 different types of attack scenarios. Each network connection in the dataset is represented by 41 features describing various attributes of the connection, such as duration, protocol type, number of failed login attempts, and traffic characteristics. The goal of the competition was to develop models capable of accurately distinguishing between legitimate connections and malicious intrusions. The KDDCUP99 dataset has several limitations, including unrealistic *Time To Live (TTL)* values in attack packets, a mismatch in the probability distributions between training and testing sets, and a lack of representation for modern low-footprint attack scenarios [113–115]. These issues undermine its reliability for evaluating modern intrusion detection systems.

The UNSW-NB15 dataset [116] was developed to address the limitations of KDDCUP99 and similar datasets. It simulates both normal network behavior and low-footprint attacks. By leveraging tools such as tcpdump, the dataset captures network traffic characteristics and provides 49 detailed features. The abnormal traffic consists of nine diverse attack families sourced from vulnerabilities listed in the *Common Vulnerabilities and Exposures (CVE)* database [117]. With 100 GB of data captured over two simulation periods in early 2015, the dataset is labeled using ground truth tables, making it a benchmark for evaluating modern network intrusion detection systems.

The CIC-IDS2017 dataset [118] is another benchmarking dataset in this domain that was developed to provide a more recent and comprehensive resource for evaluating network intrusion detection systems. It includes both benign network traffic and a range of common attacks executed over a five-day period, with normal activity recorded on the first day and attack scenarios distributed throughout the following days. The dataset contains labeled network flows with annotations based on timestamps, IP addresses, ports, protocols, and attack types.

3.5 Generating Synthetic Data

To acquire more AIOps data, two general approaches exist: (1) Generating synthetic data by simulating real-world scenarios, or (2) Acquiring real-world operational data from IT organizations. Synthetic data refers to artificially generated data that are not obtained from real-world events or measurements; instead, they are produced using different computer algorithms or simulations, with the goal of replicating the statistical properties and structural characteristics of authentic data [119, 120]. This approach provides a viable alternative in scenarios where access to real data is restricted, either due to availability, cost constraints, or regulatory and privacy considerations [121].

The motivation behind using synthetic data is driven by several key factors. First, acquiring and labeling real-world datasets is often expensive and labor-intensive, posing challenges for applications that require large volumes of high-quality annotated data [121]. Synthetic data mitigates this limitation by enabling the generation of extensive, diverse, and somehow-realistic datasets at low cost, supporting the training and evaluation of ML models in data-scarce environments [120]. Second, data quality presents another challenge in practice. Real-world datasets frequently suffer from noise, incompleteness, and inconsistencies that can reduce model performance. In contrast, synthetic data can be engineered to meet specific quality standards, ensuring completeness, consistency, and relevance, thereby enhancing model reliability [36]. Third and perhaps most importantly, privacy is the major motivation for synthetic data generation. In domains such as healthcare and finance, data sharing is often hindered by regulatory and ethical constraints, as well as risks of re-identification [119, 121, 122]. Properly generated synthetic datasets contain no PII, yet preserve the analytical value of the original data [123]. This enables privacy-preserving data sharing and collaboration, allowing researchers and practitioners to perform analysis on data that is similar to real-world complexity without compromising confidentiality.

The growing need for high-quality synthetic data has led to the development of a wide range of generative methods, spanning from statistical methods to advanced DL frameworks. These methods vary in how they model the underlying data distribution and generate new samples, and can be broadly classified into statistical approaches, traditional ML algorithms, and deep generative models.

Statistical models represents some of the earliest and most interpretable methods for synthetic data generation. They often rely on probabilistic assumptions to approximate the distribution of real data. Notable examples include *Gaussian Mixture Model (GMM)*, which represents data as a weighted sum of multiple gaussian components, and *Kernel Density Estimator*

(KDE), which estimates the probability density function of observed data points to facilitate the generation of new samples [124]. While these methods perform well for low-dimensional or structured data, they typically struggle to scale to high-dimensional, sparse, or highly non-linear datasets. Another related family includes Markov Chain Models, which are well-suited to sequential data such as software logs or time series, where the likelihood of each event depends on preceding elements [125].

In addition to statistical techniques, several optimization and heuristic methods have been used for synthetic data generation. A prominent example is the use of *Genetic Algorithm (GA)*, which simulates evolutionary processes to iteratively refine a population of synthetic samples. Through repeated operations of selection, crossover, and mutation, guided by a fitness function, GA can effectively explores complex data spaces and generate samples that approximate the characteristics of the original dataset [126]. Another approach involves rule-based systems, also referred to as knowledge-driven techniques, which generate synthetic data by enforcing domain-specific rules and logical constraints derived from publicly available sources and datasets. These methods rely on manual curation of generative models, typically expressed as rule sets and statistical or mathematical formulations to encode domain knowledge. These systems offer transparency and control but may fall short in modeling subtle dependencies or unanticipated variability found in real-world data [119].

The most notable progress in synthetic data generation, however, has emerged from deep generative models. Among them, *Autoencoder (AE)* and their probabilistic extension, *Variational Autoencoder (VAE)*, learn compact latent representations of data and enable the generation of new samples by decoding points from this latent space [127]. VAEs are especially effective in producing smooth and coherent synthetic data by ensuring that the latent space is continuous and structured. Arguably, *Generative Adversarial Network (GAN)* has had the most transformative impact in synthetic data generation. GANs consist of a generator network that synthesizes data and a discriminator network that attempts to distinguish synthetic samples from real ones [128]. Through adversarial training, the generator iteratively improves its output to produce increasingly realistic data. Diffusion models are another class of deep generative techniques known for producing highly realistic synthetic data. They work by modeling the reverse of a noise-adding process, learning to reconstruct clean data from progressively noisier versions during training. At inference, the model begins with pure noise and iteratively denoises it to generate realistic samples that reflect the underlying data distribution [129]. Diffusion models have proven to be a powerful tool for synthetic data generation.

Despite the growing interest in synthetic data, its practical adoption comes with significant challenges. One primary concern is the assessment of how closely synthetic datasets mimic real-world data distributions and relationships. Evaluating synthetic data typically involves measuring its fidelity to real data through techniques such as distance metrics [130]. However, these evaluation methods can differ widely and may result in inconsistent or even conflicting conclusions about the quality and reliability of the generated synthetic data [130]. Furthermore, the simplicity of the assumptions made during the generation process can result in synthetic data that does not accurately capture the complex relationships and interactions present in real datasets [34]. For example, while synthetic datasets can be valuable for augmenting limited real data in machine learning applications, they may fail to incorporate the high-dimensional and categorical characteristics of realistic medical data [131]. This can undermine the credibility of analyses based on synthetic datasets, as evidence in research suggest that analyses conducted using synthetic data frequently yield different results than those based on actual data [34]. Additionally, the popularity of generative models like GANs illustrates the sophistication of current synthetic data generation approaches [36]. Unlike simply using existing datasets, generating synthetic data involves complicated modeling, iterative training, and fine-tuning to accurately replicate complex data distributions. This added complexity emphasizes the technical challenges and expertise required to produce high-quality synthetic data. In the end, while advancements have been made in methodologies that aim to improve the realism of synthetic datasets, thorough and rigorous evaluation remains crucial to ensure that synthetic data can genuinely serve as a reliable substitute or complement to real data in various analytical contexts.

Prior research often lacks comprehensive analyses of workload characteristics and behaviors. Moreover, the limited availability of public datasets for analyzing real-world scenarios and simulating workloads poses a challenge in generating synthetic data that truly reflects real-world operational patterns. To address these gaps, Chapter 5 performs an SLR on web application workloads (one of the main data sources to analyze real-world scenarios) to provide a thorough understanding of how these workloads are used in research. Besides, 12 publicly available web application workload datasets are identified that can be used for simulating synthetic data. Furthermore, by identifying daily and weekly workload patterns across different web applications and presenting their distinct characteristics, this work paves the way for realistic workload generation techniques, helping to bridge the gap between synthetic data and the nuanced behaviors of real-world operational systems.

3.6 Data Privacy in IT Ops

Unlike synthetic datasets, which may unintentionally provide an oversimplified view of complex systems, real-world data captures the complex nuances and dynamics present in everyday scenarios, making it more suitable for rigorous analytical efforts. Real-world data enhances the precision and relevance of outcomes in operational processes and increase trust among stakeholders due to its direct reflection of true operational contexts. However, use of real-world data poses significant challenges primarily due to privacy and security concerns. Preserving privacy in IT Ops data is not just a technical challenge but a fundamental ethical and legal responsibility; it mitigates security risks, protects business confidentially, maintains user trust, ensures compliance with internal and external policies, and facilitates responsible data governance. A significant concern in IT Ops data management and sharing is the exposure of PII. PII refers to any data that can be used to identify an individual, either directly, such as names and email addresses, or indirectly, such as device identifiers. While PII is often the focus of privacy concerns, it is not the only source of re-identification of individuals. A well-known study by Sweeney et al. [44] demonstrated that using just three attributes of postal code, date of birth, and sex, over 86% of U.S. citizens could be uniquely identified. These attributes, known as *quasi-identifiers*, are pieces of information that, while not directly revealing someone's identity on their own, can be combined with other data to potentially identify an individual.

Privacy regulations are designed to protect personal data and ensure that they are handled responsibly and ethically. Many countries have introduced privacy regulations to guide how organizations collect, store, and use sensitive information. These regulations vary by region, and some countries enforce multiple laws to cover different types of data. In the United States, for example, the *HIPAA* [40] addresses healthcare data, while the *CCPA* [41] focuses on consumer-related information. Similarly, China enforces two major laws: *Personal Information Protection Law (PIPL)* [132] and the *Data Security Law (DSL)* [133]. In Europe, the *GDPR* [39] serves as the primary legal standard for data privacy. Likewise, Canada follows the *Personal Information Protection and Electronic Documents Act (PIPEDA)* [134], and Brazil has adopted the *General Data Protection Law (LGPD)* [135].

Several privacy regulations provide formal definitions of personal data and outline specific attributes that need protection. The GDPR, for example, defines personal data as any information that relates to an identified or identifiable individual. This broad definition covers both direct identifiers, such as names and ID numbers, and indirect identifiers, including IP addresses and location data. GDPR considers IP addresses as personal data when they can be linked to an individual, either directly or by combining them with other information accessible to the data controller. Unlike GDPR, which broadly defines personal data, HIPAA specifies 18 sensitive identifiers that must be protected. These include both direct identifiers (e.g., names, contact numbers, social security numbers) and indirect ones (e.g., geographic locations, date information, and IP addresses). Many of these identifiers are often present in IT Ops data, such as timestamps, IP addresses, device or system identifiers, and *Uniform Resource Locators (URLs).* While data protection laws such as GDPR and HIPAA provide guidance on what constitutes personal data, none are tailored specifically to the context of IT Ops. Hence, it is necessary to interpret and adapt relevant regulatory elements to IT Ops data, taking into account the applicable legal frameworks of each region.

To protect sensitive information within IT environments, anonymizing operations data is crucial for preventing unauthorized access to user, system, or infrastructure details. It also enables secure data sharing and analysis while ensuring compliance with privacy regulations and organizational policies. There exist different anonymization algorithms that reduce or eliminate sensitivity in IT Ops data, resulting in an increased level of data privacy.

Black Marker refers to the technique of masking an attribute by overwriting them with placeholder values such as $IP_ADDRESS or * [136]. While simple and effective at protecting sensitive data, this technique comes at the cost of usability, as it permanently removes potentially valuable information and reduces overall data utility. *Truncation* shortens the values by removing bits or characters beyond a chosen point. It involves selecting a cutoff point in a data field and eliminating all content beyond it [137]. For instance, an IP address like 192.168.1.1 could be truncated to 192.168.x.x. *Permutation* refers to a one-to-one mapping of original values to new ones, making it suitable for scenarios where preserving the order or count of data is important, but the actual values must be hidden [138]. Various forms of permutation functions exist, each offering different trade-offs in terms of performance, collision resistance, and security. A core requirement across these techniques is that reversing the permutation should be computationally difficult. For example, when anonymizing IP addresses using a hash function as part of the permutation, the approach can be insecure if the hash function is publicly known. This is particularly problematic for IPv4 addresses due to their limited value space, making brute-force attacks feasible unless additional safeguards are applied. *Prefix-preserving* anonymization is a specialized form of permutation that maintains the original structure of values by applying direct substitution with a key constraint: shared prefixes in the original data must be reflected in the anonymized output [139]. For example, if two private IP addresses share a common prefix of n bits, their anonymized version will also share the same n-bit prefix. This ensures that hierarchical relationships, such as subnet structures in IP addresses, are retained throughout the anonymization process. *Hashing* is the process of transformsing input data into a fixed-size output, typically using a mathematical algorithm known as a hash function [38]. In the context of anonymization, hashing functions can obscure sensitive information by mapping each input to a corresponding hash value. Unlike permutations, hash functions do not guarantee unique outputs, which may result in collisions, especially when the output must be truncated to match the original data's length. This truncation weakens the hash and increases the likelihood of collisions, reducing its effectiveness for preserving uniqueness in anonymized datasets.

Anonymizing IT Ops data presents three major challenges: (1) Understanding the sensitive information that needs to be anonymized, (2) Detecting sensitive attributes within vast amounts of information, often gigabytes or terabytes in size, and (3) Effectively anonymizing sensitive attributes while handling the utility-privacy trade-off.

As mentioned earlier, although multiple privacy regulations such as GDPR or HIPAA govern data protection in general, none of them are specifically tailored to IT Ops data. Furthermore, there is no standardized framework or set of guidelines that organizations can rely on to identify and address the sensitive information of their IT systems. As a result, organizations are left to independently assess their own IT Ops data and determine what constitutes sensitive information that should be anonymized. This often requires a context-specific evaluation based on the nature of the data, the operational environment, and the potential risks associated with data exposure.

After identifying sensitive attributes, a new challenge arises: the large volume and complexity of IT Ops data makes it difficult to accurately isolate the information that requires protection. As data grows, sensitive information often becomes entangled with other operational data, complicating the task of pinpointing which attributes need anonymizing. While historically regular expressions are the common practice to identify attributes in IT Ops data [140], they come with many limitations. They lack generalizability across different datasets and data types, they need manual efforts to design the patterns, and they are not well-suited for detecting certain attributes that do not have well-known structures, such as user names. Furthermore, the varied nature of data often leads to additional difficulties, as sensitive attributes may not always be obvious or may consist of multiple interrelated fields, complicating both detection and anonymization processes.

Once sensitive attributes are identified, the challenge of effectively anonymizing them while preserving the overall utility of the data emerges. The utility-privacy trade-off indicates that enhancing privacy measures can lead to a loss of meaningful insights from the data, as anonymization techniques often employ generalization or suppression strategies that compromise the richness of the dataset [141]. For example, removing all identifiable information can result in data that, while non-identifiable, lacks the specificity needed for detailed analyses, thus reducing its effectiveness for decision-making [142]. Achieving a balance between protecting individual privacy and maintaining data utility requires sophisticated anonymization techniques that can adaptively anonymize based on the context of use, ensuring that the anonymized dataset still serves its intended analytical purpose without exposing sensitive information. As a result, organizations need to carefully manage this challenging environment by implementing strong anonymization practices that comply with regulatory standards while addressing operational demands, ultimately improving consumer confidence and protecting security in today's data-centric world.

While general data privacy has been extensively studied, the specific domain of privacy in software logs (the most common type of data in AIOps pipelines) remains underexplored, with inconsistent definitions of sensitivity and a lack of standardized guidelines for anonymization. To mitigate this gap, Chapter 6 offers an analysis of privacy in software logs from multiple perspectives. By combining analysis from three aspects of privacy regulations, research literature, and industry practices, a set of attributes that are generally sensitive are listed. These attributes need to be anonymized before sharing the datasets.

With the knowledge about the categories of sensitive attributes in software logs, the next step is to locate these attributes across large volumes of data, ranging from gigabytes to terabytes. Chapter 7 introduces a deep learning-based framework designed to identify sensitive information in software logs, achieving much better results than regular expressions (the to-go approach to find attributes both in research and industry practices). With only 100 fine-tuning samples from the target dataset, our framework can correctly identify 99.5% of sensitive attributes and achieves an F1-score of 98.4%.

3.7 Chapter Summary

Over the last decade, the introduction of AIOps has started a shift in IT operation management across many organizations. This Chapter presents different research work in this domain, including AIOps major tasks and available public AIOps datasets. The next Chapter will present our study to understand different AIOps data sources and their application in real-world AIOps projects.

CHAPTER 4 ARTICLE 1: STUDYING THE CHARACTERISTICS OF AIOPS PROJECTS ON GITHUB Roozbeh Aghili, Heng Li, Foutse Khomh, Accepted in Empirical Software Engineering, Publication Date: 2023/10/18

4.1 Chapter Overview

AIOps leverages big data technologies and AI techniques to automate and improve various areas of IT Ops. Prior works have proposed various AIOps solutions to support different tasks such as anomaly detection and failure prediction. In this Chapter, we conduct an in-depth analysis of open-source AIOps projects to understand the characteristics of AIOps in practice. We first carefully identify a set of AIOps projects from GitHub and analyze their repository characteristics (e.g., the used programming languages). Then, we qualitatively examine the projects to understand their input data, analysis techniques, and goals. To provide context, we also sample two sets of baseline projects from GitHub: a random sample of machine learning projects and a random sample of general-purposed projects. By comparing different characteristics between our identified AIOps projects and these baselines, we derive meaningful insights. Our results reveal a recent and growing interest in AIOps solutions. We also find that the most common data type of AIOps projects is monitoring data (including software logs, performance metrics, and network data), the most common analysis techniques of AIOps pipelines are ML approaches, and the most common goal of AIOps projects is anomaly detection. Our findings offer valuable guidance to researchers and practitioners, enabling them to comprehend the current state of AIOps practices and shed light on different ways of improving AIOps' weaker aspects. To the best of our knowledge, this work marks the first attempt to characterize open-source AIOps projects.

4.2 Study Design

In this section, we first define the objectives of this study and then detail the experiment setup.

4.2.1 Study Objectives

Studying AIOps practices in real-world projects is important and has several benefits, including (1) helping researchers and practitioners understand the current status of AIOps solutions and the characteristics of AIOps projects; (2) providing guidance for researchers and practitioners to adopt best-performing AIOps solutions for their application scenarios; and (3) identifying problems in AIOps practices and shedding lights on future research opportunities. Therefore, this study identifies and analyzes a set of AIOps projects publicly available on GitHub. We also compare the selected AIOps projects with two baselines: traditional ML projects and General-purpose projects. Our goal is to understand the characteristics of these AIOps projects in the context of the baseline projects. Specifically, we first investigate the overall characteristics of these AIOps projects in terms of their GitHub metrics, then we dig deeper into the individual projects to understand the data, techniques, and goals of these projects. Our *Research Questions (RQs)* are as follows.

RQ1 ***What are the characteristics of AIOps projects in terms of their GitHub metrics?*** We analyze the GitHub metrics of AIOps and baseline projects to understand the current status of AIOps projects and also compare them with baselines in terms of their GitHub metrics, such as the programming languages and the number of stars. We observe that AIOps solutions are being developed with a faster growth rate compared to the baselines. AIOps projects also have a higher distribution of popularity metrics (e.g., number of stars and forks), and also more pull requests and issues compared to baselines.

RQ2 ***What are the characteristics of AIOps projects in terms of their input data, analysis techniques, and goals?*** In order to understand the characteristics of AIOps projects (i.e., their input data, analysis techniques, and goals), we manually investigate each project. We find that monitoring data (e.g., logs and performance metrics) is the most used input data, classical ML techniques are the most adopted analysis techniques, and anomaly detection is the primary goal of many AIOps projects.

We share the replication package of this study so that future work can replicate or extend it [143].

4.2.2 Experiment Setup

We first describe our approach for collecting the AIOps and baseline projects. Then, we present the overview of our study, and finally, we describe the steps for collecting and verifying AIOps and baseline projects, respectively.

4.2.2.1 Overview of our study

Figure 4.1 presents an overview of our approach to study the characteristics of AIOps projects. We use GitHub as the main source to extract the needed data for our analysis. At the time of the data collection (January 2023), GitHub has over 100 million registered developers and over 372 million repositories [144]. GitHub is also considered the largest hosting service for open-source software systems [145]. Hence, projects found on GitHub are likely to reflect the diversity of existing AIOps projects. Many existing studies also extract the needed information for their analysis from GitHub [146–149]. We start by searching the projects with the keyword "AIOps". Then, through manual verification (e.g., removing non-AIOps projects), keyword expansion (i.e., through pattern mining), second-round search and manual verification, and threshold-based filtering (e.g., by the number of stars), we collect a total of 119 AIOps projects that are used to answer our RQs.

In order to better understand the characteristics of AIOps projects in a bigger context, we also compare our identified AIOps projects with two baselines:

1. Randomly sampled machine learning (ML) projects;
2. Randomly sampled general projects

We choose the ML baseline because most of the AIOps projects leverage ML techniques. We choose the General baseline to compare our AIOps projects with general software applications on GitHub. Finally, we perform qualitative and quantitative analyses on the collected data to answer our RQs. Below, we describe the details of our data collection. The detailed approaches for answering our RQs are presented in Section 4.3.

4.2.2.2 Collecting AIOps Projects

Through two rounds of searching AIOps-related keywords on GitHub, we collect a total of 1016 candidate projects. Through filtering and manual verification, we end up with 119 of them as our final set of AIOps projects. The methodology we use to select these projects follows the systematic approach recommended by Basili et al. [150] and is described in the following sections.

Search AIOps projects (first round). In the initial step of identifying suitable projects, we use the GitHub interface to search for repositories specifically labeled as AIOps projects. To do this, we employ the keyword "AIOps" and search across four key sections of each repository: the repository name, the "about" section, the "topics" section, and the contents of the "readme" file. After searching for the keyword "AIOps", we find a total of 542 repositories that match our criteria. These 542 repositories represent all the available projects on GitHub that have been labeled as AIOps projects.

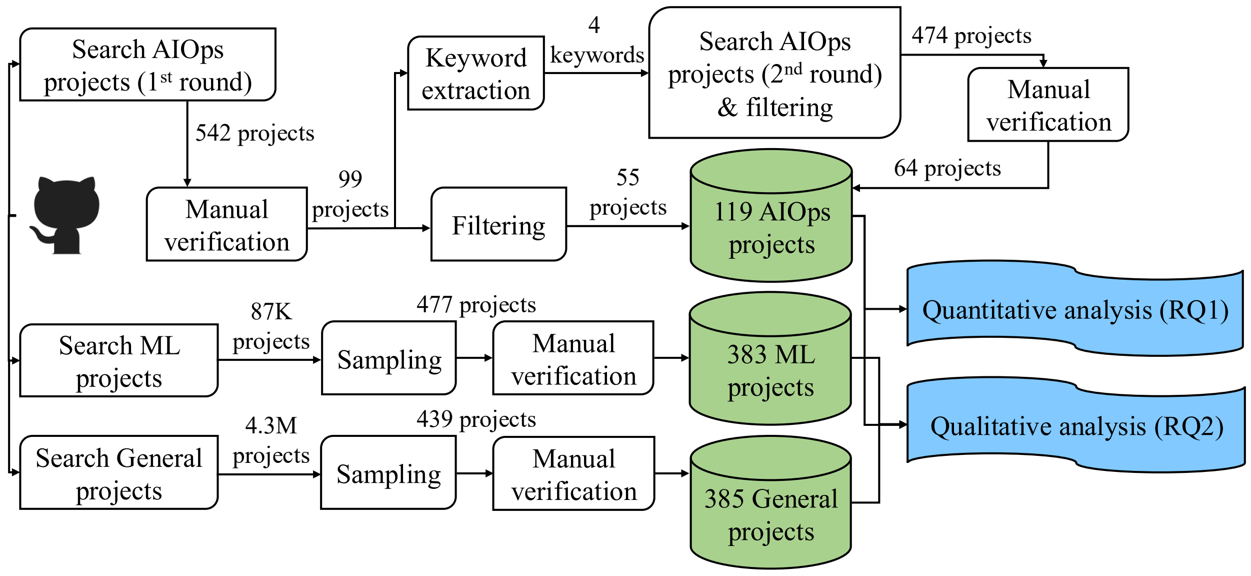


Figure 4.1 An overview of our study.

Manual verification (first round). Based on existing definitions of AIOps [7, 8], we consider AIOps projects as: *any project that uses IT Ops-related data, uses advanced analysis technologies such as machine-learning or statistical analysis to reach valuable insights or enhance the system's quality by actions such as monitoring and automation.*

Therefore, not all the 542 discovered repositories are good AIOps candidates and suitable for our study. We hence select our subject projects based on three criteria:

1. The project should be about AIOps, not similar topics such as MLOps (A set of practices to maintain and deploy machine learning models) or DevOps (A set of practices that aim to shorten the system development life cycle while preserving high quality). Therefore, we delete the projects that are mainly about other topics.

2. The projects should contain sufficient code. Therefore, we delete the repositories that do not have any code. We also delete projects that are a collection of papers, slides, or other repositories. However, we accept the projects that have created a dataset so that other developers and researchers can use it in their work.

3. The projects should not be toy projects: we delete the projects that are homework assignments or university projects.

To select the desirable projects based on the explained criteria, two coders, including the author of this thesis and a researcher with more than five years' experience in AIOps research, independently perform a coding process, adding a YES (AIOps projects) or NO (non-AIOps projects) tag to each project. We perform the coding process as below.

Step 1: Coding. Each coder studies and analyzes all the 542 repositories and independently decides if each project should be added to the final list of projects.

Step 2: Discussion. The coders share their responses and discuss their approaches for selecting a project. The discussion session's goal is to reach the same understanding of the inclusion criteria among the coders.

Step 3: Revision. Based on the discussion, each coder revises their responses from step 1.

Step 4: Resolving disagreements. In the last step, the coders discuss any conflicts that may remain and try to resolve them. If an agreement can not be reached, a third coder would analyze the project and a vote is performed.

After performing the manual verification process, we obtain a total of 99 candidate AIOps projects from the 542 projects derived from the search results, which corresponds to a selection rate of 18%.

Keyword extraction (pattern mining). Expanding our search to gain a more comprehensive view of real-world AIOps projects is necessary as the term "AIOps" was only introduced in 2018 [8]. As a result, some projects might have existed prior to the introduction of this terminology, implementing AIOps solutions without explicitly using the exact term. To achieve a broader scope, we extend our search to include additional projects that may not explicitly label themselves as AIOps repositories but are, in fact, implementing AIOps solutions. To achieve this, we extract all the topics associated with each of the 99 AIOps projects resulting from the previous step. These topics can be found in the "topics" section of each GitHub repository.

We then use frequent pattern mining [151], a method aimed at discovering associations and patterns within a given dataset. Specifically, we use the frequent pattern growth technique [152,153] to identify the most common topics among GitHub repositories. To conduct this analysis, we set the support parameter to 2, indicating that a pattern should appear in at least two projects for consideration. Through this approach, we identify a total of 194 patterns among the topics present in the repositories. Next, we perform a discussion session involving all three authors to decide which patterns hold potential for identifying additional AIOps projects. From this discussion, we narrow down our selection to four pairs of two-item patterns: "anomaly detection" and "log analysis", "log analysis" and "machine learning", "anomaly detection" and "machine learning", as well as "machine learning" and "metrics". All the selected keyword pairs are among the most frequently used topics in the projects. We use these four sets of keywords to find more AIOps repositories.

Search AIOps projects (second round). Using the four pairs of keywords obtained in the previous step, we conduct the second round of search on the GitHub interface to identify more projects related to AIOps. We follow the same process as described in Section 4.2.2.2. After completing this second-round search and removing any duplicated projects that were already identified in the first-round search results, we find a total of 474 unique projects.

Filtering. Based on the knowledge that we have gained from our first-round manual verification, we apply a filtering phase to remove the toy projects. To achieve this, we employ two filtering criteria based on the number of stars and forks for each project. Specifically, we only consider projects that have both stars and forks greater than or equal to 1 (stars: >=1 & forks: >=1). The purpose of removing toy projects is to have relatively mature projects and not soil our results with small repositories [154]. Given the limited number of AIOps projects on GitHub, adding stricter filtering criteria would result in much fewer projects. Therefore, we choose a low-bar filtering approach to reduce the manual effort of analyzing all the projects with stars or forks of 0.

We use the filtering process for both the projects from the first and second rounds of searches. In the first round, we manually verify all the projects before applying the filtering step. This deliberate approach allows us to gain a comprehensive understanding of the status of the AIOps projects. Following the filtering process, the initially verified 99 projects are reduced to 55. For the projects from the second round, we opt to apply the filtering before the manual verification process. This decision saves unnecessary manual effort by excluding projects that do not meet the filtering criteria from the beginning.

Manual verification (second round). As detailed in Section 4.2.2.2, not all of the repositories obtained from our initial search are suitable for our study. To carefully select the projects that align with our research objectives, we carry out a manual verification process, repeating the steps outlined in Section 4.2.2.2. During this verification, the coders examine all 474 projects that resulted from the expanded keyword search to determine their suitability as real AIOps projects. After steps of separate coding, discussion, revision, and resolving disagreements, we identify 64 projects from the expanded keyword search. This corresponds to a selection rate of 14%, Finally, we combine the 55 projects identified using the "AIOps" keyword with the 64 projects discovered through the expanded keywords. The final set of 119 AIOps projects is used to answer our research questions. The approaches for answering our research questions are detailed in Section 4.3.

Measuring the reliability of our manual verification. Reliability is vital to ensure the validity of the coding results [155]. The coding results are reliable if there exists a certain level of agreement between the coders, known as inter-coder agreement. In this study, we use Cohen's kappa [156] to measure the reliability of the agreements between two coders. Cohen's kappa is one of the most common approaches to measuring the inter-coder agreement [155]. Table 4.1 indicates the relation between the value of Cohen's kappa and the level of agreement [157].

Table 4.1 Interpretation of Cohen's kappa.

Value of Cohen's k	Level of Agreement
0-.20	None
.21-.39	Minimal
.40-.59	Weak
.60-.79	Moderate
.80-.90	Strong
.90-1	Almost Perfect

Our manual verification for choosing the proper AIOps projects achieves a Cohen's kappa of 0.68 before the discussion session. After the discussion session between the coders, the kappa score increased to 0.84. As shown in Table 4.1, kappa $\geq$ 0.80 indicates a strong agreement.

4.2.2.3 Collecting Baseline Projects

To understand how AIOps projects differ from traditional software projects, we create two baselines and compare the AIOps projects with them. We select ML projects as our first baseline and General projects as our second baseline.

Machine Learning projects . For our initial baseline, we choose ML projects. We select ML for the first baseline because AIOps can be considered as an application domain of ML. In similar studies, researchers typically search for specific frameworks to gather ML projects on GitHub. For example, [158] select their project set by searching the keyword "TensorFlow", while [159] gather their projects using various keywords such as "TensorFlow" and "Keras". However, as our focus is not limited to any specific framework, we use more general keywords. To gather our set of ML baseline projects, we use two keywords: "machine learning" and "deep learning." This approach allows us to cast a broader net and capture a comprehensive range of relevant ML projects for comparison and analysis.

To compare the AIOps projects with the baselines in a similar context, we apply a similar filtering process to the baseline projects. Like the AIOps projects, we only extract ML projects that have at least one star and one fork. Additionally, we take into account that ML projects generally have a longer history than AIOps projects. To address this difference, we apply the same date range for the creation of the ML baseline projects as observed in the AIOps set. Specifically, the earliest and latest creation dates in the AIOps projects are 2012/12/25 and 2022/10/27, respectively. Thus, we apply the same date range to filter the ML baseline projects. Following the search and removal of duplicate projects in two queries, we obtain a total of 87,276 unique repositories for the ML baseline. Due to time and computational resource limitations, extracting and analyzing all these projects becomes impractical. Hence, following prior work (e.g., Chen et al. [160] and Zhang et al. [161]), a sample of 383 projects is needed to represent the pool of 87,275 ML repositories with a confidence level of 95% and a confidence interval of 5%.

General projects . As our second baseline, we choose general projects from GitHub, meaning that we do not focus on any particular topic in our search. We also do not limit the General baseline to a specific programming language (e.g., Python) since AIOps projects are not limited to a single language as well. In this way, the General baseline captures the general characteristics of all GitHub projects. We then apply the same filtering phase as we did for the ML baseline, but without indicating any specific topic. After completing the filtering, we obtain 4,358,342 public and available repositories for the General baseline. Similar to the ML baseline, a sample of 385 projects is required to represent the pool of 4,358,342 General repositories with a confidence level of 95% and confidence interval of 5%.

Manual verification . In order to maintain consistency with our methodology for selecting AIOps projects, we apply manual verification to our two baseline sets as well. The selection of baseline projects is based on two key criteria:

1. The projects should contain sufficient code and be mature. Therefore, we delete the repositories that do not have code and are a collection of papers, slides, or other repositories.

2. The projects should not be toy projects: we delete the projects that are homework assignments or university projects.

To construct our baseline sets, we randomly extract 500 projects from both ML and General pool of repositories. We then divide each set of baseline projects into two parts of 20% and 80% portions. Similar to Section 4.2.2.2, the two coders independently label 20% of each baseline to ensure a reliable assessment. We then measure the reliability of our coding using Cohen's kappa [156]. The results of our manual verification for choosing the ML and General baselines indicate a strong agreement between the coders, with Cohen's kappa scores of 0.81 and 0.91, respectively. As the measurement shows a strong agreement between coders, the rest of baseline projects (the 80% portions) are labeled by the first coder.

We continue our manual labeling process until we obtain the statistical representative set for each baseline. In the case of ML projects, we thoroughly analyze 477 projects, ultimately selecting 383 of them to create our final set of ML baseline. Similarly, for the General projects, we thoroughly analyze 439 projects, resulting in the selection of 385 projects as our final set of General baseline.

As shown in Figure 4.1, we compare our AIOps projects with the baseline projects in RQ1 where we study the repositories statistics of the projects. Then, in RQ2, we perform a qualitative study for the AIOps projects, analyzing the input data, analysis techniques, and goals of AIOps solutions.

4.3 Study Approach and Results

This section presents the details of our RQs and their results. We organize each RQ by its motivation, approach, and results.

4.3.1 RQ1. What are the characteristics of AIOps projects in terms of their GitHub metrics?

4.3.1.1 Motivation

Prior studies proposed AIOps solutions that leverage AI technologies to support various software operation efforts [11, 162, 163]. However, no work has investigated real-world AIOps projects and their characteristics. Thus, this RQ bridges the gap to study the characteristics of AIOps projects and compare them with the baseline projects in terms of their GitHub metrics. With this comparison, we can understand the similar and different patterns between the characteristics of AIOps projects and the baselines. Our findings can help AIOps researchers and practitioners understand the state of AIOps in practice.

4.3.1.2 Approach

In this RQ, we analyze the repository characteristics of AIOps projects and compare them with the baseline projects. We use GitHub REST API [164] to retrieve the repository characteristics of these projects. In particular, we analyze the characteristics of AIOps projects and the baseline projects from three perspectives: growth of repositories, programming languages, and repository metrics.

Growth of repositories. To understand the evolution of the population of AIOps projects, we analyze the distribution of the AIOps projects based on their creation time. We also compare the creation time distribution with that of the baseline projects.

Programming languages. Developers may use different programming languages for AIOps projects. Understanding the distribution of the programming languages can provide insights for future work to support AIOps project development. Each project may use multiple programming languages. In this study, we extract and present the primary language of each project.

Repository metrics. We study the repository metrics of the AIOps projects, including the number of stars, forks, commits, contributors, releases, pull requests, issues, size, and the status of being archived. For the pull requests, we sum the open and closed pull requests. Similarly, for the issues, we sum the open and closed issues.

Statistical tests. We further perform statistical tests to evaluate the statistically significant difference between metrics for AIOps and baseline projects. We first conduct the *Shapiro-Wilk* test [165] to test the normality of our metrics. Using the widely accepted significance threshold of 0.05, the Shapiro-Wilk test shows that the GitHub metrics of AIOps and baseline projects do not appear to follow a normal distribution (i.e., the null hypothesis of normality is rejected with $p < 0.05$, though this does not prove non-normality with certainty). Since the assumption of normal distribution required by parametric tests does not hold for the data, we select nonparametric tests. Specifically, we use the *Mann-Whitney U* test [166] to compare our samples. We also use *Cliff's delta* test [167] to test the effect size between our samples. Regarding Mann-Whiteny U test, we use the significance threshold of 0.05. Regarding Cliff's delta test, we use the scale presented by Romano et al. [168], that effect of $|d| = 0.147$ is small, $|d| = 0.33$ is medium, and $|d| = 0.474$ is large.

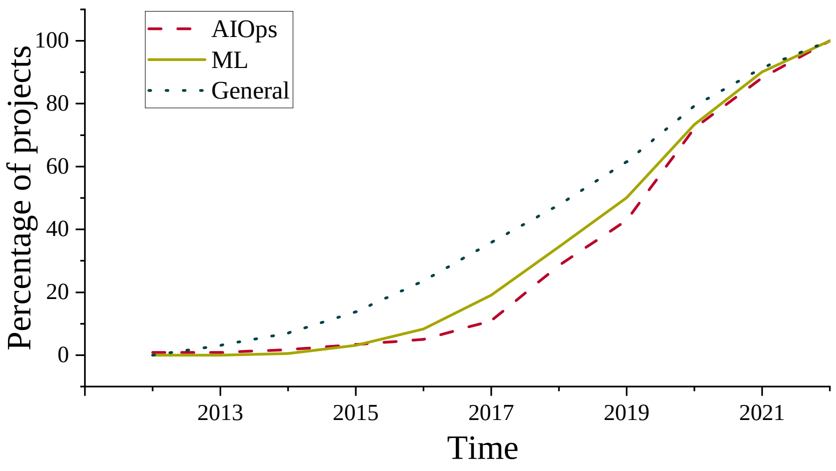


Figure 4.2 The cumulative distribution of the creation time of AIOps and baseline projects.

Table 4.2 The top-5 languages of AIOps and baseline projects.

AIOps		**ML**		**General**	
Language	Usage (%)	Language	Usage (%)	Language	Usage (%)
Python	71.4	Python	81.7	Python	21.6
Java	10.1	HTML	2.6	JavaScript	16.4
Go	3.4	R	1.8	Java	8.1
HTML	2.5	C++	1.8	TypeScript	4.7
JavaScript	1.7	JavaScript	1.8	PHP	4.4

4.3.1.3 Results

Compared to the ML and General baselines, AIOps projects are relatively new and exhibit rapid growth in recent years. Figure 4.2 represents the percentage of projects created in and before each year for the AIOps and baseline projects. As mentioned in Section 4.2.2.3, the creation date of all the projects is between 2012/12/25 and 2022/10/27. As shown in Figure 4.2, in the first few years (from 2012 to 2017), the number of AIOps projects is very small. In recent years (from 2017 to 2022), the AIOps projects experience a faster growth compared to the ML and General baselines. As can be seen, the ML projects also exhibit a faster growth than the General baseline projects.

Like in ML projects, Python is the dominant programming language in the AIOps projects; however, unlike in ML projects, Java is also a major programming language used in AIOps projects. As shown in Table 4.2, Python is the most used language among all the groups; however, the usage of Python is much higher in AIOps and ML repositories than in the General baseline (71.4% and 81.7% in contrast to 21.6%).

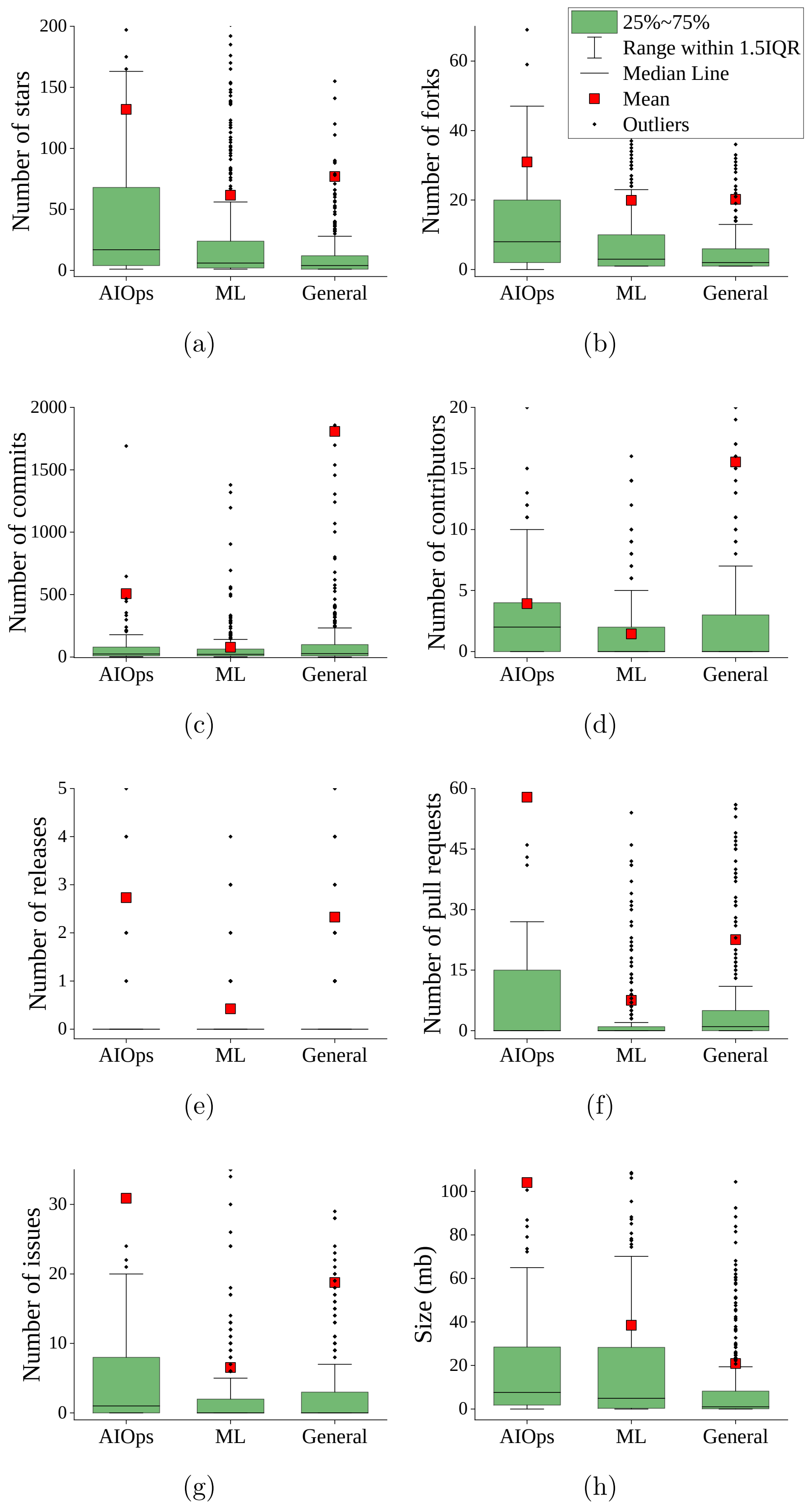


Figure 4.3 Box plots of GitHub metrics for AIOps and baseline projects. The median number of 0 for the releases, pull requests, and issues indicate that more than half of the projects do not have any release, pull request, or issue.

Table 4.3 Detailed results of *Mann–Whitney U* and *Cliff's delta* tests on projects' GitHub metrics.

Metric	**AIOps vs. ML**		**AIOps vs. General**	
	p-value	effect size	p-value	effect size
Stars	0.00	**	0.00	***
Forks	0.00	**	0.00	***
Commits	0.34	-	0.92	-
Contributors	0.00	**	0.02	*
Releases	0.04	*	0.87	-
Pull requests	0.00	**	0.59	-
Issues	0.00	**	0.01	*
Size	0.02	*	0.00	***

Mann–Whitney U results are shown in *p-value* columns. If the sets have statistically different distributions, the *Cliff's delta* results are shown in *effect size* columns.
*: negligible effect **: small effect ***: medium effect

Another interesting finding is the relatively high usage of Java in AIOps projects (10.1%), while Java is not among the top-5 popular languages in the ML baseline.

On average, AIOps projects are more popular and active than the baselines. Figure 4.3 represents the box plots of GitHub metrics for the AIOps projects and the baselines. Table 4.3 shows the p-value and effect size of the GitHub metrics of AIOps projects compared to the baselines.

Considering the number of stars and forks, AIOps projects are more popular than the baselines, as both median and mean values in AIOps projects are higher than the baselines. In terms of the median value, AIOps projects have 4 times more stars and forks than the General baseline (median values of stars in AIOps and General projects are 17 and 4, and median values of forks in AIOps and General projects are 8 and 2). Regarding the statistical test results in Table 4.3, the p-value of stars and forks indicate that AIOps projects have statistically different distributions from ML and General baselines. For both of the metrics, the effect size is small compared to the ML baseline and is medium compared to the General baseline.

Regarding the number of commits, as shown in the statistical test results in Table 4.3, there is not a statistically significant difference between AIOps and baselines. Regarding the number of contributors, it seems that AIOps projects are more collaborative (median of 2 in AIOps and 0 in baselines). Statistical results also corroborate this finding, with the effect size of small compared to ML baseline and negligible (with a statistically significant difference) compared to General baseline. Regarding the number of releases, there is not a significant difference between AIOps and General baseline; however, AIOps projects have a statistically significant difference compared to ML baseline.

Comparing the number of pull requests and issues, as shown in Figures 4.3f and 4.3g, we notice that AIOps projects experience more pull requests and issues. Statistical results also confirm this finding, as p-values of pull requests and issues are less than 0.05 (except pull requests for General baseline). This may be explained in three ways. First, the AIOps projects are more active and popular (i.e., more developers proactively develop them), leading to a more significant number of pull requests and issues. The second interpretation might be that AIOps projects are in the first stages of formation and not mature enough, having more defects and flaws and more developers trying to fix these problems. The third explanation is that AIOps projects are on average larger than the two baseline projects (as shown in Figure 4.3h), which may lead to more pull requests and issues.

Comparing the size of the projects, AIOps projects tend to be larger than General baseline as the statistical tests indicate a medium effect size. AIOps projects also are larger than ML projects; however, the difference is not as large as General baseline, as the statistical tests result in a negligible (with a statistically significant difference) effect size. Comparing the median values, the size of AIOps projects is 7 times larger than General baseline (7.6 MB for AIOps projects compared to 1.0 MB for General baseline).

We further analyze the percentage of projects that have been archived. The results indicate that only 1.7% of AIOps projects have been archived, while this amount for ML and General baseline is 1.8% and 4.2%, respectively.

Summary

On average, AIOps projects are receiving more attention than the ML and General baselines. The primary language used in them is Python, followed by Java. They are growing faster than the baselines in recent years, demonstrating the growing needs and active practices in this area. The size of AIOps projects are larger than the baselines, and considering other GitHub metrics such as number of stars, forks, and releases, AIOps projects seem to be more popular and active than both baselines.

4.3.2 RQ2. What are the characteristics of AIOps projects in terms of their input data, analysis techniques, and goals?

4.3.2.1 Motivation

AIOps researchers and practitioners leverage different techniques to analyze different types of operational data and achieve different goals. However, it is unclear how real AIOps projects leverage data and technologies to achieve the goals. In this RQ, we qualitatively analyze our set of AIOps projects to understand the characteristics of these projects' input data, analysis techniques, and goals. Our results can help researchers and practitioners further understand the status of AIOps practices and the characteristics of AIOps projects. Our results can also provide insights for future work to provide support for different AIOps application scenarios.

4.3.2.2 Approach

We manually examine each AIOps project to understand its input data, analysis techniques, and goals. For each project, we manually investigate four sources of information; the "about" section, the "readme" file, the source code, and the additional documentations if available. Figure 4.4 illustrates the three key concepts of our manual analysis (input data, analysis techniques, and goals) and their relationship.

- **Input data:** The types of data (e.g., log data) that an AIOps project takes as inputs to achieve its objectives.
- **Analysis techniques:** The main analysis techniques (e.g., ML techniques) that an AIOps project adopts to analyze the input data and achieve its objectives.
- **Goals:** The objectives (e.g., anomaly detection) that an AIOps project aims to achieve through its input data and analysis techniques.

Manual coding process. We use open coding approach [169] to extract the information related to the three key concepts shown in Figure 4.4. Open coding is widely used among software engineering researchers to conclude a high-level abstraction from lower-level data [170, 171]. To label the projects, the first two authors of the paper (i.e., coders) jointly perform a coding process, determining each project's input data, analysis techniques, and goals. We perform a five-step coding process as follows.

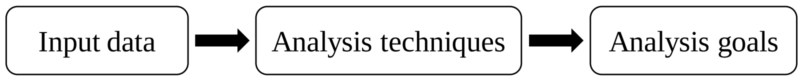


Figure 4.4 The three key concepts of our manual analysis.

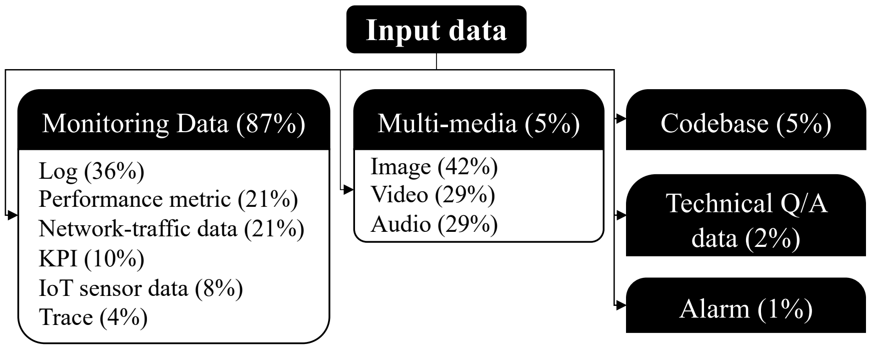


Figure 4.5 The categorization of the input data used in AIOps projects. The high level categories are highlighted in dark.

Step 1: Coding. Each coder analyzes the 97 AIOps projects and assign labels for each concept (input data, techniques, and goals) of each project. Multiple labels can be assigned to a concept of a project. This step takes a few days for each coder to complete.

Step 2: Discussion. The coders share their responses and discuss the created labels. The main goal of the discussion session is to obtain a common understanding of the labels for the input data, techniques, and goals. Based on the separate labels of the coders, we join related labels together and take apart some high-level labels into smaller ones. After this session, we finalize the labels for each concept (input data, techniques, and goals).

Step 3: Revision. Based on the results of the discussion session and the agreed-upon labels, each coder revises his responses from Step 1.

Step 4: Resolving disagreements. The coders compare their final results from step 3 and discuss any conflict that may remain. The coders try to resolve the conflicts, but if an agreement can not be reached, the third author analyzes the project, and the final decision is made.

Step 5: Final revision. In the final stage, we create a mind map from all the produced labels. We then discuss the labels and form an hierarchy, change some labels' names for clarity, and merge some small categories to be cohesive.

[2]LSTM approaches can be used in a supervised or unsupervised manner [172]. In our set of projects and primarily for anomaly detection, projects use LSTM in the unsupervised form.

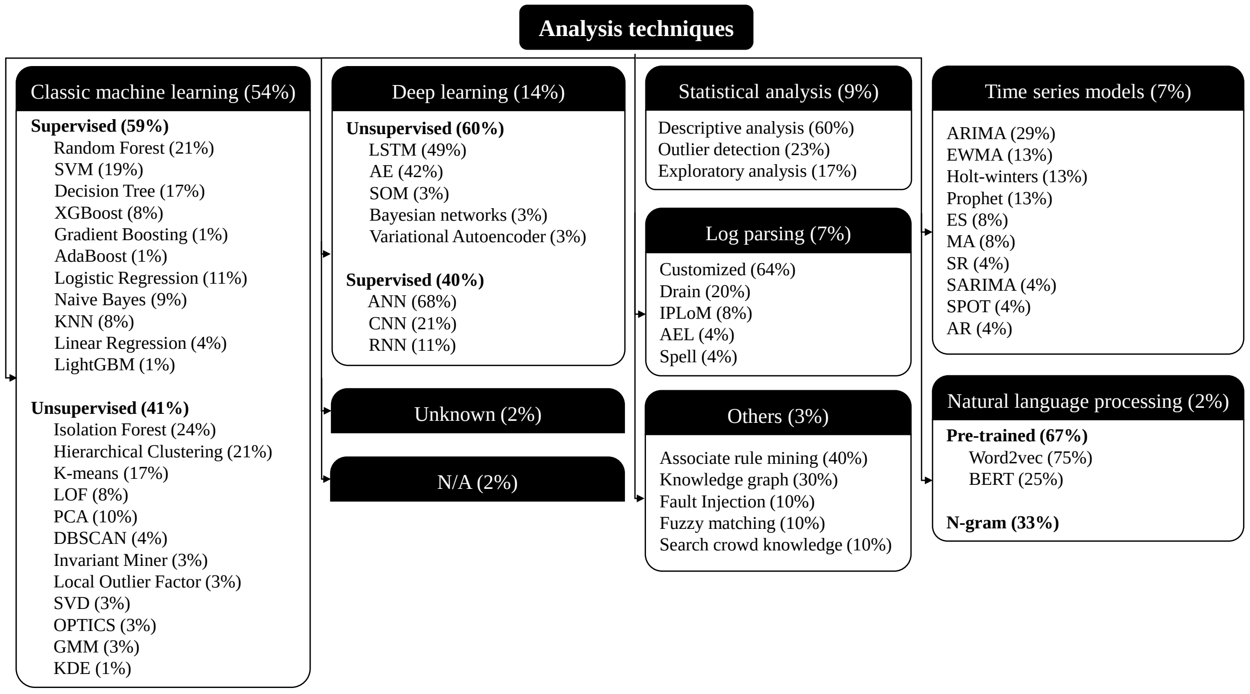


Figure 4.6 The categorization of the analysis techniques used in AIOps projects. The high level categories are highlighted in dark.[2]

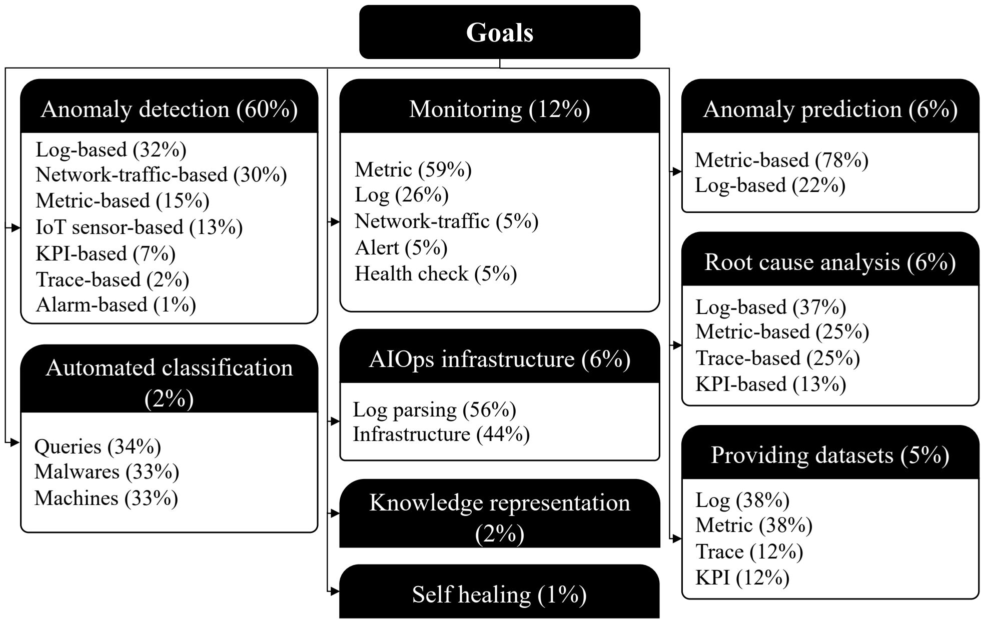


Figure 4.7 The categorization of the goals of the AIOps projects. The high level categories are highlighted in dark.

4.3.2.3 Results

Figures 4.5, 4.6, and 4.7 present our categorization of the input data, analysis techniques, and goals of the AIOps projects, respectively. We further define our coding labels in Tables 4.4, 4.5, and 4.6. Please note that each project may have multiple input data, analysis techniques, or goals.

Input data: Monitoring data (e.g., logs, performance metrics, and network-traffic data) is the dominant input data type of AIOps projects, with logs being the most commonly used input data type. The categorization of the input data is illustrated in Figure 4.5. We divide the input data of AIOps projects into five main categories: monitoring data, multi-media, codebase, technical *Question and Answer (QA)* data, and alarm. We define these categories and provide an example for each in Table 4.4. Among them, monitoring data is the most popular, used in 87% of the projects. The monitoring data is divided into 6 sub-categories: log, performance metric, network-traffic data, KPI, IoT sensor data, and trace. Among them, log data is the most commonly used. These data types could be used for different purposes such as extensive monitoring, debugging, performance analysis, test analysis, and business analytics [173, 174]. In this thesis, we use the umbrella term of "monitoring data" to name these data types.

We find two interesting input data types for the AIOps projects: network traffic and IoT sensor data. A considerable proportion of the projects (21%) use network traffic data as their input. Also, 8% of projects use IoT sensor data.

The second popular category of input data for AIOps approaches is multi-media (image, video, or audio) (5%), followed by codebase (5%), technical QA data (2%), and alarm (1%). We define the input of a project as "codebase" if it analyzes the source code or configuration files of other software. We define the input as "technical QA data" if the project analyzes data from QA websites like Stack Overflow.

Analysis techniques: Classical ML models are the most commonly used analysis techniques, followed far behind by deep learning, statistical analysis, and time series models. We present our categorization of the applied analysis techniques of AIOps projects in detail in Figure 4.6. We derive 9 high-level categories: classic machine learning (54%), deep learning (14%), statistical analysis (9%), time series models (7%), log parsing (7%), others (3%), natural language processing (2%), unknown (2%), and N/A (2%). Definitions and examples for each category can be found in Table 4.5.

Both the classic machine learning and deep learning categories are further divided into super-

Table 4.4 Different types of input data, their definitions, and examples.

Input data	Definition	Repository example
Monitoring Data	**Different types of data that record the runtime status of a system.**	
Log	System-generated data that records runtime events that have happened.	Repository (58811148) uses log data as input to perform anomaly detection.
Performance metric	Quantitative measurements used to track the performance of a system. These metrics are often operational, such as CPU usage.	Repository (160285839) uses different performance metrics such as CPU and memory usage to perform various tasks such as anomaly detection and time series forecasting.
Network-traffic data	Monitoring data of communication flows and packet exchanges that describe interactions across the network.	Repository (79239275) uses network-traffic data to identify malicious behaviors and attacks.
KPI	Measurements related to key business goals of a system. These metrics are often strategic.	Repository (142442484) uses different KPIs in time intervals to identify impactful system problems.
IoT sensor data	Data collected by devices connected to an IoT network.	Repository (142325304) uses real-time IoT sensor data to detect anomalies.
Trace	A specialized use of logging to record information about a system's execution with comprehensive details.	Repository (397983735) gathers a dataset of traces that can be used to analyze operations problems.
Multi-media	**Different types of multi-media data, including images, videos, and audios.**	Repository (287642401) generates summaries of data types including image data for monitoring purposes
Codebase	**Source code and configuration files of software systems.**	Repository (201529303) can be installed on Kubernetes source code and provide self-monitoring and self-healing.
Technical QA data	**Data collected from technical Q/A websites such as Stack Overflow.**	Repository (345320486) extracts information from Stack Overflow to find fast solutions for faults in their platform.
Alarm	**Alarms generated during system run time.**	Repository (160285839) uses alarms to find the association rules between them.

To find a GitHub repository with its ID, one can either click the hyperlink or use the link *https://api.github.com/repositories/{ID}* where *{ID}* is replaced by a specific repository ID.

vised and unsupervised learning algorithms. Regarding classic machine learning approaches, supervised algorithms are used more than unsupervised ones (59% compared to 41%). The top-3 supervised algorithms are Random Forest (23%), *Support Vector Machine (SVM)* (19%), and Decision Tree (17%). The top-3 unsupervised algorithms are Isolation Forest (24%), Hierarchical Clustering (21%), and K-means (13%). Regarding the deep learning ap-

Table 4.5 Different types of analysis techniques, their definitions, and examples.

Analysis techniques	Definition	Repository example
Classic machine learning	**Classic machine learning techniques such as Logistic Regression and Decision Tree.**	
Supervised-learning	A leaning technique that uses labeled datasets.	Repository (165321356) uses various classic supervised approaches such as Random Forest and Decision Tree to detect anomalies in KPI data.
Unsupervised-learning	An approach that system learns without using labeled datasets.	Repository (58811148) implements multiple unsupervised approaches such as Isolation Forest and Invariant Miner to detect anomalies.
Deep learning	**A subfield of machine learning that uses artificial neural networks with multiple layers (i.e., deep neural networks).**	
Supervised-learning	A leaning technique that uses labeled datasets.	Repository (187774599) uses Artificial Neural Network to find anomalous behavior on IoT sensor data.
Unsupervised-learning	A technique that learns patterns from unlabelled data.	Repository (246569386) uses Long Short-Term Memory to detect anomalies from log data.
Statistical analysis	**Analytical techniques to understand, analyze and interpret the input data.**	
Descriptive analysis	Describing the features of data and summarize data in a quantitatively manner.	Repository (114942949) uses descriptive analysis such as measuring minimum and maximum values and plotting scatter figures to interpret their IoT sensor data.
Statistical outlier detection	Applying statistical tests such as z-score to identify outlier values.	Repository (156308650) uses z-score test to detect anomalies on CPU usage data.
Exploratory analysis	Exploring data to identify new connections, inspect missing data, or check hypotheses.	Repository (123162193) uses exploratory analysis such as plotting and describing their log data to inspect and understand their input data.
Time series model	**Techniques that aim to model time series data, mainly used for finding trends and forecasting.**	Repository (302842095) applies different time series models such as Auto Regression and Holt-winters to detect anomalies and analyze root causes.
Log parsing	**Techniques that analyze and extract information from log data.**	Repository (345575577) uses Drain to do log parsing. It then uses the extracted information to perform anomaly detection.
Natural language processing	**Techniques that aim to analyze and model text data.**	Repository (316407231) leverages Bidirectional Encoder Representations from Transformers (BERT), a pre-trained language model as one of their techniques to perform anomaly prediction.
Others	**Other techniques that could not be categorized in previous categories.**	Repository (146802240) uses fuzzy matching to classify web queries.
Unknown	**Projects for which we could not find any specific techniques.**	
N/A	**Projects that only provide datasets or do not use any analysis techniques.**	Repository (238914477) is a dataset of KPI data and does not use any analysis techniques.

To find a GitHub repository with its ID, one can either click the hyperlink or use the link *https://api.github.com/repositories/{ID}* where *{ID}* is replaced by a specific repository ID.

Table 4.6 Different types of goals, their definitions, and examples.

Goal	Definition	Repository example
Anomaly detection	**Identifying anomalies that deviate from the normal behavior.**	Repository (134266587) uses log data and analyze them to find anomalous behavior. It then display anomalies using dynamic graphics.
Monitoring	**Collecting and observing the real-time stream of data to understand system runtime status.**	Repository (221989665) provides data monitoring and alerting.
Anomaly prediction	**Analyzing historical data to forecast future anomalies.**	Repository (169132015) uses metric data of hard drives to predict the failures in the near future.
Root cause analysis	**Identifying the root causes of faults or problems.**	Repository (238914477) analyzes the logs of Kubernetes containers to find the root causes of issues.
AIOps infrastructure	**Providing infrastructure support or utility functions such as log parsing.**	Repository (244678163) provides automated parsing of raw logs.
Providing datasets	**Providing datasets to be used in other AIOps projects.**	Repository (60705895) collects various system log datasets that can be used for log analysis.
Knowledge representation	**Extracting and summarizing knowledge from datasets or websites.**	Repository (345320486) extracts information from Stack Overflow to find fast solutions for faults in their platform.
Automated classification	**Classifying different input data instance based on their similarities.**	Repository (238914477) classifies web queries to find similar important information and trends.
Self-healing	**Conducting health checks and automatically fixing the issues.**	Repository (238914477) aims to provide software systems with monitoring and self-healing.

To find a GitHub repository with its ID, one can either click the hyperlink or use the link *https://api.github.com/repositories/{ID}* where *{ID}* is replaced by a specific repository ID.

proaches, unsupervised techniques are more popular than supervised algorithms (60% compared to 40%). *Long Short-Term Memory (LSTM)*, *AE*, and *Self-Organizing Map (SOM)* are the most used unsupervised algorithms, while ANN is the dominant supervised technique.

Overall, considering supervised and unsupervised usage in classical machine learning and deep learning approaches, 37% of the projects in our study employ supervised learning, while 31% use unsupervised approaches. Hence, the difference between the adoption of supervised and unsupervised methods is not substantial. Therefore, we could not find a strong correlation with Dang et al. [7], where they state that in *many* AIOps cases, only unsupervised machine learning models are practical. However, we find that some projects in our study explore a combination of supervised and unsupervised models, conducting comparisons to assess their performance for specific tasks. While unsupervised techniques outperforms the supervised approaches in scenarios where data labeling is limited or new patterns of data can emerge, there are also instances where supervised approaches achieve better results. For example, as mentioned by He et al. [140] and Chen et al. [172], supervised machine learning algorithms usually outperform in anomaly detection use cases.

The following three common techniques are statistical analysis, time series models, and log parsing. We find three major statistical analysis techniques: descriptive analysis, outlier detection, and exploratory analysis. We categorize the technique of a project as descriptive analysis if it analyzes the data numerically, and we group it as exploratory if it uses visualization to analyze the data. Usually, the exploratory analysis will be done after performing the descriptive analysis.

As most of the input data is time series (e.g., logs, performance metrics, network traffic data), it is not surprising that some projects use time series techniques to model their data. *AutoRegressive Integrated Moving Average (ARIMA)* is the most common time-series technique followed by *Exponentially Weighted Moving Average (EWMA)*.

As shown in Figure 4.5, logs are the most-used input data in AIOps projects. To handle logs, projects either use log parsing techniques or NLP approaches. Regarding log parsing techniques, developers tend to write their customized version of log parsers (with 64%). After coming up with customized log parsers, the most common log parser that is used among AIOps projects is Drain (with 20%). Regarding NLP approaches, only 2% of AIOps projects use them, with Word2vec and BERT as the most common techniques. with the recent developments in the NLP field, for example generating language models such as BERT [175] and CodeBERT [176], we believe AIOps solutions can also benefit more from NLP techniques. We discuss this point in more detail in Section 4.4.

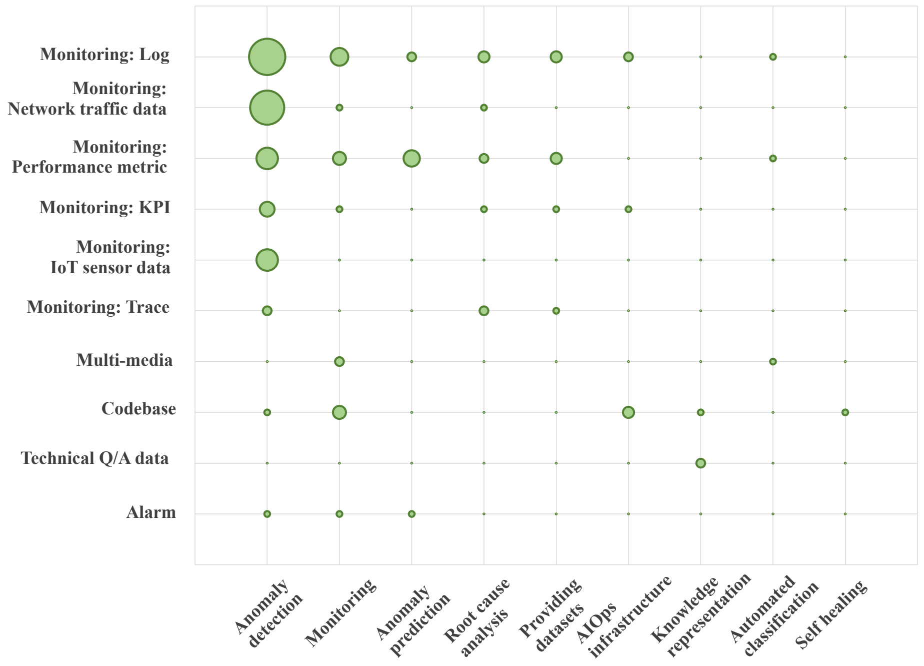


Figure 4.8 The relation between the input data and goals of the AIOps projects. The sizes of the circles are proportional to the number of projects that use a certain input for a certain goal.

Goals: Anomaly detection is the most popular goal of the studied AIOps projects, followed by monitoring, anomaly prediction, root cause analysis, and AIOps infrastructure. We find 9 categories for the goals of the AIOps projects which are shown in Figure 4.7 and are described in Table 4.6. Anomaly detection (60%) and monitoring (12%) are the most common reasons for using AIOps solutions. Anomaly prediction, root cause analysis, and AIOps infrastructure are the next main goals of our set of projects. We categorize the projects as “providing datasets” if they mainly provide public AIOps data that can be used by other practitioners or researchers. We categorize projects as “AIOps infrastructure” if they provide infrastructural support such as log parsing.

We find that for tasks such as anomaly detection and root cause analysis, developers usually use logs as their main source. However, for monitoring and anomaly prediction tasks, the main source of data is performance metrics. We also find that only 1% of projects do self-healing as their final goal. It means, in the other projects, after achieving the final goal, for example, anomaly detection, an agent (e.g., developer) should decide what to do with the founded anomalies. This not completely automated procedure will lead to a loss of time and resources. We discuss this point in more detail in Section 4.4.

The relation between the input data and the goals of the AIOps projects is shown in Figure 4.8. As shown in this figure, log data and network traffic data are the most used data types for achieving the goal of anomaly detection, while performance metric data is the most used data type for anomaly prediction. Log data and performance metric data are also the most common data sources for the goal of root cause analysis.

Summary

Logs are the most commonly used input data in the studied AIOps projects, followed by performance metrics and network-traffic data. Classical machine learning techniques are the most used analysis techniques, followed (far behind) by deep learning, statistical analysis, time series models, and log parsing. The most popular goals of the AIOps projects are anomaly detection, followed by monitoring, anomaly prediction, root cause analysis, and AIOps infrastructure.

4.4 Discussion

Based on the findings of our study, in this section, we discuss the state of AIOps in open-source projects. We then discuss the current challenges of developing AIOps applications and foresee possible future directions for AIOps researchers and practitioners.

4.4.1 AIOps: Where is it now?

The number of AIOps applications is growing fast, and they are receiving a lot of attention from the open-source community. Regarding the results of Section 4.3.1.3, we find that the speed of growth for AIOps applications is faster than that for general-purposed projects on GitHub. We further find that AIOps applications are receiving more attention regarding GitHub metrics compared to machine learning and general-purposed applications. This attention towards the AIOps area encourages researchers and practitioners to develop new technical innovations in various areas of AIOps. We discuss some of the potential future directions below.

Monitoring data, especially logs, performance metrics, and network-traffic data, are the most common data types of AIOps applications. Regarding the results of Section 4.3.2.3, we find that almost 70% of the projects use logs, performance metrics, or network-traffic data. The usage of other types of input data (e.g., alarms) is limited in open-source AIOps applications. This finding is in line with Notero et al. [79] that many AIOps-related papers also use software logs and performance metrics as their data sources.

Future efforts may be invested to investigate how to optimally process these common data types in various tasks. For example, time-series data representation techniques [177] and time-series segmentation techniques [178] may be explored to improve data representations of such time-series data.

Classical machine learning is the most common technique in AIOps applications. We find that over half (54%) of the AIOps applications use classical machine learning as their primary technique, while only 14% use deep learning techniques. AIOps practitioners also develop AIOps applications using a variety of methods, including time-series and statistical models. However, some techniques (e.g., natural language processing) are not used frequently. We suggest that future AIOps solutions may further use more sophisticated techniques of deep learning and natural language processing. However, as suggested by Li et al. [11] and Lyu et al. [52], AIOps models should be trustable and interpretable; hence, using black-box models without the ability to interpret is not recommended. Therefore, we also suggest that future AIOps solutions should always experiment with classical solutions instead of simply assuming deep learning techniques are the optimal solution.

Anomaly detection is the most common goal of AIOps applications. 60% of open-source AIOps applications' primary goal is to detect anomalies. Monitoring, anomaly prediction, root cause analysis, and providing AIOps infrastructure are the other common goals of AIOps applications. Only 5% of projects provide a public AIOps dataset, and only 1% of projects provide self-healing features. Our results align with the findings of Notaro et al. [79], where they observe that 62% of AIOps papers are associated with failure management, including failure detection, failure prediction, and root cause analysis.

4.4.2 AIOps: Challenges and future directions.

More benchmarking datasets could be designed for AIOps applications. We find that only 5% of the projects aim to design and publish AIOps benchmark datasets. As also mentioned by previous studies [4, 7, 18, 19, 21–24], there is a lack of good and public AIOps benchmarks, where different AIOps approaches could be compared. As most of the AIOps applications use software logs, performance metrics, and network-traffic data as their input data source, we suggest AIOps researchers design and publish real-world and public datasets of these data types so other AIOps practitioners and researchers could leverage them in their applications.

NLP techniques can receive more attention in AIOps in the coming years. Our results in Section 4.3.2 indicate that only 2% of the studied projects use NLP techniques. On the other hand, approximately one-third of the projects use logs as their input data. Prior research has shown that logs can be approximately represented as natural language text since the logs are generated by logging statements in the source code written by humans [179]. Other works also illustrate that software logs are even more predictable and repetitive than natural languages, such as English [180–182]. Hence, we suggest that AIOps projects can benefit from leveraging the advances in NLP techniques to analyze and model the input data such as logs.

More attention should be paid to increasing automation and reducing human interventions in AIOps solutions. Regarding the goal of the studied projects, most projects aim to detect and predict anomalies, monitor, and analyze the root causes of failures. All these systems need human interventions when the goal is reached. For example, when an anomaly is detected, an operator should decide the next required action to handle the situation. As demonstrated in Figure 4.7, only 1% of projects aim for self-healing, meaning that the system can make necessary changes when needed without human interventions. We believe AIOps solutions should move toward becoming more automated, detecting/predicting the incidents and resolving them autonomously. Self-healing techniques may get more attention and be integrated with detection/prediction techniques [15, 16, 183].

Open-source AIOps solutions can benefit from broader use-cases and scenarios. As described in previous works [7, 8], AIOps can encompass a wide range of scenarios, from predicting the future status of systems to improving the productivity of engineers. However, the scope of current AIOps projects available on GitHub appears to be limited, focusing primarily on certain aspects with a skewed distribution. For example, we find that anomaly detection is the dominant goal of AIOps projects with 60% of use cases. To address this disparity, AIOps researchers and practitioners may pay more attention to other AIOps scenarios, such as minimizing engineers' tedious tasks and better system automation. This can contribute to advancing this interdisciplinary field and realizing its full potential across diverse use cases and applications.

4.5 Threats to Validity

This section discusses threats to the validity of our study.

External validity. In this study, we identified and studied a set of AIOps projects on GitHub. These projects may not cover all AIOps projects on GitHub, those hosted on other platforms, or private projects. Besides, as “AIOps” is a new terminology, not all AIOps projects mention the keyword of “AIOps” in their repositories. Future work examining other sources of AIOps projects (e.g., those published in the literature or closed-source projects) can complement our results. To broaden the generalizability of our work and maximize the coverage of AIOps projects, we follow a process that combines automated search (two rounds), keyword expansion, manual verification, and filtering. Two coders carefully examined each of the candidate projects to select the ones for our study. The third coder steps in to resolve any disagreement. Nevertheless, future work can leverage our replication package and extend our study by analyzing more AIOps projects.

To select the projects, we consider repositories with stars and forks greater than or equal to 1. We selected this criteria to have a balance between filtering the low-bar projects and having a proper portion of projects to study. Having stricter filtering criteria would heavily reduce the number of AIOps projects, as there are not many AIOps projects available on GitHub.

We also collect the ML baseline based on two keywords of “machine learning” and “deep learning”. However, having these two keywords may not include all machine learning repositories on GitHub. Future work can expand our results by analyzing more projects extracted from more diverse keywords.

Construct validity. To expand our set of AIOps projects, we perform pattern mining and choose 4 out of 194 pairs of keywords. We choose these 4 pairs of keywords after a systematic process and based on the discussions between the authors, in order to reduce the bias. All of the keywords are among the most frequently used keywords.

To answer RQ2, we use qualitative analysis to categorize the input data, analysis techniques, and goals of each AIOps project. Our results may be biased by personal deductions like any other qualitative study. In order to mitigate this threat, two authors of the paper performed the qualitative analysis carefully and followed a 5-step process to deduce the categories. We achieve a Cohen’s kappa value of 0.84 which shows a strong and reliable agreement.

4.6 Chapter Summary

This Chapter studies the characteristics of AIOps projects on GitHub and conducts a comparative analysis with two baseline sets (ML and General projects). We combine both quantitative and qualitative analyses to understand the current state of AIOps solutions. Specifically, we illustrate the state of AIOps projects on GitHub in RQ1, observing that they are relatively new and have been growing rapidly in recent years. We determine the most common input data, techniques, and goals of AIOps solutions in RQ2. For example, we found that software logs are the most common data types used in AIOps pipelines. By uncovering these patterns, researchers and practitioners can learn from successful approaches and adopt optimal AIOps solutions for their specific application scenarios. Our findings highlight the need for future efforts to enhance AIOps practices. Specifically, we find that the shortage of public and accessible AIOps datasets is indeed a challenge for open-source AIOps projects. Similarly, we find some aspects of AIOps pipelines that are not getting enough attention, such as self-healing.

CHAPTER 5 ARTICLE 2: UNDERSTANDING WEB APPLICATION WORKLOADS AND THEIR APPLICATIONS: SYSTEMATIC LITERATURE REVIEW AND CHARACTERIZATION **Roozbeh Aghili, Qiaolin Qin, Heng Li, Foutse Khomh, Accepted in ICSME 2024, Publication Date: 2024/12/19**

5.1 Chapter Overview

In the previous Chapter, we analyzed open-source AIOps projects on GitHub. We found that the shortage of public AIOps datasets is indeed a challenge for AIOps pipelines. To acquire more AIOps data, two general approaches exist: (1) Generating synthetic data by simulating real-world scenarios, or (2) Acquiring real-world operational data from IT organizations. This Chapter focuses on the first strategy, generating synthetic data. To do so, researchers and practitioners aim to replicate real-world scenarios and behaviors. Typically, they begin by examining real datasets, analyzing them to uncover patterns, and then applying various models and techniques to produce new, synthetic data. One of the main sources of real-world data commonly used for this purpose is web application workloads. Web applications, accessible via web browsers over the Internet, facilitate complex functionalities without local software installation. In the context of web applications, a workload refers to the number of user requests sent by users or applications to the underlying system. Existing studies have leveraged web application workloads to achieve various objectives, such as workload prediction and auto-scaling. However, these studies are conducted in an *ad hoc* manner, lacking a systematic understanding of the characteristics of web application workloads. In this Chapter, we first conduct a systematic literature review to identify and analyze existing studies leveraging web application workloads. We focus on web application workloads because they are one of the most accessible and extensively studied forms of operational data, offering a practical basis for understanding real-world behaviors and supporting synthetic data generation [184, 185]. Our analysis sheds light on their workload usage, analysis techniques, and high-level objectives. We further systematically analyze the characteristics of the web application workloads identified in the literature review. Our analysis centers on characterizing these workloads at two distinct temporal granularities: daily and weekly. We identify and categorize three daily and three weekly patterns within the workloads. By providing a statistical characterization of these workload patterns, our study highlights the uniqueness of each pattern, paving the way for the development of realistic workload generation and resource provisioning techniques that can benefit a range of applications and research areas.

5.2 Study Design

Web applications are delivered via the World Wide Web to users, allowing them to access complex functionality without installing or configuring local software components (except a browser). In the context of web applications, the term *workload* refers to requests sent by users or applications to the underlying system. Web applications, such as Wikipedia [186], typically monitor the workload data for various purposes, such as user behavior analysis [187, 188] or resource allocation [189, 190]. Studying and analyzing these workloads are crucial in understanding the dynamics of user interactions, server responses, and resource utilization for web applications.

Web application workloads are valuable not only for workload prediction [191, 192], auto-scaling [193, 194] and the development of self-healing systems [195, 196], but also for software maintenance activities such as performance optimization [197, 198] and capacity planning [199, 200]. Previous efforts have also included literature reviews and survey studies that summarize advancements in closely related areas of workload characterization [201, 202], auto-scaling [203, 204], and workload prediction [205].

Despite the significance of workload data in employing and evaluating these techniques, no existing work has systematically studied the characteristics of web application workloads and their applications. Therefore, in this study, we undertake an SLR to identify existing studies utilizing web application workloads. From our SLR, we identify 12 web application workload datasets (worth more than 8.5 years of workloads in total) and study their applications. We then systematically characterize these workloads, providing valuable insights for future research on software maintenance practices, such as realistic workload generation and resource provisioning strategies. Our two RQs are as follows.

RQ1 ***How are web application workloads used in existing research?*** While web application workloads are known to be valuable for various purposes, there is a gap in understanding their usage. This RQ aims to bridge this gap by identifying diverse applications of these workloads. To achieve this, we conduct an SLR examining articles that use web application workloads. This SLR involves a comprehensive search across two research databases, followed by article selection and forward and backward snowballing, ultimately resulting in a dataset of 78 articles. We analyze how these workloads are used, revealing current trends and potential limitations.

RQ2 ***What are the existing patterns in web application workloads?*** Although web application workloads are crucial for tasks such as workload prediction and auto-scaling,

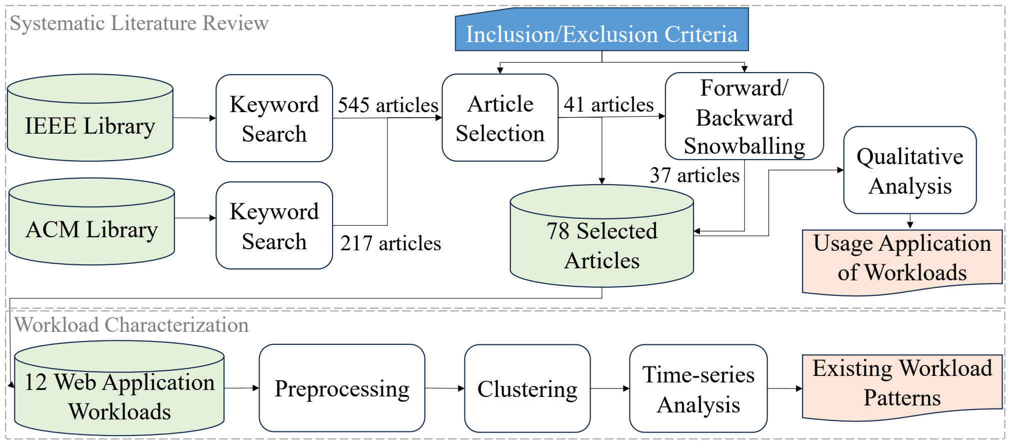


Figure 5.1 Overview of our study

their inherent characteristics remain largely unexplored. This RQ focuses on systematically characterizing these workloads and identifying recurring patterns. Through a thorough examination of web application workloads using clustering techniques, we offer insights that researchers and practitioners can use to develop more accurate and efficient tools and techniques that are customized to the specific characteristics of workload patterns.

Figure 5.1 provides an overview of our study. Our research is initiated by the identification of 78 articles published between 1995 and 2024. These research articles were selected because they used web application workloads in their studies. To identify these papers, we perform a systematic literature search involving three stages: a thorough exploration of research databases, article selection, and snowballing the selected articles to find relevant papers. Once the set of articles is finalized, we identify the web applications they have used. We extract the available workloads and employ clustering techniques to characterize these workloads. Characterizing workload patterns of web applications is of importance, offering several advantages, including (1) uncovering hidden patterns among web application workloads, (2) revealing specific characteristics associated with each workload pattern, and (3) enabling the development of load testing tools, workload generators, and resource provisioning techniques based on our findings.

We share the replication package of this study [206] so that future work can replicate or extend it.

5.3 Study Approach and Results

This section presents the details of our RQs and their results. We first perform a systematic literature review on the web application workloads. Then, we characterize real-world web application workloads to pave the way for the development of realistic workload generation.

5.3.1 Systematic Literature Review

5.3.1.1 Dimensions of the Review

The goal of this review is to provide researchers and practitioners with a structured overview of existing research using web application workloads. Through a meticulous examination of the research literature, we derive several key dimensions. These dimensions provide objective descriptions of different techniques and objectives and organize the literature review. The identified dimensions are as follows.

- *Workloads:* Which web application workloads have been used in the literature?
- *Techniques and Objectives:* What are the overall research objectives of the literature and what types of techniques have been used to achieve them?
- *Trends:* What are the temporal trends in workload usage and research objectives over time?

5.3.1.2 Approach

Through a comprehensive search across two research databases, we identify 762 potential candidate articles. After applying inclusion and exclusion criteria, we narrow down the selection to 41 papers. To ensure a comprehensive review, we employ forward and backward snowballing techniques, adding 33 more articles to our selection. The article selection methodology aligns with the systematic approach recommended by Wohlin et al. [207] and is described in the following.

Searching Research Databases. We conduct our literature search on two highly reputable research databases: IEEE library [208] and ACM library [209]. We aim to identify papers from both conference proceedings and journals, written in English and published between January 2014 and December 2023, a time span of 10 years. To pinpoint the articles that either introduce or use web application workloads, we use the following query to search the databases:

[[Title: web application] OR [Title: workload] OR [Title: trace]] AND [Abstract: dataset] AND [E-Publication Date: (2014/01/01 TO 2023/12/31)]

Upon conducting this query on the IEEE library, we identify a total of 545 articles that satisfy our criteria. Simultaneously, our search from the ACM library yields an additional 217 articles after deduplication. Combining papers from both sources, we successfully compile a total of 762 candidate articles.

Article Selection. We screen each paper, beginning with its abstract, and then proceed through the entire article until a decision is made based on the inclusion and exclusion criteria.

Inclusion criteria:

- Papers must be written in English.
- Papers must be in conference proceedings or journals.
- Papers must introduce or use web application workloads that are publicly available and representative of real-world applications.
- The used workloads must include timestamp information, as this data will enable us to conduct subsequent analysis and characterization of the workloads.

Exclusion criteria:

- Papers not publicly available.
- Papers introducing or utilizing private data (e.g., with *Non-Disclosure Agreement (NDA)* contracts).
- Papers introducing or utilizing non-web application data such as cloud-based batch-jobs (e.g., Google cluster trace), social networks data, mobile devices workloads, and synthetic workloads. Overall, we exclude articles that leverage data unrelated to user interaction workloads.

After performing the article selection step, we end up selecting 41 out of the 762 candidate articles.

Forward and Backward Snowballing. To ensure the comprehensiveness of our study and to capture potentially relevant articles, we initiate the forward and backward snowballing

Table 5.1 List of web application workloads of different literature

ID	Workload	Duration [a]	# Instances	Description	Freq.	Reference [d]
1	Wikipedia [b]	5.5 years	1.0 T	Wikipedia workload consists of the number of users accessing Wikimedia Foundation articles from January 2018 to August 2023.	23	[191, 210, 211]
2	Calgary	1 year	727 K	Calgary workload includes one year of HTTP requests to the University of Calgary's Department of Computer Science server in 1994-1995.	13	[212–214]
3	Saskatchewan	7 months	2.4 M	This workload contains seven months of HTTP requests sent to the University of Saskatchewan's server in Canada in 1995.	18	[213, 215, 216]
4	Boston	6 months	1.1 M	Boston workload contains HTTP requests sent to the Boston University Computer Science Department from November 1994 to May 1995.	4	[217, 218]
5	Retailrocket	4.5 months	2.8 M	This workload contains HTTP requests to servers of an anonymous real-world e-commerce website over a period of 4.5 months in 2015.	1	[219]
6	Worldcup98	2 months	1.3 B	Worldcup98 workload consists of all requests made to the 1998 World Cup website between 30 April 1998 and 26 July 1998.	29	[193, 219, 220]
7	NASA	2 months	3.5 M	NASA workload contains the requests sent to NASA Kennedy Space Center in Florida, USA, between 1 July 1995 and 31 August 1995.	35	[185, 193, 221]
8	YouTube	44 days	1.5 M	A collection of workloads from a campus network measurement on YouTube traffic between June 2007 and March 2008 spans 10 months, though the actual data covers a period of 44 days.	3	[188, 222, 223]
9	Madrid	1 month	___ [c]	Real web service logs from the Complutense University of Madrid. This dataset was collected hourly throughout the month of May 2018.	1	[224]
10	ClarkNet	14 days	3.3 M	Clarknet was an Internet service provider located in Maryland, USA. This workload consists of two weeks of data in September 1995.	18	[190, 220, 225]
11	EPA	1 day	47.7 K	EPA workload contains one day of requests sent to the EPA server located at North Carolina, USA in August 1995.	1	[226]
12	SDSC	1 day	28.3 K	SDSC workload includes requests to the servers of the San Diego Supercomputer Center in California over a single day in August 1995.	1	[227]

[a] This indicates the duration of each workload dataset. Certain literature used segments of these durations.
[b] Unlike other workloads with a single dataset, Wikipedia offers an API allowing users to download customized data sets. The duration and instances mentioned here are the data extracted from Wikipedia for this study.
[c] Unlike other workloads, this dataset does not provide raw data; instead, it offers preprocessed data.
[d] The full list of reference articles is available in our replication package.

Table 5.2 List of different literature objectives with their corresponding techniques

Objective	Utilized Techniques	Freq.	Reference
Resource Management			
Load Balancing	Classical Machine Learning, Deep Learning, Time-series/Statistical Analysis, Queueing Theory	4	[214, 228]
Resource Provisioning	Classical Machine Learning, Deep Learning, Optimization, Control Theory, Fuzzy Logic, Queueing Theory	6	[185, 190]
Caching Optimization	Time-series/Statistical Analysis	2	[197, 198]
Workload Analysis			
Workload Prediction	Time-series Models, Classical Machine Learning, Deep Learning, Optimization, Time-series/Statistical Analysis, Filtering and Signal Processing, Markov Models, Fuzzy Logic	44	[216, 219, 229]
Workload Classification	Time-series Models, Classical Machine Learning, Optimization, Time-series/Statistical Analysis	4	[185, 187, 230]
Workload Characterization	Optimization, Time-series/Statistical Analysis	13	[231–233]
Self Adaptation			
Auto-scaling	Optimization, Control Theory, Fuzzy logic, Queueing Theory	14	[193, 219]
Self-healing	Optimization, Markov Model	2	[195, 196]
Benchmarking	Time-series Models, Optimization, Control Theory, Time-series/Statistical Analysis, Queueing Theory	6	[212, 234]

phase on the selected 41 papers. During this phase, for each of the selected articles, we inspect the list of papers that have cited them (i.e., forward snowballing) and their reference papers (i.e., backward snowballing) and perform the article selection step (as described in Section 5.3.1.2) to decide whether the new article matches with our inclusion and exclusion criteria. After performing this step, we end up having 33 new articles. Combining the articles retrieved through our initial search of research databases and the subsequent snowballing process, we obtain a total of 78 articles, each of which introduces or uses public web application workloads.

Qualitative Analysis. We manually examine each of the selected articles to extract its dimensions (i.e., workloads, techniques, objectives). This labeling process involves reviewing all sections of each paper to identify the relevant information, though most of this information is typically found in the methodology sections. We use an open coding approach [169] to extract the desired information. To label the articles, two coders, including the author of this thesis and a researcher with more than three years' experience in AIOps research, jointly perform a coding process, determining each article's workloads, techniques, and objectives. We perform a five-step coding process as follows.

Step 1: Coding. Each coder independently analyzes the first batch of articles (i.e., the initial 37) and assigns labels for each dimension (i.e., workloads, techniques, objectives) of each paper. Multiple labels can be assigned to each dimension.

Step 2: Discussion. The coders share their responses and discuss the labels they created, aiming for a common understanding. We join related labels and refine high-level ones, then revise the coding of the first batch accordingly.

Step 3: Coding. Each coder analyzes the second batch of articles (i.e., the final 37) based on the discussion results.

Step 4: Resolving disagreements. The coders compare their final results from step 3 and discuss any remaining conflicts, attempting to resolve them. If an agreement cannot be reached, a third coder makes the final decision.

Step 5: Final revision. In the final stage, we create a mind map from all the produced labels. We then discuss the labels and form a hierarchy, change some labels' names for clarity, and merge some small categories to be cohesive.

Measuring the Reliability. Ensuring reliability is crucial for validating coding results [155]. The coding results are reliable when there is a specific level of agreement between coders, referred to as inter-coder agreement. In this study, we employ Cohen's kappa [156] as a metric to quantify the reliability of agreements between two coders. We evaluate our coding reliability for the second batch of the articles (i.e., after the discussion session) and achieve a Cohen's kappa of 0.94. A value of kappa ≥ 0.80 indicates a strong agreement [157].

5.3.1.3 Results

We present the findings of our SLR in three main parts: the analysis of web application workloads, the discussion of objectives and techniques, and the temporal trend analysis.

Web Application Workloads. Table 5.1 presents workload datasets used in the literature, showcasing details such as duration, data instances, description, frequency of usage in articles, and example references. Our review identifies 12 distinct workloads commonly used in the literature: Wikipedia [186], Worldcup98 [235], NASA [236], Saskatchewan [237], Calgary [238], EPA [239], Clarknet [240], Retailrocket [241], Boston [242], SDSC [243], Youtube [244], and Madrid [245].

Some workloads, such as the NASA dataset, appeared in many articles (35 in total), while others, such as RetailRocket, SDSC, and Madrid, were used much less frequently, each appearing in just one article. These datasets vary significantly in duration and the number of instances. For instance, the WorldCup98 workload, characterized by high demand, consists of 1.3 billion instances. In contrast, the SDSC workload spans only one day with 28 thousand instances.

Existing studies repetitively use 12 public web application workload datasets, covering a wide range of workload duration (from a single day to 5.5 years) and size (from 28.3 thousand to 1.0 trillion instances).

Objectives and Techniques. Table 5.2 presents objectives and primary techniques from literature articles, along with their frequency of usage and example references. A more detailed list of objectives and techniques along with all the reference articles can be found in our replication package [206]. Below we define the objective categories.

The first objective theme is **Resource Management**, which aims to ensure the efficient allocation, utilization, and optimization of computing resources within a system or network. It includes three objectives: *Load Balancing*, *Resource Provisioning*, and *Caching Optimization*.

Load balancing is the distribution of incoming network traffic or computational tasks across multiple resources to optimize resource utilization and ensure high availability. Various approaches are employed for this purpose. For example, Riska et al. [184] apply queueing theory to evaluate load balancing policies in distributed multi-server systems, modeling each server as a queue using *Markov chains*.

Resource provisioning is the process of initially allocating the resources and services from a cloud provider to a customer to meet the requirements of applications. Shahidinejad et al. [185] employ classical machine learning models to design their Resource Provisioning system. They propose a system to first cluster the workloads using K-Means and then use *Decision Tree Regression (DTR)* to determine scaling decisions for efficient resource provisioning. In another work, Zhou et al. [190] combines deep learning models (i.e., LSTM) with classical machine learning models (i.e., XGBoost) to provide proactive resource provisioning. They center their work on microservice systems.

Caching optimization is the techniques and strategies aimed at improving the efficiency and effectiveness of caching mechanisms. Time-series and statistical analysis are the main techniques used for this category. For example, Bairavasundaram et al. [197] create an image of the file system cache using inferred disk traffic data and propose an array caching mechanism.

The second objective theme is **Workload Analysis**. It is the examination and understanding of patterns, trends, and characteristics of system or network usage. This objective also includes three objectives: *Workload Prediction*, *Workload Classification*, and *Workload Characterization*.

Workload prediction involves using historical data and trends to forecast future workload demands and patterns, aiming to predict resource requirements and maintain optimal system performance. This objective represents the most common theme across our literature articles, comprising 45% of the total objectives outlined in these papers. Due to the number of articles aiming to improve workload prediction approaches, various techniques have been used. Kim et al. [229] build an ensemble architecture encompassing various time-series models such as *Weighted Moving Average (WMA)* and ARIMA. Their ensemble architecture chooses a subset of these models based on the characteristics of the upcoming workload. Deep learning is another popular solution. Shi et al. [199] use *Deep Reinforcement Learning (DRL)* and Attia et al. [246] use deep learning with a *Differential Evolution (DE)* algorithm as their optimization technique to predict the workloads. In another work, Saravanan et al. [215] use markov models in order to filter redundant historical data before passing them to a GAN model.

Workload classification involves using classification techniques to categorize workloads. In [247], time-series models such as *Sample Entropy* and classical machine Learning models such as *K-Nearest Neighbors (KNN)* are employed to classify workloads. In [187], Urdaneta et al. use time-series analysis techniques to classify Wikipedia workload, emphasizing challenges from its large, heterogeneous dataset.

Workload characterization is the understanding and describing the behavior and characteristics of workloads, including patterns, trends, and variations. The majority of articles within this category rely on time-series and statistical analysis due to the ability of time-series analysis to capture temporal dependencies and identify recurring patterns in workload data.

Self Adaptation is the third objective theme. It is the automatic adjustment of system parameters and resources in response to changing conditions or system faults, aiming to improve performance and reliability without human intervention. It includes two categories of *Auto-scaling* and *Self-healing*.

Auto-scaling is the dynamic adjustment of the number of resources allocated to an application based on its workload, ensuring optimal performance and cost efficiency. While resource provisioning focuses on the initial allocation and management of resources, auto-scaling continuously monitors system metrics and automatically adjusts resource levels to match current demand. Control Theory techniques such as *Proportional Integrative Derivative (PID)* [248] have been applied to automatically scale out public-cloud resources, utilizing only the customer's existing knowledge base. In another attempt to create an auto-scaling system, Kumar et al. [249] use queueing theory and design a system that dynamically performs resource corrections at the virtual machine level by considering both underutilization and over-utilization scenarios.

Self-healing systems are designed to autonomously detect, diagnose, and recover from failures or performance degradation without human intervention, ensuring continuous operation and reliability. In both works of [195] and [196], the authors employ a *Markov model* along an optimization technique (i.e., *Stochastic Programming*) to design a self-healing system compatible with various adaptation objectives.

As the last objective, **Benchmarking** evaluates the performance, reliability, and scalability of systems by measuring and comparing their performance. In [250], Papadopoulos et al. propose a performance evaluation framework that uses *Control Theory* and optimization to evaluate different auto-scaling systems. In another work, Kumar et al. [234] evaluate optimization algorithms such as GA and DE in the task of workload prediction. In a similar attempt, Kumar et al. [212] propose a performance evaluation framework that assesses the performance of six prediction models such as *ARIMA* and Exponential Smoothing on Workload Prediction.

Current research covers various objectives across four thematic areas: Resource Management, Workload Analysis, Self-Adaptation, and Benchmarking. These studies employ diverse methodologies, ranging from Classical Machine Learning to Markov Models and Time-series Models.

Temporal Trend Analysis. Initially, we compare the publication years of the studied articles with those of the workload datasets they employ. Our findings indicate that a significant proportion of the articles use relatively dated workload datasets. Specifically, while the median publication year of the articles is 2018, the median publication year of the datasets is 1995. To provide further insight, over 82% of articles published in 2015 and later rely on datasets that were published in 2000 or earlier. This analysis emphasizes the frequent utilization of older workloads in research studies. Figure 5.2 presents the scatter plot of this phenomenon.

Subsequently, we analyze the popularity of different objectives over time. Figure 5.3 presents the cumulative number of articles associated with each objective throughout the years. The analysis highlights workload prediction, auto-scaling, and workload characterization as the most prominent objectives. Particularly, there is a noticeable increase in articles focusing on workload prediction, especially after 2014. However, objectives such as self-healing and caching optimization have received relatively less attention, indicating potential areas for further exploration.

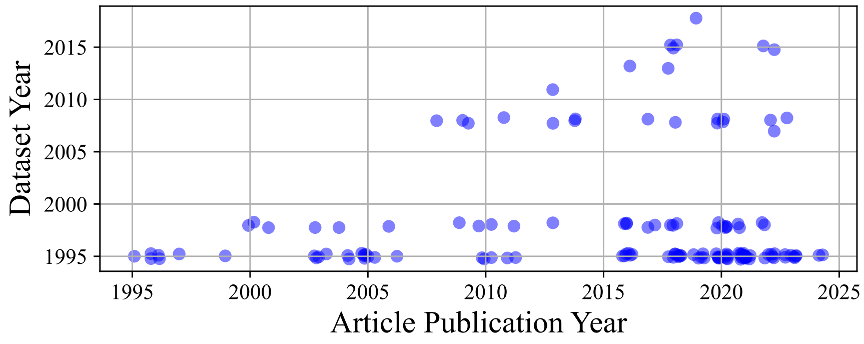


Figure 5.2 Comparison of literature publication years and corresponding workload datasets years

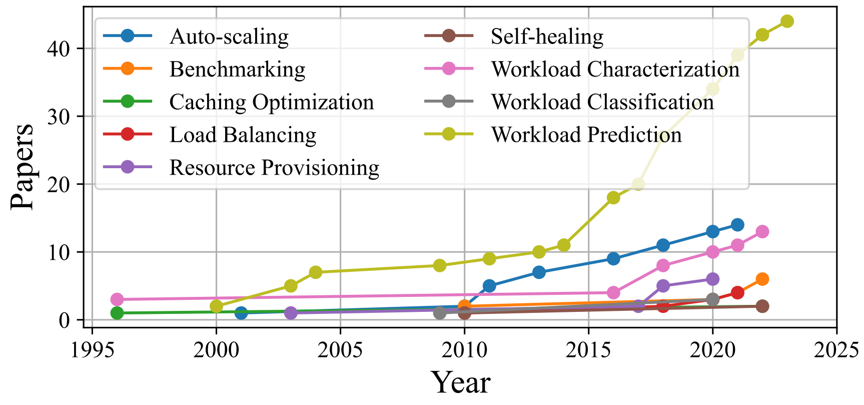


Figure 5.3 Cumulative number of papers per objective over the years

Our trend analysis reveals that Workload prediction has emerged as the primary objective in recent studies. Besides, existing studies rely on significantly dated workload datasets.

5.3.2 Workload Characterization

5.3.2.1 Motivation

Clustering is an essential process for uncovering existing patterns within data. By identifying these patterns in web application workloads, we can gain valuable insights into the behavior and dynamics of web application systems. This understanding is foundational for various tasks, including the development of workload generators and resource provisioning techniques. Leveraging the insights obtained by clustering workload data, researchers and practitioners can design more accurate and efficient tools and techniques that are customized to the specific characteristics of workload patterns.

5.3.2.2 Approach

To analyze web application workloads and identify the existing workload patterns, we start by extracting the workloads. We continue by preprocessing through aggregation, standardization, and smoothing. We then analyze the workloads in terms of variability and burstiness. After that, we employ K-Means to cluster the workloads, and finally, we analyze cluster characteristics.

Web Application Workloads. As mentioned in Section 5.3.1.3, the selected 78 articles use 12 web application workloads. A comprehensive description of these workloads, along with key features for each, is presented in Table 5.1.

Workload Extraction. Most of these 12 web application workloads are accessible for download. However, some, such as Wikipedia, require scripting for data extraction. For the Wikipedia workload, we develop a script to retrieve data spanning 5.5 years, from January 2018 to August 2023.

After gathering all the workloads, we realize that each of them possesses a unique data format. Some are summary-based and only provide information on the number of users who accessed a particular server (e.g., the Wikipedia workload) per time interval, while the others are in the format of log lines and are event-based (e.g., the Worldcup98 workloads). To illustrate this contrast, we provide a comparative example of the raw data from Wikipedia and Worldcup98 workloads in Table 5.3. We write separate scripts for each of these data types, extracting the number of users interacting with the web application within one-hour time intervals.

Preprocessing the Workloads. The preprocessing phase involves three stages: aggregation, standardization, smoothing.

Aggregation. After extracting the user interaction information for all the workloads, we establish two temporal granularities for our analyses: *daily* and *weekly*. These granularities have been widely used for workload-related tasks such as workload generation for load testing and resource provisioning [251–253].

Daily Granularity: Analyzing daily granularity allows for a micro-level understanding of user behavior and system performance. This examination offers detailed insights into user en-

Table 5.3 Comparing two workload types: summary-based (e.g., Wikipedia) and event-based (e.g., Worldcup98)

Wikipedia - 2023/01/01-00
en.m Cristiano_Ronaldo 4888 0
en.m Lionel_Messi 2322 0
en.m Frédéric_Chopin 83 0
Worldcup98
2705258 - - [13/Jul/1998:22:00:01 +0000] "GET/images/102378.gif HTTP/1.0" 200 1658
1630377 - - [13/Jul/1998:22:00:01 +0000] "GET/images/hm_score_up_line03.gif HTTP/1.0" 200 90

gagement, system load, and operational peaks over a 24-hour cycle. This granularity enables the detection of short-term trends, such as hourly spikes in traffic or variations in user engagement between weekdays and weekends.

We aggregate workloads in one-hour time intervals. Thus, if a single workload spans one day, it will be represented in 24 instances, each corresponding to the number of user accesses in a 1-hour window. We provide an example of the daily granularity in Table 5.4a, where each row corresponds to one day of data. Figure 5.4 presents the Wikipedia and Worldcup98 workloads in daily granularity, where each data point signifies the workload for one hour. As evident, patterns observed within these two days exhibit both similarities and differences, and the ultimate objective of this work is to uncover and understand these patterns within web application workloads.

Weekly Granularity: Weekly granularity offers a broader perspective, capturing trends and variations that span across different days of the week. Analyzing these patterns allows for the identification of cyclical behaviors and longer-term trends. For instance, it reveals differences in user engagement and system load along different seasons of the year.

To analyze weekly granularity, we aggregate the workloads in time intervals of one day. In this case, if a workload encompasses a duration of one week, it will be reflected in seven instances, each representing the number of user accesses in a 1-day window. We provide an example of the weekly granularity in Table 5.4b, where each row corresponds to one week of data. Figure 5.5 shows the Wikipedia and Worldcup98 workloads in weekly granularity, where each data point signifies the workload for one day.

Standardization. We standardize our data to ensure uniformity across different workloads, especially when they exhibit significant load variations. For example, in Figure 5.5, we observe a considerable difference in scale between Wikipedia and Worldcup98 workloads: Wikipedia's workload scale is more than 10 times greater compared to that of Worldcup98.

Table 5.4 The schema of daily and weekly granularities

Workload	**Day**	**0**	**1**	**2**	**...**	**22**	**23**
Wikipedia	2023-01-01	20.7	20.6	18.5	...	27.7	24.9
Wikipedia	2023-01-02	22.3	19.8	19.2	...	27.4	25.3
Wikipedia	2023-01-03	21.5	19.7	19.0	...	26.5	24.0

(a) Daily granularity example extracted from the Wikipedia workload. The numbers are in million.

Workload	**Week**	**M**	**Tu**	**...**	**Sa**	**Su**
Wikipedia	2023-01-01	541.5	533.8	...	527.6	556.1
Wikipedia	2023-01-08	543.1	525.8	...	531.4	581.3
Wikipedia	2023-01-15	567.2	549.5	...	518.2	566.9

(b) Weekly granularity example extracted from the Wikipedia workload. The numbers are in million.

Thus, standardization is essential to ensure that the subsequent clustering is not biased by different workload intensities.

Z-score is a widely used technique that adjusts data to have a mean of 0 and a standard deviation of 1. The formula applies to each data point in both daily and weekly granularities.

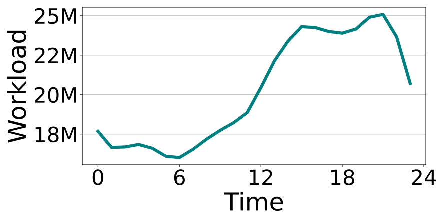
$$z = \frac{X(t) - \mu}{\sigma} \tag{5.1}$$

where X(t) represents the original data value at time t, and μ and σ represent the mean and standard deviation, respectively.

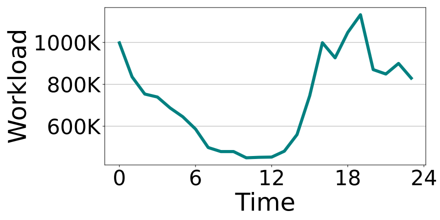
Smoothing. Smoothing is employed to reduce noise and fluctuations in the workload data, thereby enhancing its clarity and facilitating trend identification. One commonly used smoothing technique is *Exponential Moving Average (EMA).* The main benefits of using the exponential smoothing method are its low cost and ease of application [254]. It assigns exponentially decreasing weights to past observations, ensuring that recent data points have a higher influence on the smoothed value compared to older ones. The EMA smoothing formula is as follows:

$$\text{EMA}(t) = \alpha \times \text{X}(t) + (1 - \alpha) \times \text{EMA}(t - 1) \tag{5.2}$$

where EMA(t) represents the smoothed value at time t, X(t) denotes the original data value at time t, and α is the smoothing factor, typically a value between 0 and 1, determining the weight assigned to the current observation.

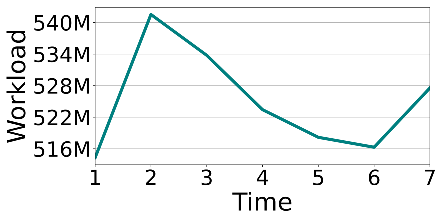


(a) Wikipedia

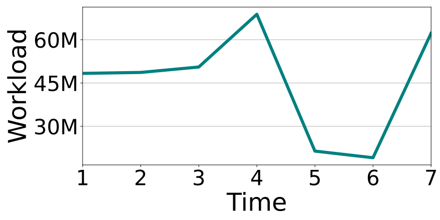


(b) Worldcup98

Figure 5.4 Daily granularity of Wikipedia and Worldcup98 workloads over a one-day time span



(a) Wikipedia



(b) Worldcup98

Figure 5.5 Weekly granularity of Wikipedia and Worldcup98 workloads over a one-week time span

Variability and burstiness analysis. Variability and burstiness analysis is performed before clustering to understand the inherent dynamics of web application workloads. This analysis helps identify distribution differences that might influence clustering results. By examining raw datasets for variability and burstiness after aggregation but before standardization, we retain the original data characteristics.

Variability is defined as the extent to which data points in a time series differ from each other and from the mean value [255]. We calculate variability by measuring the *Coefficient of Variation (CV)* of the workload intensities within one day (i.e., 24 hourly workload intensity values) or one week (i.e., seven daily intensity values) using the below formula:

$$\mathrm{CV} = \frac{\sigma}{\mu} \tag{5.3}$$

where σ is the standard deviation and μ is the mean.

Burstiness refers to the tendency of a time series to exhibit sudden, irregular increases in activity or intensity over short periods. To quantify burstiness, we calculate the mean and standard deviation of workloads per day and per week. Burstiness is then determined using the formula [256]:

$$\text{Burstiness} = \frac{\sigma - \mu}{\sigma + \mu} \tag{5.4}$$

Burstiness ranges from -1 to 1, where 1 indicates high irregularity, 0 indicates no significant burstiness, and -1 suggests a regular pattern. This quantification highlights periods with significant deviations from overall trends.

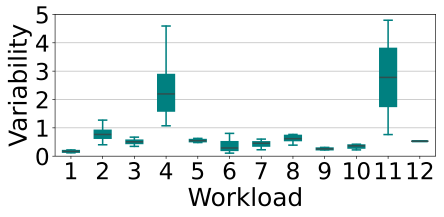
Clustering the Workloads. To investigate general workload patterns across a comprehensive dataset, we employ workload clustering. We first develop a script to combine the standardized and smoothed daily and weekly workloads from all 12 workload datasets. This combination process results in two unified datasets: the daily workload dataset and the weekly workload dataset. This combination enables us to construct a unified dataset that can effectively capture robust and generalized patterns. The daily dataset contains 3191 data points, each representing one day, and the weekly dataset contains 466 data points, each representing one week.

After constructing the combined daily and weekly datasets, we employ K-Means clustering. K-Means clustering is a widely recognized and employed clustering method, designed to divide n observations into k clusters, in which analyzed data sets are partitioned in relation to the selected parameters and grouped around cluster centroids [257].

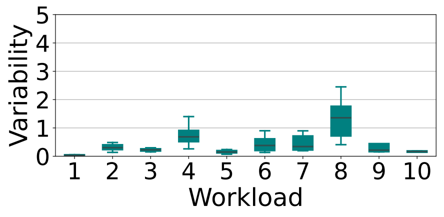
Determining the optimal number of clusters (a.k.a., k value) is critical in applying the K-Means algorithm. Various methods exist for this purpose, including the elbow method [258] and the silhouette score [259]. In this study, we opt for the silhouette score, which measures the cohesion and separation of clusters, with values ranging from -1 to 1. A higher value indicates well-separated clusters and a near-zero or negative score suggests overlapping or incorrectly assigned data points.

We experiment with *Euclidean* [260], *Dynamic Time Warping (DTW)* [261], and *Soft-DTW* [262] distance metrics to determine the most suitable metric for our data. Our evaluation indicates that Euclidean distance yieldes the most meaningful cluster separations based on the silhouette score. We execute the K-Means algorithm for k values ranging from 1 to 20 and determine the optimal k based on the silhouette score. Additionally, to validate our findings, we visually inspect the clustering results using *t-SNE* illustration [263].

Analyzing Cluster Characteristics. Beyond identifying workload patterns, understanding their underlying characteristics and interrelationships is crucial. We analyze the clustering

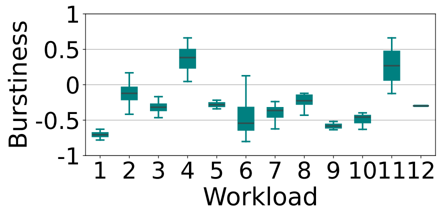


(a) Daily granularity

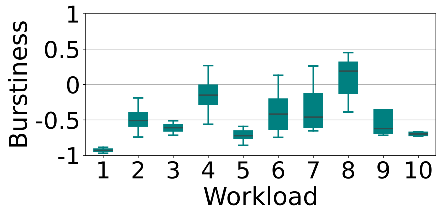


(b) Weekly granularity

Figure 5.6 Variability analysis of daily and weekly granularities



(a) Daily granularity



(b) Weekly granularity

Figure 5.7 Burstiness analysis of daily and weekly granularities

results from various perspectives: analyzing centroids, studying associations between daily and weekly patterns, and exploring the time dependence of workload patterns.

Centroid Analysis. A centroid is a representative point that summarizes the central tendency of the data and provides insight into its distribution or trend over time. In our analysis, we compute centroid values for each daily and weekly pattern, aiming to capture the characteristics of each cluster's temporal behavior. To effectively model these centroids, we use *Polynomial Models*, specifically *Quadratic* and *Cubic* polynomials, chosen for their ability to capture the nuanced shapes and fluctuations observed in workload centroids. The quadratic model is mathematically expressed as:

$$at^2 + bt + c = 0 \tag{5.5}$$

whereas the cubic model is expressed as:

$$at^3 + bt^2 + ct + d = 0 \tag{5.6}$$

where a, b, c, and d represent the coefficients of the models, and t denotes the time variable.

When fitting the polynomial models, we optimize the algorithm using the Levenberg-Marquardt algorithm [264]. This method continuously optimizes the model parameters to reduce the total squared differences between the model's predictions and the real data points. By doing so, this objective function guarantees that the adapted models precisely choose the temporal dynamics observed within the clusters.

Association between daily and weekly patterns. We determine the associations between daily and weekly patterns by calculating the presence of daily patterns within each weekly pattern. Specifically, for each weekly pattern, we calculate the percentage of the instances associated with each daily pattern. Through quantifying the frequency of daily patterns within weekly patterns, our objective is to unveil the hierarchical arrangement of temporal dynamics. This analytical strategy facilitates the interpretation of temporal associations, hence providing detailed insights into the patterns.

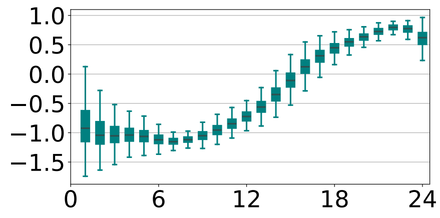
Time dependence of workload patterns. We explore the association between workload patterns and time. For daily patterns, we consider their association with days of the week (i.e., weekdays or weekends), and for weekly patterns, we consider their association with seasons of the year. Specifically, we calculate the percentage of each workload pattern's presence within weekdays/weekends or each season.

5.3.2.3 Results

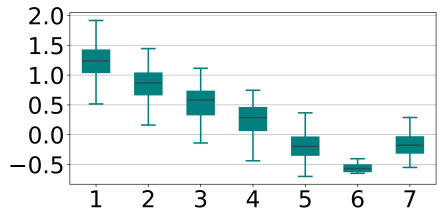
We first present the variability and burstiness analysis. Then, we provide the clustering findings in four parts: clustering patterns, centroid analysis, association between daily and weekly patterns, and time dependence of workload patterns.

Variability and burstiness analysis. Figure 5.6 illustrates the variability analysis for daily and weekly granularities across each workload. The number of workloads depicted in this figure (from 1 to 12) corresponds to the "ID" column listed in Table 5.1. A CV close to 0 indicates low variability, suggesting stable workloads, while a CV greater than 1 suggests high relative variability. Figure 5.6 reveals two key observations: 1. Daily variability is significantly higher than weekly variability, and 2. Although most workloads exhibit relatively stable workloads, high variability is observed in two daily workloads (i.e., Boston and EPA) and one weekly workload (i.e., YouTube).

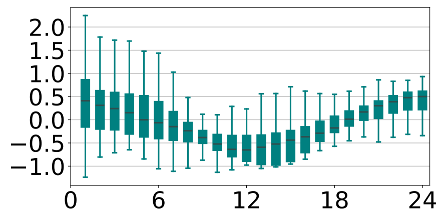
Figure 5.7 presents the burstiness analysis for daily and weekly granularities. Similar to variability findings, daily workloads exhibit higher burstiness. Specifically, while none of the weekly workloads exhibit burstiness values exceeding 0.5, two of the daily workloads, Boston and EPA, reach this threshold. The Wikipedia workload shows the lowest burstiness, nearing -1, indicating regular patterns. This regularity may result from its large global user base,



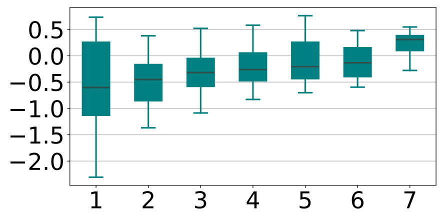
(a) D1



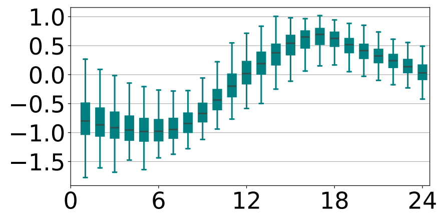
(b) W1



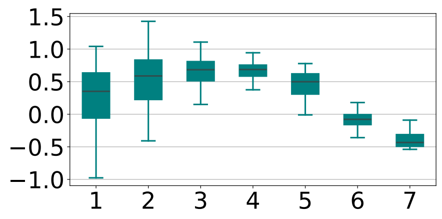
(c) D2



(d) W2



(e) D3



(f) W3

Figure 5.8 Distribution of daily and weekly clusters

which balances bursts across different regions.

There exist significant variations and burstiness within the workloads of studied web applications, which should be taken into account by practical resource provisioning and workload generation strategies.

Clustering Patterns. The clustering results are presented in Figure 5.8, showcasing the identified patterns. As mentioned in Section 5.3.2.2, we obtain 3191 datapoints at the daily granularity of web application workloads. After clustering, we identify three distinct patterns. Specifically, Figure 5.8a illustrates the first cluster (i.e., D1) with 2262 instances, Figure 5.8c shows the second cluster (i.e., D2) with 406 instances, and Figure 5.8e depicts the third cluster (i.e., D3) with 523 instances. The x-axis of these figures represents a 24-hour day.

As discussed in Section 5.3.2.2, 466 datapoints exist at the weekly granularity of web application workloads. After clustering, we uncover three patterns presented in Figures 5.8b, 5.8d, and 5.8f, and are donated as W1, W2, and W3, with 283, 64, and 119 instances, respectively. The x-axis of these figures corresponds to a 7-day week, commencing from 1 (Monday) and concluding at 7 (Sunday).

Looking at daily and weekly patterns, it is evident that they exhibit unique patterns while having commonalities. Most of the clusters (i.e., D1, D2, D3, W1, W3) are non-monotonic and have curvy shapes. On the other hand, W2 has a sublinear shape with a slightly ascending pattern. All the patterns show a distinct peak and low period, yet the rate of ascending/descending to/from these peaks differs.

Our clustering reveals three distinct patterns for both daily and weekly workloads. Interestingly, most of these patterns are non-monotonic and non-linear.

Centroid Analysis. Figure 5.9 shows the centroids of daily and weekly patterns. These centroids serve as fundamental representations of the underlying temporal dynamics captured within each cluster. Utilizing both quadratic polynomial and cubic polynomial approaches, we discover that the cubic polynomial model best fits our daily clusters and can successfully model the underlying nuances. On the other hand, for weekly patterns, due to their simpler shapes, the quadratic model is enough to fit the centroids. The coefficients of the cubic model (for daily patterns) and the quadratic model (for weekly patterns) are presented in Figure 5.9.

Complementing our mathematical modeling, we assign descriptive names to the clusters based on their centroid patterns.

Daytime active with rapid decline (D1): Substantial activity during daytime hours with a rapid decrease in activity from peak to off-peak hours.

Nighttime active (D2): Primary activity during the nighttime with gradual workload changes.

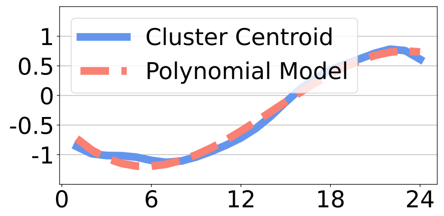
Daytime active with gradual decline (D3): Substantial activity during daytime hours with a gradual decrease in activity from peak to off-peak hours.

Weekday Active (W1): Higher activity on weekdays with consistent decrease.

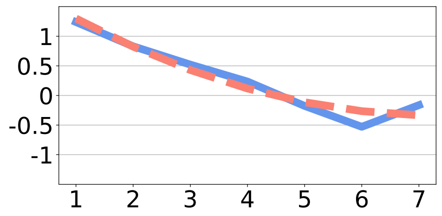
Weekend Active (W2): Increased activity on weekends with steady rise.

Midweek Active (W3): Activity concentrated in midweek, exhibiting smooth changes.

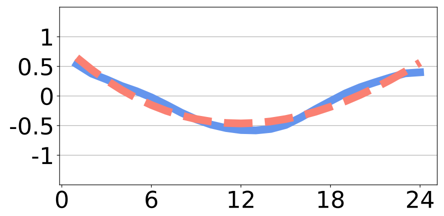
The daily and weekly patterns follow polynomial (i.e., cubic and quadratic) models. These patterns represent varying levels of user activity related to daytime/nighttime and weekdays/weekends.



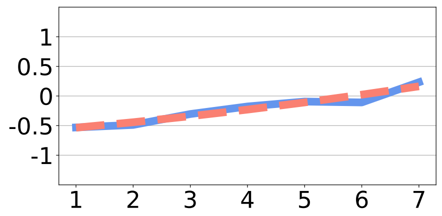
(a) D1, a = -0.001, b = 0.029, c = -0.221, d = -0.728



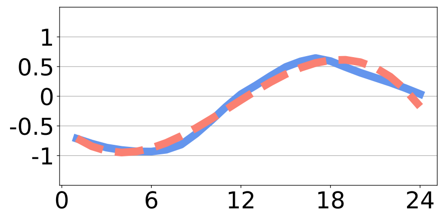
(b) W1, a = 0.041, b = -0.516, c = 1.299



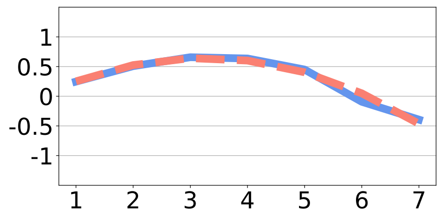
(c) D2, a = 0.000, b = 0.011, c = -0.214, d = 0.648



(d) W2, a = 0.005, b = 0.087, c = -0.535



(e) D3, a = -0.001, b = 0.031, c = -0.166, d = -0.708



(f) W3, a = -0.079, b = 0.352, c = 0.251

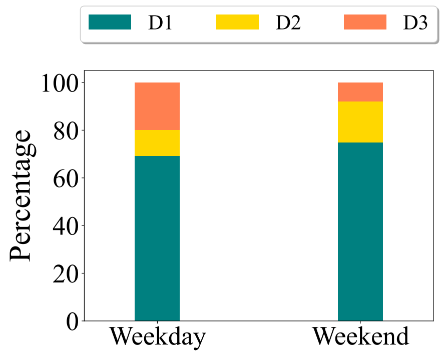
Figure 5.9 Daily and weekly centroids

Association between daily and weekly patterns. Table 5.5 shows the relative frequency of daily and weekly patterns. Notably, D1 emerges as the predominant daily pattern, while W1 stands out as the most common weekly pattern. Moreover, the combination of D1 and W1 represents the most frequent occurrence, encompassing over 43% of the workloads. In Table 5.5, we include the count of workload datasets associated with each pattern within parentheses. For instance, the combination of D1 and W1 appears in six workload datasets. This observation suggests that the identified patterns are not specific to individual datasets, rather, they exist in multiple workload datasets.

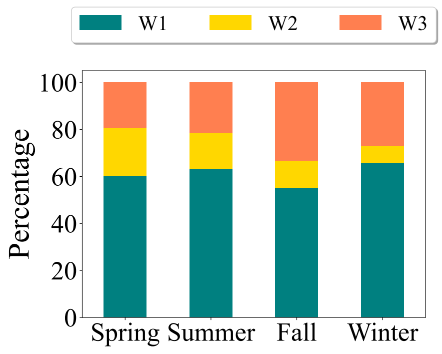
Each weekly pattern has a dominant (i.e., >50%) daily pattern; D1 for W1 and W2, D3 for W3. Besides, the identified patterns are general across different workload datasets.

Table 5.5 Relative frequency of daily and weekly clusters, expressed in percentage. The numbers in parentheses indicate the frequency of these patterns across workload datasets.

	D1	**D2**	**D3**	**Total**
W1	43.6 (6)	2.4 (5)	3.8 (7)	49.8 (9)
W2	7.3 (8)	2.7 (9)	3.2 (6)	13.2 (10)
W3	7.8 (7)	9.4 (10)	19.8 (9)	37 (12)
Total	58.7 (9)	14.5 (10)	26.8 (11)	100 (12)



(a) Daily granularity



(b) Weekly granularity

Figure 5.10 Composition of daily and weekly workload patterns regarding time

Time dependence of workload patterns. We evaluate how time affects workload patterns. Figure 5.10 illustrates each pattern's distribution across different time periods. Figure 5.10a compares patterns between weekdays and weekends for each daily cluster. Analyzing the figure reveals that while D1 follows a typical distribution, D2 and D3 diverge from this trend, with D2 showing more weekend data and D3 displaying a higher proportion of weekday data, suggesting variations in workload patterns between weekdays and weekends.

In Figure 5.10b, we examine the workload pattern distribution across weekly clusters throughout the seasons. Similar to daily patterns, there are noticeable variations in the distribution of weekly patterns across different seasons. Specifically, cluster W1 shows a higher proportion during the Summer and Winter periods, whereas cluster W3 displays more variability, with a notable increase in percentage during the Fall season.

Variations in workload patterns across different time periods, such as weekdays versus weekends, underscore the importance of considering temporal dynamics for realistic workload generation approaches.

5.4 Discussion

Random, steady, or linearly increased/decreased workloads should be replaced by polynomially evolving workloads in realistic workload generation. For synthetic workload generation, researchers commonly use steady (i.e., constant level of activity) or linear (i.e., linear step-wise increase/decrease of the level of activity to model the light/normal/peak usage) patterns [252]. These patterns have been extensively adopted in existing work (e.g., [205, 248]). Some studies also use random workloads (e.g., [265, 266]). However, our RQ2 findings reveal that real-world web application workloads exhibit non-monotonic and non-linear (polynomial) behavior. Thus, we recommend that future studies consider polynomial workload patterns instead of random, linear, or steady ones. The polynomial models that we derived in this work can be directly used to guide the design of realistic workloads.

Resource provisioning strategies can leverage the identified daily and weekly workload patterns to achieve simpler and more robust resource allocations. Existing proactive resource provisioning strategies usually rely on predictive models to make provisioning decisions (e.g., [185,190]). However, such predictive models often suffer from the challenges of interpretability and short prediction windows [267]. Practitioners can leverage the characterized workload patterns in our work to enhance the simplicity and robustness of their resource provisioning strategies. For example, they can progressively increase or decrease their provisioned resources based on a polynomial pattern (the parameters of the polynomial can be periodically learned from their workload data).

We call for the sharing of newer web application workload datasets. Our findings indicate a tendency to use outdated workload datasets in the literature. This reliance may fail to reflect the current dynamics of real-world scenarios due to significant changes in web technologies and the exponential growth of data in recent years. We encourage researchers to use more up-to-date workload datasets such as Wikipedia, which better capture the behaviors of modern users. Additionally, we urge researchers and practitioners to extract and share more recent web application workloads publicly.

5.5 Threats to Validity

External validity. We conduct a review of web application workloads, analyzing 78 articles that leverage them. To mitigate the risk of missing some studies, we conduct a systematic approach including keyword search, forward and backward snowballing, and manual analysis. However, there's a possibility of missing some studies, such as studies based on private datasets or non-English publications.

While this study characterizes patterns in web application workloads to support realistic workload generation, simulating true real-world workloads remains very challenging. User behavior, traffic bursts, and heterogeneous request types cannot be fully reproduced, and any synthetic generation approach necessarily involves simplified assumptions. Our characterization highlights representative patterns, but should not be interpreted as a complete replication of all real-world workload dynamics.

Internal validity. Internal validity may be threatened by the assumption that daily and weekly patterns adequately capture the temporal dynamics of web application workloads. It is possible that different temporal sequences, beyond daily and weekly, could reveal alternative patterns that were not considered. We deliberately focus on daily and weekly patterns as they are widely recognized temporal units in the context of web applications. Our analysis remains open to future exploration of other temporal granularities.

Construct validity. We use the K-Means algorithm for clustering workload patterns, known for its simplicity and effectiveness, with widespread usage in previous studies (e.g., [185, 268]). However, the choice of clustering algorithm can impact results. We encourage future research to explore alternative clustering algorithms. Moreover, the selection of the number of clusters may affect the quality and insights of the outcomes. To minimize bias, we employ a combination of quantitative metrics (i.e., silhouette score) and qualitative techniques (i.e., visualization) to determine the optimal number of clusters.

5.6 Chapter Summary

In this Chapter, we perform a systematic literature review to understand the usage and characteristics of web application workloads. Using a systematic approach, we identify 78 articles leveraging web application workloads and the 12 public workload datasets they use. While we observe that a wide spectrum of studies repetitively leverages these workload data for resource management, workload analysis, self-adaption, and benchmarking, we also notice a significant reliance on dated datasets. Through the characterization of the 12 identified workload datasets at daily and weekly granularities, we uncover three daily and three weekly patterns. Using statistical modeling, we find that these patterns display polynomial (non-monotonic and non-linear) behaviors. Since these patterns and insights were derived from real-world datasets, they can be used in realistic workload generation by replicating the behavioral trends observed in actual operational data. Future work can leverage these findings to develop realistic workload generation pipelines.

CHAPTER 6 ARTICLE 3: PROTECTING PRIVACY IN SOFTWARE LOGS: WHAT SHOULD BE ANONYMIZED? Roozbeh Aghili, Heng Li, Foutse Khomh, Accepted in FSE 2025, Publication Date: 2025/06/19

6.1 Chapter Overview

Previous Chapter discussed the characteristics of web application workloads and generating synthetic data. In this Chapter, we discuss acquiring real-world operational data from IT organizations. As discussed in Chapter 4, software logs are the most common data type in AIOps pipelines. Therefore, in this Chapter, we focus on protecting privacy in software logs. Software logs, generated during the runtime of software systems, are essential for various development and analysis activities, such as anomaly detection and failure diagnosis. However, the presence of sensitive information in these logs poses significant privacy concerns, particularly regarding PII and quasi-identifiers that could lead to re-identification risks. While general data privacy has been extensively studied, the specific domain of privacy in software logs remains underexplored, with inconsistent definitions of sensitivity and a lack of standardized guidelines for anonymization. To mitigate this gap, this study offers a comprehensive analysis of privacy in software logs from multiple perspectives. We start by performing an analysis of 25 publicly available log datasets to identify potentially sensitive attributes. Based on the result of this step, we focus on three perspectives: privacy regulations, research literature, and industry practices. We first analyze key data privacy regulations, such as the GDPR and the CCPA, to understand the legal requirements concerning sensitive information in logs. Second, we conduct a systematic literature review to identify common privacy attributes and practices in log anonymization, revealing gaps in existing approaches. Finally, we survey 45 industry professionals to capture practical insights on log anonymization practices. Our findings shed light on various perspectives of log privacy and reveal industry challenges, such as technical and efficiency issues while highlighting the need for standardized guidelines. By combining insights from regulatory, academic, and industry perspectives, our study aims to provide a clearer framework for identifying and protecting sensitive information in software logs.

6.2 Study Design

Software logs are record statements used by software developers to capture valuable runtime information about software systems. They are generated by logging statements inserted into the source code, which produce execution logs during runtime [269, 270]. These logs are then either appended to log files for later analysis or output to the standard console for immediate monitoring. Logging is crucial for tasks such as failure diagnosis and anomaly detection [271–273]. For instance, logs are often the only resource available for diagnosing field failures [274]. By recording different events and activities within a software system, logs help developers and other software practitioners manage and maintain the health and performance of their systems effectively.

As the number of log statements in software systems increases, the likelihood of capturing sensitive information within logs also rises, making privacy preservation more critical [275]. Protecting privacy is essential not only to comply with regulatory requirements but also to maintain user trust and prevent potential harm that could arise from data leaks. A significant concern in software log management and sharing is the exposure of PII. PII refers to any data that can be used to identify an individual, either directly, such as names and email addresses, or indirectly, such as device identifiers. While PII is often the focus of privacy concerns, it is not the only source of re-identification of individuals. A study by Sweeney et al. [44] demonstrated that using just three attributes—postal code, date of birth, and sex—over 86% of U.S. citizens could be uniquely identified. These attributes, known as *quasi-identifiers*, may not reveal any information when used individually, but when combined, they can effectively re-identify individuals.

While data privacy is a well-defined area with several decades of research, the specific domain of privacy in software logs remains largely underexplored. Despite extensive work in general data privacy, there is a significant gap in our understanding of what constitutes sensitive information within software logs. We lack clear agreement on which log attributes should be considered sensitive and which are harmless. Moreover, there is little knowledge about which combinations of log attributes might lead to the re-identification of individuals or entities. Existing studies on log privacy and anonymization [276–278] often develop their own definitions of sensitivity, leading to inconsistent approaches and a fragmented understanding of the risks involved. This inconsistency highlights the need for a systematic exploration of log sensitivity to establish a clearer framework for protecting privacy in this context.

A significant challenge in the field of software log privacy is the reluctance of organizations to share their datasets, including software logs and traces. Privacy concerns largely drive this hesitation, as the risk of exposing sensitive information can be substantial. As a result, there is a notable lack of publicly available datasets for log analysis tasks such as anomaly detection [18]. Besides, there have been incidents upon publishing anonymized data where the public datasets led to the identification of users. For example, in 2006, AOL released a dataset containing 20 million keyword searches made by its users. Although the data was anonymized, the presence of PII led to identifying several individuals by analyzing the unique search queries associated with them [279, 280]. Similarly, the Netflix Prize dataset, which contained anonymous movie ratings of 500,000 subscribers, was found to be vulnerable to de-anonymization. Narayanan et al. [281] demonstrated that by cross-referencing the Netflix data with information from the *Internet Movie Database (IMDb)*, they could identify individual subscribers, revealing their political preferences and other potentially sensitive information. In the context of software logs, similar risks exist as logs often contain detailed information about system events, user activities, and configuration settings. If such logs are anonymized inadequately, there is a risk of re-identifying individuals or revealing sensitive information, necessitating effective privacy-preserving methods in log management.

To address privacy challenges in software logs and better understand sensitive information, we conduct a comprehensive study. We examine the topic from multiple perspectives. First, we analyze 25 publicly available log datasets to identify common attributes in software logs. We then review key privacy regulations, such as GDPR and CCPA, to understand legal requirements related to data privacy, framing privacy expectations and risks. Next, we review existing research and tools on log anonymization to identify common privacy attributes and gaps in the field. Finally, we survey industry professionals to gather insights on privacy concerns in software logs. By combining findings from log analysis, regulatory analysis, literature review, and industry perspectives, our study aims to provide a well-rounded understanding of which attributes in software logs should be anonymized to protect privacy effectively. We specifically aim to answer the following RQs.

RQ1 ***What are the most common attributes in software logs?*** In this first RQ, we explore the information collected in different software logs. By identifying the most common log attributes, we can better assess the potential sensitivity of the data being logged and ensure compliance with privacy regulations.

RQ2 ***What do privacy regulations define as sensitive information in software logs?*** In this RQ, we inspect the regulations related to data privacy. By analyzing these regulations, we aim to clarify the legal obligations that organizations must adhere to when handling log data, ensuring that privacy is maintained.

RQ3 ***What do existing research and tools define as sensitive attributes in software logs?*** Existing research and tools for log anonymization provide valuable insights into what the academic community considers sensitive in software logs. By exploring these definitions, we uncover common practices and gaps that need to be addressed to enhance log privacy.

RQ4 ***What do industry professionals consider sensitive in software logs?*** Industry professionals often deal with the practical challenges of anonymizing software logs in real-world scenarios. By surveying these experts, we aim to capture the attributes they prioritize as sensitive and understand how they approach log privacy, providing a practical perspective that complements the regulatory and academic views.

6.3 Study Approach and Results

This section presents the details of our RQs and their results. We start by analyzing public software logs to understand the most common attributes of them. Then, we analyze privacy regulations, perform a systematic literature review on academic papers, and perform a survey of industry participants, to understand the privacy aspects of software logs from different perspectives.

6.3.1 Common Attributes in Software Logs

Software logs can have diverse forms and structures, varying widely in attributes, format, and content. Logs may range from simple text files recording basic events to complex formats capturing detailed system behaviors, user interactions, or security-related incidents. This variability reflects the different purposes and contexts of log generation, making it challenging to standardize or categorize them. Understanding what constitutes sensitive information within these logs becomes a complex issue, which is why in this section, we analyze a selection of publicly available logs and identify common attributes that could potentially be considered sensitive.

6.3.1.1 Approach

In our analysis, we examine 25 log datasets drawn from two well-known collections: LogHub [101] and ITA [26]. These datasets provide a wide range of logs from different domains and system types, enabling us to investigate various attributes and formats. Specifically, we analyze all datasets from the LogHub collection, which includes logs from *distributed systems* (HDFS, Hadoop, Spark, Zookeeper, OpenStack), *supercomputers* (BGL, HPC, Thunderbird), *operating systems* (Windows, Linux, Mac), *mobile systems* (Android, HealthApp), *server applications* (Apache, OpenSSH), and *standalone software* (Proxifier). Additionally, from the ITA collection, we focus on *web application* logs, including datasets from Calgary, Saskatchewan, Boston, WorldCup98, NASA, ClarkNet, EPA, and SDSC. We also incorporate the YouTube dataset collected by Zink et al. [244]. For detailed information about each dataset, please refer to the references.

To identify common attributes across different log datasets, we start by parsing each of the 25 log datasets using Drain [282], a well-known log parsing tool. Specifically, we analyze a sample of 2,000 log lines per dataset to extract log templates. We identify up to 350 unique log templates per dataset and then examine a subset of example log lines for each template to determine potential attributes. These attributes correspond to variables in the log structure, which Drain marks with an asterisk (*). To assess potential sensitive attributes, we take an inclusive approach, collecting any attribute that could pose a vulnerability. We later refine this by considering different perspectives on sensitivity (see Sections 6.3.2, 6.3.3, and 6.3.4) and ultimately defining our own criteria (see Section 6.4).

Table 6.1 presents four sample logs from ClarkNet, OpenSSH, Hadoop, and HPC datasets. As can be seen, each log follows a distinct pattern, provides specific information, and includes particular attributes. The first log is a typical HTTP access log entry showing a request made to a web server. This log has attributes such as IP address, request method, and status code. The second log indicates a failed authentication attempt via SSH and contains details such as date and time, host information, and process ID. The third log shows the maximum resource capabilities for containers in Hadoop MapReduce, with attributes such as log level and configuration details. Finally, the fourth log sample reports the temperature of a node, providing attributes such as node ID and environmental data.

6.3.1.2 Results

Through analysis of 25 log datasets, **we identify 18 frequently shared attributes in software logs**. Detailed in Table 6.2, the table includes definitions, examples, and frequency, which represents the percentage of log datasets containing each attribute. **The most common attributes are timestamps (100%), IP addresses (80%), file paths (72%), IDs (72%), and components (60%)**.

An IP address is often the first attribute identified as sensitive in software logs by the literature [278, 283, 284]. For example,the IP address *'129.188.154.200'* can reveal that it belongs to Motorola Inc. and originates from Schaumburg, Illinois, USA. While an IP address may not be traced directly to an individual, it can often be linked to a household or company, which is typically considered sensitive. Similarly, a host name can provide insights into the user's location. For instance, the host name *'ec2-52-80-34-196.cn-north-1.compute.amazonaws.com.cn'* indicates that the host machine is utilizing Amazon services,

Table 6.1 Example software logs from different datasets.

1. `132.201.126.207 - - [28/Aug/1995:17:23:48 -0400] "GET /pub/atomicbk/images/spicy2.gif HTTP/1.0" 200 29018`
2. `Dec 10 07:07:38 LabSZ sshd[24206]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=ec2-52-80-34-196.cn-north-1.compute.amazonaws.com.cn`
3. `2015-10-18 18:01:53,713 INFO [main] org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator: maxContainerCapability: <memory:8192, vCores:32>`
4. `76723 node-55 node temperature 1077205904 ambient=33`

Table 6.2 The most frequent log attributes and examples

ID	Attribute	Definition	Example	Freq. (%)
1	Timestamp	Date and time of the log entry.	2024-08-15 - 12:11:37	100
2	IP address	Unique number for network devices.	192.168.1.1	80
3	File path	Location of a file in the filesystem.	/user/root/rand/_temporary/part-00742	72
4	IDs	Identifiers for system entities.	Process ID, Thread ID, Job ID, Node ID, Application ID, Device ID	72
5	Component	Module of the system generating the log.	org.apache.hadoop.mapreduce.v2.app.MRAppMaster	60
6	Host name	Unique name for network devices.	ec2-52-80-34-196.cn-north-1.compute.amazonaws.com.cn	44
7	Log level	Severity of the log event.	INFO	40
8	Port number	Number identifying a specific service.	8080	36
9	Request protocol	Protocol used for the request.	HTTP/1.0	36
10	Request status code	HTTP status code returned by the server.	200	36
11	Request response size	Size of the server's response.	56 B	36
12	Configuration details	System configuration information.	vCores:32	36
13	Request method	Method used to request a resource.	GET	32
14	URL	Address of resources on the internet.	http://cs-www.bu.edu/lib/pics/bu-logo.gif	24
15	MAC address	Unique identifier for network interfaces.	5c:50:15:4c:18:13	8
16	Request response time	Time taken for server response.	0.3 s	8
17	Environmental data	Data related to environmental conditions.	temperature ambient=33	8
18	Username	Unique user identifier.	cheng	8

with the *'cn-north-1'* label suggesting that it is located in Beijing, China. In another example, the host name *'proxy.cse.cuhk.edu.hk'* reveals that it belongs to the *Computer Science and Engineering (CSE)* department at the *Chinese University of Hong Kong (CUHK).*

One major category of potentially sensitive data that has been less emphasized in studies is the configuration details (see Section 6.3.3). We define configuration details as all information related to system settings and configurations. For instance, in the Thunderbird dataset, we encounter information such as the CPU specification *(Intel(R) Xeon(TM) CPU 3.60GHz stepping 03)*, the number of CPU cores *(checking TSC synchronization across 4 CPUs: passed)*, memory capacity *(L2 cache: 2048K)*, and network connection status *(NIC Link is Up 1000 Mbps Full Duplex)*. Similarly, the Linux dataset includes details such as CPU specification *(CPU: Intel Pentium III)*, Linux version *(Linux version 2.6.5-1.358)*, and gcc version *(gcc version 3.3.3 20040412)*. Another example from *"Jul 27 14:41:59 combo kernel: PCI: Using IRQ router PIIX/ICH [8086/2410] at 0000:00:1f.0"* reveals that the chipset is manufactured by Intel based on *PIIX/ICH* and vendor ID *8086*. Additionally, we find configuration details such as the number of nodes, sleep/timeout durations, CPU fan speed, and installed package names in Android device logs. We believe that this information could be sensitive to the log provider and pose a risk of unintentional data leakage, for example, exposing detailed information about a company's computing environment that may be vulnerable to adversarial attacks. Configuration details are found in 36% of log datasets.

Most of the log datasets examined are raw and unaltered. According to the description of the LogHub collection, "Wherever possible, the logs are NOT sanitized, anonymized, or modified in any way". Further confirmation from the LogHub collection publishers verified that most datasets remain intact, with the exceptions of Android, HDFS, and Windows datasets, which were edited to remove certain private IP addresses and package names. For the ITA collection, the Calgary dataset has had IP addresses entirely removed for privacy reasons, file paths shortened, and file names changed. In the WorldCup98 and SDSC datasets, IP addresses have been renumbered and converted to integer values. In the Boston dataset, User IDs have been transformed using a one-way function.

Summary

Timestamps, IP addresses, file paths, IDs, and components are the most common attributes in logs. We also identify attributes such as configuration details and environmental data.

6.3.2 Privacy Regulations

Privacy regulations are designed to protect personal data and ensure that organizations handle this data responsibly. Different countries and continents have their own sets of privacy regulations, with some countries having multiple regulations to address various aspects of data protection. For example, the United States has HIPAA for health data privacy and CCPA for consumer data protection. In this study, we focus on some of the most widely recognized and adopted regulations, including the GDPR in Europe, HIPAA, and CCPA in the United States, PIPEDA in Canada, and ISO standard 27001, which provides an international framework for information security management. By analyzing these key regulations, we aim to cover a broad range of privacy practices and requirements.

6.3.2.1 GDPR

GDPR [39] is a data regulation on information privacy established by the *European Union (EU)* that applies to the *European Economic Area (EEA)*. Established in 2018, GDPR is widely recognized as one of the most comprehensive and influential data privacy regulations globally. It sets strict guidelines for the collection, processing, storage, and transfer of personal data, aiming to protect the privacy and rights of individuals within the EU and EEA.

At the core of GDPR is the broad definition of personal data, which encompasses **any information related to an identified or identifiable natural person**. This definition is wide-ranging, covering both direct identifiers, such as names and identification numbers, and indirect identifiers, such as location data and IP addresses. The regulation emphasizes that IP addresses can be considered personal data if they can be linked to an individual, either directly or indirectly. For instance, if a data controller has the means to obtain additional information that could identify the person behind an IP address, that IP address would be classified as personal data under GDPR. Moreover, GDPR acknowledges that identifiers from devices, applications, and protocols can build detailed profiles of individuals. These identifiers, such as IP addresses and cookie IDs, can leave digital footprints that, when combined with other data, might disclose an individual's identity.

In the famous case of "Breyer v. Germany" [285], the Court of Justice of the European Union examined whether a dynamic IP address qualifies as personal data. The court ruled that while a dynamic IP address on its own might not identify an individual, it can be considered personal data if it can be combined with additional information that allows for identification. This is particularly relevant when an online service provider has the legal ability to access such identifying information.

6.3.2.2 HIPAA

HIPAA [40] is a U.S. legislation aimed at protecting the privacy and security of individuals' health information. Established in 1996, HIPAA creates national standards for data protection, particularly focusing on the privacy of health information. This regulation is crucial in the healthcare industry, ensuring that sensitive patient information is adequately protected while allowing for the necessary flow of data to provide high-quality healthcare services. Unlike GDPR, which broadly defines personal data, HIPAA specifies 18 sensitive identifiers that must be protected. These identifiers range from direct identifiers such as names, phone numbers, and Social Security numbers, to indirect identifiers such as geographic subdivisions, dates related to an individual, and IP addresses. Some of these 18 identifiers could be found in software logs, **such as dates, IP addresses, device identifiers, URLs, and any other unique identifying number, characteristic, or code.**

6.3.2.3 CCPA

CCPA [41] is a state statute intended to strengthen privacy rights and consumer protection for residents of California, United States. Signed into law in 2018, CCPA provides California residents with significant control over their personal information, granting them the right to know what data is being collected about them, the right to request the deletion of their data, and the right to opt out of the sale of their personal information. Under the CCPA, personal information is broadly defined as **any data that identifies, relates to, or could reasonably be linked with an individual or their household**. This includes obvious identifiers such as names, social security numbers, and email addresses, but it also extends to less direct identifiers such as internet browsing history and geolocation data. Importantly, CCPA distinguishes between "personal information" and "sensitive personal information", which includes highly specific data such as government identifiers, precise geolocation, email contents, genetic data, and biometric identifiers. The CCPA recognizes that not all personal information requires the same level of protection, including exemptions for publicly available government records and data already governed by other laws such as HIPAA.

6.3.2.4 PIPEDA

PIPEDA [134] is a Canadian law that regulates how private sector organizations handle personal information during commercial activities. Established in 2000, PIPEDA mandates that organizations collect, use, and disclose personal information responsibly, ensuring the privacy of individuals. Personal information under PIPEDA includes **any factual or subjective information, recorded or not, about an identifiable individual**, such as age, name, income, ID numbers, or opinions. This broad definition encompasses various forms of data, making PIPEDA a comprehensive privacy law. However, PIPEDA does not apply universally. Exceptions include personal information managed by federal government organizations under the Privacy Act, provincial and territorial governments, and business contact information used solely for professional communications. Additionally, PIPEDA does not cover information collected for personal, journalistic, artistic, or literary purposes.

6.3.2.5 ISO27001

ISO 27001 [286] is an internationally recognized standard for *Information Security Management Systems (ISMS)*, providing **a framework for organizations to systematically manage sensitive data.** Unlike previously discussed regulations, ISO 27001 does not define personal or sensitive data but focuses on managing information security risks, allowing organizations to determine sensitivity based on context and requirements. Initially published in 2005, ISO 27001 guides the implementation and maintenance of an ISMS through a combination of technological, organizational, physical, and human controls, offering flexibility for different industries and data types.

Summary

While many data privacy regulations exist, most are broad in scope and not specifically designed for software logs. Therefore, it is essential to extract and interpret relevant information from these regulations that could be applicable to logs. Some regulations explicitly define personal data and specify attributes that need protection. For instance, GDPR and HIPAA both classify IP addresses as sensitive data. In contrast, ISO 27001 does not define personal or sensitive data but instead offers a flexible framework for managing any data that an organization identifies as sensitive. Overall, systematically identifying sensitive information in logs and designing robust anonymization strategies remains an open challenge and requires further investigation beyond the regulatory frameworks themselves.

6.3.3 Articles and Tools

In this section, we conduct an SLR of existing research articles and tools related to log privacy to deepen our understanding of privacy in software logs. The main goal of this review is to determine which software log attributes are regarded as sensitive by researchers.

6.3.3.1 Approach

We conduct an extensive search across two research databases, initially identifying 88 potential candidate articles. Applying inclusion and exclusion criteria, we reduce the selection to 22 papers. To ensure a thorough review, we perform forward and backward snowballing, leading to the inclusion of an additional 36 articles. Combining these, we analyze a total of 58 papers. Our selection process follows the systematic approach recommended by Wohlin et al. [207], as detailed below.

Searching Research Databases. We conduct our literature search using two well-known databases: the IEEE Xplore Digital Library [208] and the ACM Digital Library [209]. Our objective is to find papers from both conference proceedings and journals. To identify articles that focus on defining privacy in software logs or perform log anonymization, we employ the following search query:

Title: (log AND privacy) OR (log AND sensitiv) OR (log AND anonymiz*)*

Using this query, we find 57 relevant articles from the IEEE library. We also obtain an additional 31 articles from the ACM library after deduplication. In total, combining results from both databases, we identify 88 candidate articles for further review.

Article Selection. We start by reviewing each paper's abstract and continue through the entire article, applying the inclusion and exclusion criteria until a final decision is reached.

Inclusion criteria:

- Papers must be written in English.
- Papers must be in conference proceedings or journals.
- Papers must be in the scope of software logs.
- Papers must either define privacy in software logs, introduce an anonymization tool, explore privacy considerations, or improve privacy measures in software logs.

Exclusion criteria:

- Papers not publicly available.
- Papers that focus on general data privacy without specific emphasis on software logs.
- Papers working with other types of data such as search logs.

After performing the article selection step, we end up selecting 22 out of the 88 candidate articles.

Forward and Backward Snowballing. To ensure thoroughness in our study, we conduct both forward and backward snowballing on the initially selected 22 papers. In this phase, we examine the papers that have cited these selected articles (i.e., forward snowballing) as well as the references cited within them (i.e., backward snowballing). For each of these papers, we apply our article selection criteria (detailed in Section 6.3.3.1) to determine if they meet our inclusion and exclusion criteria. This process results in the identification of 36 new articles. By combining these with the articles initially found through our database search, we compile a total of 58 papers that specifically address privacy in software logs.

Information Extraction. In the final step, we analyze how each paper defines privacy in the context of log data. We review their approach sections to determine which attributes are considered sensitive.

6.3.3.2 Results

IP addresses, timestamps, and port numbers are most frequently considered as sensitive attributes in logs. Table 6.3 summarizes the attributes most frequently identified as sensitive across the reviewed articles. Among these, IP address stands out as the most discussed attribute, appearing in 59% of the analyzed articles. This is followed by the timestamp, which is considered sensitive in 28% of the studies, and the port number, which is mentioned in 21% of the papers.

Comparing Table 6.2 with Table 6.3, we find a notable overlap between the attributes found to frequently occur in software logs and those highlighted in the existing literature. Our log analysis indicates that every software log includes the timestamp attribute, with 80% also having the IP address. This aligns with the literature, where IP address and timestamp are identified as the most frequently studied attributes. We also find that many attributes, such as port number, various IDs, usernames, and file paths, are common in both tables.

We identify three new attributes from the reviewed articles that were not present in the analyzed logs: network-related features, email addresses, and location data. Network-related attributes are features that are specific to network logs, such as the "don't fragment" bit, *Transmission Control Protocol (TCP)* window size, and TTL. The "don't fragment" bit is a flag in the IP header that determines if a packet should be fragmented. TCP window size specifies the amount of data the receiver can handle at once. TTL is a field in the IP header that limits the number of hops (i.e., routers or devices) a packet can make before being discarded. The other newly detected attributes are email addresses and location information such as address or zip code.

On the other hand, several attributes that are commonly found in software logs have not been thoroughly discussed in the literature. Some of these attributes are not mentioned in any of the reviewed articles. For instance, attributes such as component, log level, request status code, and environmental data fall into this category. These attributes can provide insights into the internal workings of a system, such as specific components involved in processing or the severity of events. Additionally, some attributes are mentioned in articles but at a relatively low frequency. For example, although file paths are present in 72% of the logs examined, they are discussed in only 7% of the articles. Similarly, while 44% of logs contain host names, only 5% of studied articles discuss this.

Attributes that have been discussed in only one article are categorized under "others." This group includes computer commands, passwords, URLs, ages, and credit cards.

Many studies only focus on the privacy of IP addresses. 14 of the reviewed articles (almost 24%) focus exclusively on IP addresses as the sensitive attribute (e.g., [278, 284, 287, 288]). One of the earliest and most influential works in software log privacy by Xu et al. [278], introduces CryptoPAn, a tool for prefix-preserving IP address anonymization. This means that if two original IP addresses share a k-bit prefix, their anonymized versions will also share the same k-bit prefix. CryptoPAn employs a cryptography-based approach, using bitwise anonymization, where the anonymized IP addresses are determined by previous anonymization. However, CryptoPAn is known to be vulnerable to fingerprinting and injection attacks [42, 43], where attackers use known network flows or inject fake flows to infer other flows by analyzing unchanged fields such as timestamps.

In a more recent development, Han et al. [287] introduce AFT-Anon, a method designed for real-time anonymization of IP addresses in network packets. This approach leverages flow tables to ensure efficient and secure anonymization as data flows through the network. In another study, Manocchio et al. [288] explore the use of a non-reversible pseudonymization function to anonymize IP addresses. They compare the accuracy of two machine learning models on a network dataset both before and after anonymization. Their findings reveal a decrease in accuracy, ranging from 1% to 4%, post-anonymization, highlighting the trade-off between privacy and data utility.

In a different approach, Mohammady et al. [284] propose the Multi-view method, generating multiple anonymized versions of network traces that are so similar adversaries cannot distinguish between them, preserving privacy. Simultaneously, one version provides accurate analysis results for the data owner, ensuring utility. However, the study recommends creating 160 distinct data views for the analyst, which may not be practical due to time and computational constraints.

Many studies focus only on the network-related attributes. As discussed in Section 6.3.1, software logs can originate from various sources, including distributed systems, supercomputers, operating systems, and server applications. Despite this broad spectrum of software logs, many selected articles, and particularly most anonymization tools, focus exclusively on network logs (e.g., [136, 283]).

As highlighted by Slagell et al. [289], there are 16 types of network logs that require anonymization, including TCPdump, NetFlow, and Syslog. Several well-known log anonymization tools are designed to handle these network data types. For instance, CANINE, developed by Li et al. [283], is a tool specifically for anonymizing NetFlow data. CANINE can anonymize five

Table 6.3 Usage of sensitive log attributes in reviewed articles

Attribute	**Freq. (%)**	**Attribute**	**Freq. (%)**	**Attribute**	**Freq. (%)**
IP address	59	Username	14	Email	7
Timestamp	28	Request response size	10	File path	7
Port number	21	Configuration details	9	Host name	5
IDs	17	MAC address	9	Location	3
Network-related	16	Request protocol	9	Others	9

different attributes, including IP addresses, timestamps, port numbers, request protocols, and response sizes. It employs different anonymization techniques for each attribute; for instance, IP addresses can be anonymized using truncation, random permutation, or prefix-preserving pseudonymization.

FLAIM is widely regarded as one of the most comprehensive log anonymization tools available. Introduced by Slagell et al. [136], FLAIM employs a range of anonymization techniques, including black marker, truncation, and hashing, to anonymize netfilter logs, pcap traces, and NetFlow data, covering attributes such as IP addresses, timestamps, *Media Access Control (MAC)* addresses, host names, and port numbers. Additionally, it can handle specific network-related attributes such as *Internet Control Message Protocol (ICMP)* codes and types, the "don't fragment" bit, TCP window size, and TCP options.

AnonTool is a specialized anonymization tool designed for NetFlow data [290,291]. Developed in C, it can anonymize live packet traces in the libpcap file format, which is used for network traffic capture. AnonTool handles various attributes, including IP addresses, timestamps, port numbers, response sizes, and specific NetFlow attributes such as *Type of Service (TOS)* and TCP flags.

Created by Minshall, TCPdpriv [292] is another anonymization tool that works with the libpcap library. However, it has some limitations, including compatibility with only SunOS, Solaris, and FreeBSD systems. Based on TCPdpriv, Plonka developed IP2anonIP [293] to convert IP addresses into host names or anonymous IP addresses. IP2anonIP also offers the option to add custom fields, but preparing a dataset for a single day can take up to several hours.

Some studies discuss privacy without specifying any attribute. 13 articles (22%) do not identify any log attributes as sensitive. We categorize these articles into two main groups: (1) surveys and taxonomies, and (2) log parsers.

1) Surveys and taxonomies. Burkhart et al. [294] explore the vulnerabilities of network trace anonymization techniques, particularly focusing on the threat posed by traffic injection attacks. The authors demonstrate the ease of executing these attacks and discuss why some anonymization techniques such as aggregation, randomization, or field generalization might be ineffective. They argue that anonymization alone is insufficient for data protection and must be integrated with legal, social, and technical measures to ensure effective network trace sharing and data protection.

King et al. [295] present a taxonomy and adversarial model for classifying attacks against network log anonymization. By categorizing various attacks, they provide a framework for identifying the strengths and weaknesses of different anonymization techniques, including pseudonymization, IP address prefix preserving, black marker, and random permutation.

Silva et al. [296] provide a survey focused on privacy within cloud computing. The paper reviews the current state of privacy in cloud services, addressing various threats, concepts, and technologies. The authors examine *Privacy Enhancing Technologies (PETs)* and different anonymization mechanisms, tools, models, and metrics, exploring their relevance to cloud environments. The paper highlights the importance of integrating various privacy mechanisms and metrics to ensure compliance with regional regulations and strengthen data protection in the cloud.

In another literature survey, Majeed et al. [297] emphasize the increasing importance of privacy protection in data sharing, particularly through *Clustering-based Anonymization Mechanisms (CAMs)*. CAMs are identified as effective methods for preserving both privacy and utility in data publishing, surpassing traditional models such as k-anonymity and differential privacy. The paper analyzes existing CAMs, categorizing them by data types and assessing their strengths and weaknesses. It also discusses the challenges CAMs encounter, such as managing heterogeneous data, ensuring resilience against AI-powered attacks, and balancing privacy with utility.

2) Log parsers. A trend appears to be emerging in designing log parsers that prioritize privacy concerns. The Delog parser [298] is designed to enhance privacy and performance in log filtering for long-running software applications. It leverages previous successful runs to automatically identify errors, applies *Locality Sensitive Hashing (LSH)* to parse log lines, and employs a privacy-preserving mechanism, using encryption and bloom filters, to securely learn new patterns from client logs. The parser is validated using four of the LogHub datasets, namely HDFS, BGL, HPC, and Zookeeper.

In another attempt, Li et al. [299] introduce TripleLP, a blockchain-based tool for privacy-focused log management. TripleLP automates the log parsing process by organizing logs into a structured format within a blockchain, facilitating efficient analysis and secure storage. It consists of three main components: the log server, log parser, and blockchain. The log parser dynamically adjusts templates and parameters, using a heuristic log tree approach to classify and analyze logs. After parsing, logs are encrypted and verified via blockchain before being stored securely. The tool is validated using four of the LogHub datasets: HDFS, Zookeeper, Thunderbird, and HealthApp.

Summary

A systematic review of 58 articles reveals that IP addresses, timestamps, and port numbers are the most frequently studied sensitive attributes in logs. Some research also highlights additional sensitive elements, such as network-related attributes (e.g., TCP window size), email addresses, and location data, though they may not appear in many types of software logs. We also find that most studies only focus on a small set of attributes, with only a few addressing multiple sensitive attributes.

6.3.4 Industry Practices

In this section, we design a survey to understand industry professionals' perspectives on software log privacy. The survey explores which attributes in software logs are considered sensitive by those involved in log management and anonymization. By gathering insights from experts, we aim to complement our findings from log analysis, regulatory review, and literature research, providing an overview of prioritized attributes and handling approaches for privacy protection.

6.3.4.1 Survey Design

In designing the survey, we use a variety of question types, including yes/no questions, multiple-choice questions, Likert-scale questions, and open-ended questions, to capture both quantitative and qualitative insights from participants. Using these different types of questions, we aim to gather diverse data that includes both statistical and descriptive insights, allowing us to analyze not just what professionals think but also how they justify their opinions and approaches.

The survey is designed to respect the privacy and time of the participants. It takes between 5-10 minutes to complete, ensuring that it is not inconvenient. We do not collect any personal information, such as names, email addresses, or the company names where participants are employed, ensuring that responses remain anonymous and protecting participants' confidentiality.

Conducting a pilot survey is beneficial in order to identify and address any unforeseen problems and challenges before releasing the final version. It is crucial for pilot studies to use the same artifacts and procedures (e.g., the invitation, explanation, and format) as those used in the main study [300]. Therefore, we conducted a pilot survey with the first three respondents. Based on their feedback, we made slight changes in the wording of some questions and answer options to improve clarity and ensure that the questions are easily understood by the target audience.

6.3.4.2 Participant Selection Strategy

According to Kitchenham et al. [301], it is crucial that the target population is capable of addressing the research questions effectively. To identify the relevant audience for software log privacy, we concentrate on professionals involved in log management, log analysis, and data privacy. We select survey participants using two methods. The first method involves searching for specific roles and keywords on LinkedIn [1], a popular professional networking platform. We use four keywords: (1) privacy engineer, (2) data privacy officer, (3) IT manager, and (4) IT security engineer. These keywords help us filter professionals likely to have direct experience with software logs and data privacy. The second method involves distributing the survey link through the authors' network to reach industry partners and professionals. Due to the broad distribution, we lack the exact count of recipients, preventing us from calculating a precise participation rate.

We conduct the survey over three weeks, providing sufficient time for participants to respond while ensuring timely data collection. This duration is chosen to strike a balance between obtaining a robust sample size and accommodating the busy schedules of professionals.

6.3.4.3 Survey Questions

Figure 6.1 shows the survey flowchart, which starts by determining if the participant has experience in analyzing, processing, or managing software logs. If the answer is "no", the participant is ineligible to answer the survey and does not proceed further; if the answer is "yes", the participant moves on to answer questions 1 and 2.

The survey then asks if participants have experience with log anonymization. If the response is "no", the participant skips directly to question 14. On the other hand, if the participant has experience in log anonymization, they proceed to question 3. This branching structure effectively creates two participant profiles: one for those with software log experience but no anonymization, and another for those experienced in both. Questions 3 to 13 focus more

[1]`https://www.linkedin.com/`

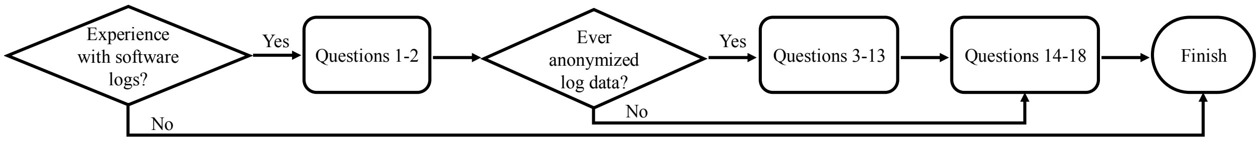


Figure 6.1 Structure of our survey

on specific aspects of log anonymization, such as balancing utility with privacy, to better understand the practices and challenges faced by professionals in the field. Below, we provide our survey questions.

Multiple-choice questions:

Q1. Are you familiar with the following data protection regulations and standards?

Q2. Which of the following attributes do you consider sensitive in software logs?

Q3. When you want to share software logs, which attributes do you anonymize?

Q4. Why does your response to Q2 differ from Q3?

Q5. What factors influence your decision to anonymize certain log attributes?

Q6. Which anonymization techniques do you primarily use for log data?

Likert-scale questions:

Q7. How much do you believe anonymization impacts the usefulness of the data?

Q8. How challenging do you find balancing anonymization with preserving the utility of log data?

Q9. How effective are your anonymization practices in protecting sensitive information?

Q10. How effective are your anonymization practices in preserving utility?

Q11. How efficient are your anonymization practices regarding computation cost and time?

Open-ended questions:

Q12. Can you share strategies or best practices to maintain data utility while anonymizing logs?

Q13. What challenges do you face when applying anonymization techniques to software logs?

Q14. Do you have any additional comments, suggestions, or insights?

Demographic questions:

Q15. Which best describes your role in the organization?

Q16. How many years of experience do you have?

Q17. In which industry does your organization operate?

Q18. What is the approximate size of your organization?

We incorporate insights from previous sections when designing the survey. For example, we use the regulations discussed in Section 6.3.2 to form the response options for Q1. Similarly, we adapt Table 6.2 (with some minor modifications) as response options for Q2 and Q3.

6.3.4.4 Participants Demographics

During the three weeks the survey was open, we collected 61 responses. Among them, 16 respondents lacked experience with software logs and were therefore ineligible to complete the survey, leaving 45 valid participants. Of these, 53% fell into the first profile, having experience with software logs but not with log anonymization, while the remaining 47% belonged to the second category, having experience with both software logs and log anonymization. Out of our 45 participants, only seven were unfamiliar with any data regulations. The remaining 84% were familiar with at least one, with all aware of the GDPR, making it the most recognized data regulation. Other regulations, such as CCPA (58%), HIPAA (53%), and PIPEDA (49%), were also well-known. Additionally, 29% of participants mentioned familiarity with regulations not covered in this study, such as Brazil's LGPD or various provincial regulations. Based on Q15 responses, we categorize participants' roles into six groups to better understand their professional backgrounds and expertise. These categories are **Data Privacy roles**, **Software Engineering roles**, **Security roles**, **Network/System roles**, **Data Science/Engineering roles**, and **Management roles**.

Data Privacy roles include professionals focused on ensuring compliance with data protection regulations and implementing privacy-preserving practices, such as data privacy officers and privacy engineers. **Software Engineering roles** consist of developers involved in designing, building, and maintaining software systems and applications. **Security roles** are experts responsible for protecting systems and data from unauthorized access, breaches, and other cyber threats, such as security engineers. **Network/System roles** include specialists in designing, managing, and optimizing IT infrastructure, such as network architects. **Data Science/Engineering roles** involve professionals who analyze data, build models, and develop data pipelines to derive insights. Lastly, **Management roles** include senior leaders and team leads who oversee technical teams, make strategic decisions, and manage organizational resources, such as a *Chief Technology Officer (CTO)*.

Table 6.4 shows the demographics of survey participants, highlighting their job roles, experience, industries, and organization sizes. Most respondents hold roles related to data privacy (40%), software engineering (24%), and security (20%). Most participants are highly experienced in their fields; 42% have over 10 years of experience, and 22% have between 7 to 10 years, indicating senior-level expertise. The majority work in the technology industry (70%), closely tied to software logs and data privacy. Additionally, 78% of the participants

Table 6.4 Demographics of survey participants

(a) Job Role

Job Role	Percentage
Data Privacy roles	40%
Software Engineering roles	24%
Security roles	20%
Network/System roles	8%
Data Science/Engineering roles	4%
Management roles	4%

(b) Experience

Experience	Percentage
Less than 1 year	5%
1-3 years	11%
4-6 years	20%
7-10 years	22%
More than 10 years	42%

(c) Industry

Industry	Percentage
Technology	70%
Finance	9%
Healthcare	4%
Manufacturing	4%
Government	4%
Other	9%

(d) Organization Size

Size	Percentage
1-100 employees	18%
101-500 employees	4%
More than 500 employees	78%

are employed in large organizations with more than 500 employees, which often have more complex data privacy and security challenges compared to smaller companies. Overall, the combination of extensive experience, professional focus on relevant fields, and employment in large organizations suggests that our survey results can provide valuable insights into industry practices and perspectives on log privacy.

6.3.4.5 Survey Results

IP addresses, MAC addresses, host names, and file paths are the most sensitive log attributes from the industry perspective. A significant majority of industry professionals, 87%, regard IP addresses as sensitive data in software logs that require anonymization, followed closely by MAC addresses at 82%. These two attributes are considered the most critical to protect, far higher than other attributes. Over 50% of the participants also identify host names and file paths as sensitive elements requiring careful handling. Other attributes considered sensitive by industry include various forms of IDs, URLs, port numbers, components, and usernames.

9% of participants propose additional factors in determining sensitivity in software logs, emphasizing the importance of contextual and legal considerations. For instance, Respondent

30 notes that "Sensitivity depends on the context of the environment, community, and secondary actions (e.g., re-identification, linking, and correlation.)" Similarly, Respondent 31 mentions that "IP may be sensitive or public, other elements such as IDs, date and time, and user actions associated with a log may be sensitive depending on the context in which it is used, how it is connected to a person or represents a person". Respondent 46 also points out that "Sensitivity could depend on the type of website visited (i.e., healthcare vs. gaming)". Additionally, some respondents highlight that sensitivity should be assessed based on relevant laws and regulations.

Log utility, policies, and technical challenges are among the main factors influencing the decision for anonymizing a log attribute. The process of deciding whether to anonymize a log attribute varies widely among professionals in the industry. Through Q4, we explore why there is an inconsistency between what respondents consider sensitive and what they actually anonymize in practice. Many participants cite various practical reasons: 52.4% point to the potential utility loss, 38.1% anonymize only as required by policies or regulations, 28.6% face technical limitations, and a smaller group, 14.3%, anonymize fewer attributes because their logs are not shared externally.

When asked what factors influence their decision to anonymize specific log attributes (Q5), most participants highlight legal compliance (81%) and the risk of re-identification (81%) as key factors. Company policies also play a significant role, with 76.2% of respondents noting it. Additionally, 55% of participants mention the impact on data utility as an influencing factor. Customer requirements are less frequently cited, with 42.9% indicating this as a consideration. These responses highlight a complex decision-making environment where legal obligations, technical feasibility, data utility, and organizational policies all play a role in shaping anonymization strategies for software logs.

Finding a balance between privacy and utility is extremely challenging. Finding a balance between privacy and utility in log anonymization is a significant challenge, as highlighted by the survey responses. A notable number of respondents recognize the impact of anonymization on data utility, with 33.3% reporting a "moderate" impact and another 38.1% indicating a "significant" impact. This indicates that while anonymization is essential for protecting sensitive information, it often comes at a cost to data usability. This trade-off emphasizes the difficulty practitioners face in balancing to protect privacy while maintaining data utility for analytical purposes.

Respondents also note the complexity of balancing privacy and utility, with 76.1% rating it as "extremely challenging", "challenging", or "moderately challenging". This reveals the need for more effective anonymization methods that can better preserve both privacy and utility.

Table 6.5 Sensitive log attributes from industry perspective

Attribute	**Freq. (%)**	**Attribute**	**Freq. (%)**	**Attribute**	**Freq. (%)**
IP address	86	Component	27	Request method	9
MAC address	82	Username	20	Request status code	9
Host name	59	Configuration details	18	Request response time	4
File path	52	Date and Time	18	Request response size	2
IDs	43	Environmental data	11	None	2
URL	39	LOG level	11	Others	9
Port number	34	Request protocol	9		

Current techniques such as differential privacy, pseudonymization, and data masking offer varied levels of success. For instance, while differential privacy and synthetic data generation can enhance privacy without completely sacrificing data utility, pseudonymization allows for conditional re-identification when necessary. These techniques reflect the need for context-specific approaches.

Despite using various techniques, only 57.1% of respondents consider their anonymization practices “effective” or “very effective” in protecting sensitive information. Similarly, only 57.2% believe their methods adequately maintain data utility. This indicates that, although there is some confidence in current anonymization strategies, there is still considerable room for improvement. The diversity in anonymization approaches highlights the need for more sophisticated and adaptive solutions capable of handling nuances of different data types and use cases. Developing frameworks that dynamically adjust to changing requirements and contexts could significantly enhance the effectiveness of anonymization practices in balancing privacy and utility.

Current anonymization practices face efficiency, technical, and operational challenges. A significant finding from the survey is that only 33.3% of the participants view their anonymization techniques as efficient in terms of computational costs and time, with the remaining 66.7% categorizing them as either “moderately efficient” or “not efficient”. None of the respondents rated their methods as “highly efficient”, highlighting a clear need for optimization in this area.

Participants also highlighted several technical challenges in anonymization, such as maintaining consistency and referential integrity, especially with high-volume, real-time logs that are often unstructured. The variety in log formats and the need for diverse data handling tools complicate the anonymization process. Besides, many respondents pointed out that manual processes and the necessity for ongoing checks to ensure anonymization effectiveness increase both the time and computational cost while also introducing the risk of human error. This emphasizes the need for more advanced, automated solutions to manage diverse data types and volumes efficiently.

Another significant operational challenge is navigating legal and compliance complexities. Respondents frequently mentioned difficulties in aligning log anonymization practices with various regulations such as GDPR and CCPA. Balancing regulatory compliance with the need to maintain data utility for purposes such as fraud detection and security monitoring adds complexity to the process. This highlights the need for a comprehensive log anonymization approach that addresses both technical and legal dimensions, ensuring anonymization practices align with regulations.

Context is important. The participants' responses also highlight that effective anonymization practices are highly context-sensitive. For instance, respondent 47 mentions: "Context is key! Who are you sharing the data with? What outside context do they have access to that could lead to identification? What is the threat/worst case scenario?" Another participant mentions "Defining PII requires interpretation of context. A log that demonstrates action associated with a person, while factual, may be PII if it demonstrates a failure to comply with legislation or policy." Besides, for some scenarios and for specific organizations, there might be the need for re-identification: "There is a need, from time to time, to be able to be specific for outliers (e.g. for criminal investigations or cyber incidents). It's important to be able to have a structured approach to reidentify materials." This need for context-specific and adaptable anonymization approaches indicates that there is no one-size-fits-all solution for anonymization; instead, log privacy management should be dynamic, with strategies regularly evaluated and modified to address emerging requirements.

Summary

Surveying 45 industry experts, we discovered that IP addresses, MAC addresses, host names, and file paths are most often viewed as sensitive in software logs from the industry perspective. However, multiple factors, such as log utility, policies, and technical challenges, influence practitioners' decisions to anonymize specific attributes. In particular, balancing data privacy with utility poses a significant challenge. Our findings also suggest that current anonymization tools need to improve their efficiency concerning computational costs and processing time, especially given the diverse and unstructured nature of log data.

6.4 Discussion

6.4.1 What Should be Anonymized in Software Logs?

Based on the results of Section 6.3.4, the context in which software logs are generated and shared is crucial in determining what needs to be anonymized. It is essential to understand both internal organizational policies and regional regulations. Furthermore, knowing whether the data will be shared internally among employees, externally with third parties, or even publicly is vital. All these factors must be carefully considered when deciding which log attributes require anonymization.

Figure 6.2 presents the sensitive log attributes from three different perspectives: privacy regulations, articles and tools, and industry practices. From an industry perspective, IP addresses, MAC addresses, host names, file paths, and various IDs are recognized as the most sensitive attributes in software logs. These attributes are frequently cited by experts and commonly appear in publicly available logs. For example, 86% of survey respondents view IP addresses as sensitive, and they are found in 80% of software logs. While IP addresses have been well-researched, other attributes are less studied. For instance, although 52% of industry participants consider file paths sensitive, and they are present in 72% of analyzed logs, they are rarely discussed in the literature, with only 7% of reviewed articles addressing them.

Although attributes such as IP addresses and MAC addresses may not always be considered direct PII, they are classified as such under regulations such as GDPR due to their potential for re-identification when combined with other data. The presence of multiple attributes in logs increases this risk, necessitating a thorough evaluation of all exposed attributes for effective anonymization.

Based on our analyses of software log privacy from multiple perspectives, **we consider IP addresses, MAC addresses, host names, file paths, IDs, URLs, usernames, port numbers, and configuration details as generally sensitive information in software logs.** These attributes, particularly when combined, can disclose private user information or sensitive organizational details. Hence, a comprehensive anonymization strategy should consider all potential attribute combinations to prevent unintended disclosures.

6.4.2 Research Gaps and Future Directions

1. *Broadening the focus on diverse log attributes.* Comparing the findings from Tables 6.2, 6.3, and 6.5, it is clear that research papers often do not adequately cover

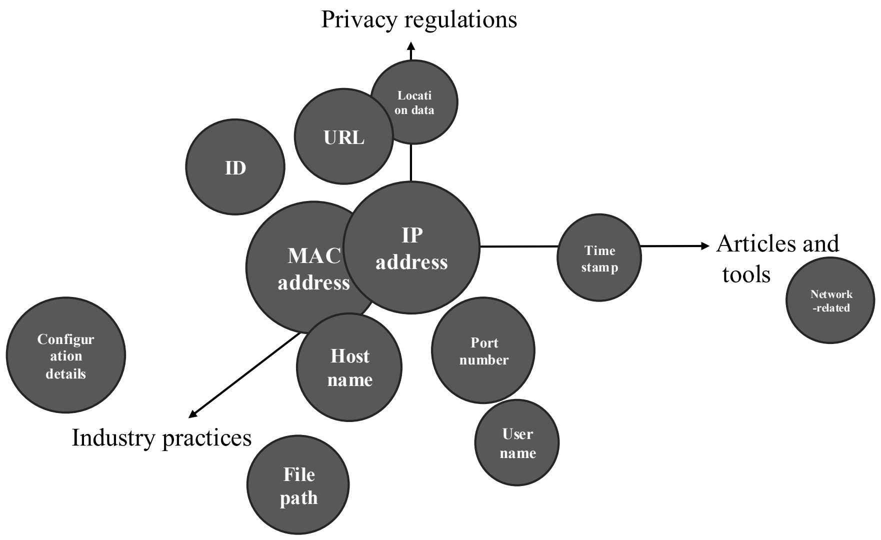


Figure 6.2 Sensitive log attributes from different perspectives

a broad range of sensitive log attributes. As discussed earlier, while IP addresses are extensively studied in the literature, other attributes, such as MAC addresses or file paths, have not been sufficiently explored. Future research should address these gaps for a more comprehensive understanding of log privacy. Additionally, a significant portion of existing studies focuses on anonymizing timestamps to prevent attacks. However, timestamps are often essential for log utility, especially for key tasks such as anomaly detection and performance prediction [302, 303]. We recommend future research focus on anonymizing other sensitive attributes that may reveal private information instead of relying heavily on timestamp modifications to better balance privacy and utility.

2. *Developing specialized anonymization tools for software logs.* The unique characteristics of software logs (such as their unstructured nature, the combination of strings and numerical data, and the high volume of entries) requires the development of specialized anonymization tools customized for these logs. While there exist some log anonymization tools, they are either very specific to some log types (e.g., network logs), handle a limited number of sensitive attributes (e.g., only IP addresses), or require a structured input format, making them unsuitable for unstructured logs. Future efforts should focus on creating software log anonymization tools that can manage a broad range of log attributes and overcome challenges such as computational efficiency and processing time, enhancing both privacy protection and data utility.

3. *Developing a privacy score for software logs.* As highlighted in this study, a combination

of various log attributes can lead to significant privacy risks. Future research could focus on designing a comprehensive privacy scoring system for software logs. This privacy score would evaluate the sensitivity of a log line or log file by analyzing the presence and combination of specific attributes, such as IP addresses, user identifiers, and network-related information. By establishing a threshold based on this score, organizations could more effectively assess whether sharing a particular log would pose a privacy risk. Such a scoring system could serve as a practical tool for data controllers to automate privacy assessments, enforce privacy policies, and mitigate potential data breaches while still enabling necessary data analysis.

4. *Examining privacy challenges in small-sized companies vs. larger organizations.* Future research could explore whether data privacy practitioners in small and medium-sized enterprises encounter unique challenges compared to those in larger organizations. Small companies often have fewer resources, less regulatory oversight, and limited expertise in log data privacy, which may impact their ability to implement effective anonymization and compliance measures. Investigating these differences could help tailor privacy solutions and regulatory frameworks to better support organizations of varying sizes.

5. *Comparing stated privacy policies with actual log data collection.* Future research could investigate the gap between what companies claim in their privacy policies and the log data they actually collect. Analyzing discrepancies between stated commitments and real-world practices could provide insights into transparency issues and help develop stronger regulatory guidelines for log privacy.

6.5 Threats to Validity

Internal Validity. The definition of "sensitivity" can differ between individuals and across industries. To mitigate this issue, we explored log privacy from multiple perspectives, including regulatory frameworks, academic research, and industry experts.

External Validity. We analyzed 25 publicly available log datasets; however, the findings may not be applicable to other types of logs, such as those used in enterprise or government environments. Similarly, while our survey provides valuable insights, it captures the views of a specific group of professionals and may not fully represent all industries or global practices. To mitigate this risk, we included a relatively large sample size of 45 participants to capture a diverse range of opinions. Moreover, our study focuses on specific data privacy regulations, which could limit its relevance to regions with different legal frameworks. Nevertheless, we focused on the most widely recognized privacy regulations to ensure broader applicability.

Construct Validity. The survey questions may not capture all aspects of privacy concerns in software logs and could concentrate on certain log types or attributes. To mitigate this, we first conducted a preliminary study to identify the most common attributes in software logs and designed our survey accordingly. Additionally, the parsing tool used in our log analysis might have accuracy limitations, and some log attributes may have been missed. However, by combining insights from log analysis, data regulations, research articles, and industry experts, we aimed to achieve a well-rounded view of sensitive attributes in software logs.

6.6 Chapter Summary

We conduct an in-depth analysis of privacy in software logs by examining the issue from multiple perspectives: regulatory frameworks, research literature, and industry practices. Our analysis of 25 publicly available log datasets identifies the most common log attributes that may require anonymization, such as IP addresses, MAC addresses, and host names. Our review of privacy regulations, including GDPR and CCPA, highlights the legal obligations to protect such sensitive data. Additionally, our systematic literature review reveals that while IP addresses and timestamps are commonly studied, other attributes, such as MAC addresses and file paths, are underexplored. Insights from a survey of 45 industry professionals further emphasize the challenges in balancing privacy and utility in log data, as well as in dealing with computational costs and processing time.

By bridging gaps between regulatory requirements, academic research, and industry practices, our study lays the groundwork for a more unified approach to privacy in software logs. Our contributions include a clear definition of sensitive information in software logs, identifying key gaps in current research and industry practices, and emphasizing the importance of context when determining which log attributes should be anonymized.

CHAPTER 7 ARTICLE 4: SDLOG: A DEEP LEARNING FRAMEWORK FOR DETECTING SENSITIVE INFORMATION IN SOFTWARE LOGS

Roozbeh Aghili, Xingfang Wu, Foutse Khomh, Heng Li, Submitted to TOSEM on 2025/05/20

7.1 Chapter Overview

In the last Chapter, we discussed the sensitive attributes of software logs. In this Chapter, we introduce a deep-learning framework to find these sensitive attributes in gigabytes and terabytes of software logs. In practice, log anonymization techniques primarily rely on regular expression patterns, which involve manually crafting rules to identify and replace sensitive information. However, these regex-based approaches suffer from significant limitations, such as extensive manual efforts and poor generalizability across diverse log formats and datasets. To mitigate these limitations, we introduce SDLog, a deep learning-based framework designed to identify sensitive information in software logs. Our results show that SDLog overcomes regex limitations and outperforms the best-performing regex patterns in identifying sensitive information. With only 100 fine-tuning samples from the target dataset, SDLog can correctly identify 99.5% of sensitive attributes and achieves an F1-score of 98.4%. To the best of our knowledge, this is the first deep learning alternative to regex-based methods in software log anonymization.

7.2 Chapter Design

In this section, we first define the objectives of this study and then detail the experiment setup.

7.2.1 Study Objectives

Software logs are messages recorded during the execution of a software system, providing crucial run-time information about events and activities. Typically created through logging commands such as *logger.info()*, log message can include different components depending on the logging configuration. Figure 7.1 shows a log statement generated using Python's *logging* module and contains four components: a timestamp indicating when the event occurred (e.g., 2025-03-26 08:53:29,653), a log level representing the message's severity (e.g., INFO), the function name that produced the log (e.g., *build_log_url*), and the message content, which forms the unstructured part of the log. Software logs are essential for developers, operators, and analysts to understand application execution [304], diagnose system failures [271, 305], detect anomalies [306,307], and monitor systems [269,270]. In many real-world deployments, logs remain often the only resource for understanding and resolving complex system challenges [269].

Modern software systems are complex and large-scale, generating huge volumes of logs (e.g., 120-200 million lines per hour [1]). Despite this substantial data volume and the extensive research focused on various aspects of software logs [272,308], publicly accessible log datasets remain limited. Several studies have highlighted this shortage and called for more open datasets [18, 101, 309, 310]. Although initiatives such as LogHub [101] and ITA [26] have attempted to address this gap, privacy concerns continue to discourage organizations from sharing their production logs, making publicly available datasets scarce.

To share log datasets while protecting privacy, anonymization becomes a critical prerequisite. Privacy regulations such as GDPR and CCPA mandate strict protection of personal and potentially identifiable information [311]. Log entries frequently contain sensitive attributes such as user identifiers, IP addresses, host names, and system configurations, which could compromise individual or organizational privacy if exposed [312]. Currently, anonymization techniques primarily rely on regular expression patterns, which involve manually crafting rules to identify and replace sensitive information [38, 313, 314]. However, these regex-based approaches have significant limitations: they require manual effort, struggle with context-aware identification, and often fail to capture nuanced variations in log structures across different systems and logging frameworks. The static nature of regex makes them particularly ineffective for handling the complex, semi-structured text in modern software logs, where sensitive information can appear in diverse and unpredictable formats.

Given the limitations of regex-based anonymization, we introduce SDLog: **S**ensitivity **D**etector in Software **Log**s, a deep learning-based framework designed to identify sensitive information in software logs. SDLog builds upon the pre-trained *CodeBERT* [176], which we fine-tune using sensitive attributes found in real-world logs. Our method uses CodeBERT's contextual understanding to address the limitations of regex-based techniques, which rely on fixed patterns and often fail to generalize across diverse log structures. Unlike regex, SDLog dynamically interprets surrounding context, making it well-suited for the complex and unstructured nature of real-world software logs. We validate our approach by annotating and analyzing 32,000 log lines from different software systems, demonstrating high precision in identifying sensitive attributes such as IP addresses, user identifiers, and system paths. We evaluate the effectiveness of our approach by answering the following RQs.

RQ1 ***How effective are regex-based approaches in identifying sensitive attributes***

```python
def build_log_url(base_url, container_id, log_file):
    url = f"{base_url}/containerlogs/{container_id}/{log_file}"
    logging.info(f"The URL for getting the log: {url}")
    return url
```

↓

```
2025-03-26 08:53:29,653 - INFO - build_log_url - The URL for getting the log: https://example.com/api/v1/containerlogs/container-123/application.log
```

Figure 7.1 A logging statement example in Python.

in software logs? We first investigate the effectiveness of regex-based methods in detecting sensitive attributes in software logs. To do this, we gather regex patterns from academic research, open-source projects, and industry practices, evaluating them on different sensitive attribute types. This evaluation allows us to identify variations, inconsistencies, and coverage gaps across different regex patterns.

RQ2 ***How effective is SDLog in identifying sensitive attributes in unseen software logs?*** We conduct a comprehensive evaluation across 16 diverse log datasets, benchmarking SDLog against the best set of regexes found in RQ1. Our evaluation consists of two parts: first, we assess SDLog's overall ability to distinguish sensitivity from non-sensitivity in software logs; second, we analyze SDLog's performance in accurately identifying specific categories of sensitive attributes, such as IP addresses, MAC addresses, and file paths.

RQ3 ***How effective is SDLog in identifying sensitive attributes when fine-tuned on a target dataset?*** In a real-world scenario, an organization might want to fine-tune SDLog using a subset of their log datasets to tailor the model's sensitive attribute detection to their specific data characteristics. To measure the performance of SDLog in this scenario, we evaluate the fine-tuned SDLog using the same evaluation procedure as in RQ2, analyzing both the sensitive attribute detection of SDLog as well as its performance on the specific categorization of each sensitive attribute.

To facilitate the integration of SDLog into log anonymization pipelines, we release our models, datasets, and supporting scripts. Both the SDLog Main and SDLog Net models are hosted on Hugging Face: [315, 316]. In addition, we provide the annotated dataset of 32,000 log lines used to train SDLog, along with scripts for dataset preparation and fine-tuning. These resources are available on our GitHub repository [317].

7.2.2 Named Entity Recognition (NER) Background

In this Chapter, we introduce a deep-learning framework to identify sensitive information in software logs. We model our problem as a *Named Entity Recognition (NER)* task. NER is one of the most common token classification tasks in NLP that attempts to find a label for each token in a natural language sequence [318, 319]. In NLP, NER serves not only as a standalone task for information extraction but also supports many tasks, such as automatic text summarization [320] and question answering [321]. A variety of methods have been proposed to tackle this task. In early approaches, manual engineering is greatly employed to design domain-specific features and heuristic rules [318]. Moreover, several probabilistic models, including *Hidden Markov Models (HMM)* [322], *Maximum Entropy (ME)* models [323], and *Conditional Random Fields (CRF)* [324] have been employed within supervised learning frameworks to perform this task. More recently, deep learning techniques have become dominant solutions to the NER task and have achieved promising results [319]. Neural NER models leverage deep learning architectures such as LSTM [325], *Convolutional neural network (CNN)* [326], and BERT [71], to learn contextual representations of tokens. To generate output label sequences, they use various decoding strategies, including CRFs, softmax classifiers, and span-based methods. Besides, language models can be fine-tuned with additional structures built on top of them to address specific tasks. For example, the NER task can be formulated as a *Machine Reading Comprehension (MRC)* task, which can be solved by fine-tuning a BERT model [327].

BERT [71] is a transformer-based architecture that has demonstrated a strong ability to model long-range dependencies in natural language by leveraging bidirectional attention mechanisms. This capacity enables BERT to effectively capture contextual information, making it highly suitable for a wide range of natural language understanding tasks. Following the success of BERT, numerous *Pre-trained Transformer-based Models (PTMs)* have been proposed, many of which have achieved state-of-the-art performance across diverse NLP benchmarks [328].

General-purpose BERT-based PTMs may face limitations in certain scenarios due to factors such as large model size, scalability issues, or suboptimal training strategies. To address these challenges, a variety of BERT-based variants have been proposed, each introducing modifications to the original BERT architecture or training setup to better suit specific requirements. For example, RoBERTa [329] retains BERT's core architecture but improves performance by removing the next sentence prediction task, dynamically changing masking patterns during training, and increasing the size of the training corpus and training duration. ALBERT [330] enhances BERT's scalability by introducing parameter reduction techniques such as factorized embedding parameterization and cross-layer parameter sharing, which significantly reduce the number of model parameters without sacrificing accuracy. DistilBERT [331] follows a different strategy by applying knowledge distillation to compress BERT into a lighter model. It reduces the number of parameters by approximately 40% while retaining over 95% of BERT's language understanding capabilities, making it more suitable for resource-constrained environments.

While these general-purpose models work well in a broad range of NLP tasks, they often struggle with domain-specific texts such as those found in software engineering, which are rich in technical jargon and structured expressions that differ significantly from standard natural language. Therefore, researchers have adapted the BERT architecture to handle program-related data, such as source code and technical discussions, and proposed various variants of BERT models as well. CodeBERT [176] is trained on paired data consisting of source code and corresponding natural language descriptions, enabling it to generate semantically rich embeddings for both types of input. This dual understanding allows CodeBERT to perform well in various tasks such as code search [176], code summarization [176], automated program repair [332], and question answering [333]. Similarly, BERTOverflow [334] is fine-tuned on data from Stack Overflow. It incorporates NER to better handle technical terms and identifiers commonly used in developer discussions, thereby improving its performance on software-related tasks.

7.2.3 Experiment Setup

In this section, we begin by introducing the sensitive attributes found in software logs, then describe the datasets used for training our model and outline our annotation methodology. Finally, we present the design of our model in identifying sensitive attributes in logs.

7.2.3.1 Sensitivite Attributes in Software Logs

Software logs contain various attributes, including IP addresses, timestamps, and log levels. We use the definition of sensitivity in software logs proposed in Chapter 6, classifying IP addresses, MAC addresses, host names, file paths, IDs, URLs, usernames, port numbers, and configuration details as commonly recognized sensitive attributes in software logs. Table 7.1 presents these attributes along with an example for each.

7.2.3.2 The Benchmark Dataset

We use the log datasets from LogHub [101] to build and test our sensitive attribute detection framework. LogHub contains 16 datasets collected from both open-source and commercial systems across different domains of *distributed systems* (HDFS, Hadoop, Spark, Zookeeper, OpenStack), *supercomputers* (BGL, HPC, Thunderbird), *operating systems* (Windows, Linux, Mac), *mobile systems* (Android, HealthApp), *server applications* (Apache, OpenSSH), and *standalone software* (Proxifier). These datasets have been widely adopted in prior research on software logs [335–337]. Each dataset includes a subset of 2,000 logs with manually extracted log templates serving as the ground truth. Throughout the remainder of this paper, we refer to this collection of subsets (16 datasets with 2,000 logs each, totaling 32,000 logs) as the Benchmark Dataset.

7.2.3.3 Sensitive Attribute Annotation Process

Our sensitive attribute detection framework relies on fine-tuning a pre-trained LLM. Annotated training data is required to fine-tune the model in order to identify sensitive attributes in software logs. We use the Benchmark Dataset, labeling each word in each of the 32,000 log lines with its corresponding attribute. The annotation follows the *Inside-Outside-Beginning (IOB)* format [338], where *B-* denotes the beginning of an attribute, *I-* is used for subsequent words in the attribute, and *O* indicates non-attribute words. Each sensitive attribute is annotated with its abbreviation along with the respective IOB prefix, while non-sensitive words are marked as *O*.

To prepare the data, we preprocess each log message by splitting it into tokens using whitespace as the delimiter, treating each resulting word as a token for sequence labeling. This approach works well for most attributes, such as URLs, file paths, and IDs. However, network-related information, including IP addresses, host names, and port numbers, often appear concatenated (e.g., *10.250.18.114:50010*), with no whitespace separating them. Standard tokenization fails to isolate such components. To handle this issue, we introduce a new category, *net*, as a parent attribute of IP addresses, host names, and port numbers. This design choice enables the model to recognize and classify network-related entities even when they are concatenated, ensuring accurate detection despite the absence of whitespace separation.

Figure 7.2 demonstrates the annotation process for a log message from the *HDFS* dataset. In this example, the word *10.250.18.114:50010* is a combination of IP address and port number, therefore it belongs to the network category and we annotate it as *B-net*; and finally, we annotate the *blk_-5140072410813878235* as *B-ID*.

Table 7.1 Types and examples of sensitive attributes in software logs

Attribute	**Example**
IP Address	*Invalid user webmaster from 173.234.31.186*
MAC address	*Roamed or switched channel, reason #8, bssid 5c:50:15:4c:18:13, last RSSI -64*
Host name	*proxy.cse.cuhk.edu.hk: 5070 close, 0 bytes sent, 0 bytes received, lifetime 00:01*
File path	*workerEnv.init() ok /etc/httpd/conf/workers2.properties*
ID	*Verification succeeded for blk_-4980916519894289629*
URL	*the url = http://baike.baidu.com/item/%E8%93%9D%92%8C/462624?fr=aladdin*
Username	*Invalid user webmaster from 173.234.31.186*
Port number	*proxy.cse.cuhk.edu.hk: 5070 close, 0 bytes sent, 0 bytes received, lifetime 00:01*
Configuration details	*mapResourceRequest:<memory:1024, vCores:1>*

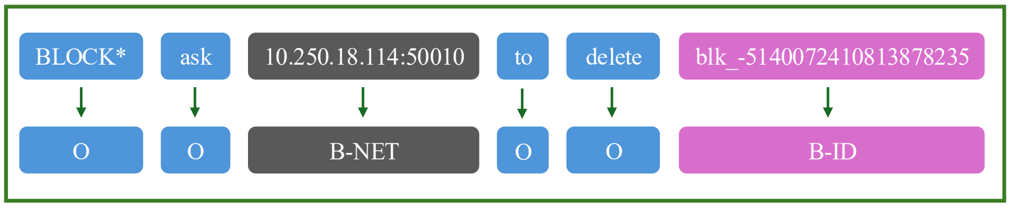


Figure 7.2 An example of our data annotation process

To further address the challenge of detecting concatenated network-related entities, we derive a specialized subset of the Benchmark Dataset containing only net attributes, and we refer to it as the Net Benchmark Dataset. In this variant, we preprocess network-related tokens by explicitly separating their components using special characters. For example, a combined network token such as *10.250.18.114:50010* is split into two distinct tokens of *10.250.18.114* and *50010*. This allows us to assign more specific labels, annotating them as *B-IP* and *B-PORT*, respectively. This separation enables the model to better distinguish between different types of network information, improving its ability to accurately detect and classify sensitive net attributes.

In order to annotate the sensitive attributes in the Benchmark Dataset, the first two coders independently performed an open coding procedure on 200 log templates randomly selected from all the log templates of all the datasets. The annotation methodology follows the systematic guidelines proposed by Basili et al. [150] and is similar to prior studies [339, 340].

Step 1: First Round of Coding, Discussion, and Revision. The Benchmark Dataset contains 1,363 log templates in total. To initiate the annotation process, we randomly sampled 100 templates. Each coder independently annotated the sensitive attributes. During this phase, the coders did not have a shared annotation protocol. After completing their individual annotations, the coders shared their responses and discussed their annotation reasoning.

The goal of this discussion session was to resolve inconsistencies and establish a shared understanding. Finally, each coder revised their initial annotations according to the consensus reached.

Step 2: Measuring Reliability for First Round. Establishing reliability is crucial to validate the annotation process [155]. We evaluated inter-coder agreement using both Cohen's kappa [156] and accuracy, as Cohen's kappa alone can underestimate agreement in datasets like ours, where most tokens are labeled as non-sensitive (i.e., annotated as *O*). In the first round, the coders achieved a Cohen's kappa of 0.53 and an accuracy of 0.82, with most disagreements related to the annotation of IDs and configuration details.

Step 3: Second Round of Coding, Discussion, and Revision. Following the initial round, the coders conducted another cycle of independent coding, discussion, and revision on a new random sample of 100 log templates. By this stage, a clearer and more consistent understanding of the annotation guidelines had been developed.

Step 4: Measuring Reliability for Second Round. After the second round, we again measured the inter-coder agreement. The Cohen's kappa improved significantly to 0.87, with an accuracy of 0.94. A Cohen's kappa greater than 0.80 indicates strong reliability in the annotation process [157], confirming that a consistent annotation guideline was successfully established.

Step 5: Full Dataset Annotation. Upon achieving a high level of agreement and establishing a clear annotation protocol, the remaining log templates in the Benchmark Dataset were annotated by the first coder.

Table 7.2 presents the detailed statistics of the Benchmark Dataset. Among the 16 datasets, *Apache* contains the fewest log templates with only 6 templates, while *Mac* has the most with 341 templates. The median number of templates across datasets is 50. The proportion of sensitive attributes also varies significantly; *HealthApp* contains no sensitive attributes, whereas every template in *HDFS* and *Proxifier* includes sensitive attributes. Overall, based on the median across all datasets, approximately one-third of the log templates (32.4%) contain sensitive attributes. Among the different types of sensitive information, net (i.e., IP addresses, host names, and port numbers) is the most common, appearing in 11.2% of log templates in median. It is followed by ID and file path attributes, with median percentages of 9.8% and 5.6%, respectively. The remaining attributes (i.e., MAC addresses, URLs, usernames, and configuration details) have a median percentage of 0, indicating that less than half of the datasets contain these attributes. For instance, although 48.1% of *OpenSSH* templates contain usernames, 13 out of the 16 datasets have no usernames at all. These observations highlight the need for a generalizable sensitive attribute detection pipeline capable of handling various attributes across diverse datasets, as relying on a fixed set of attributes could result in missing important sensitive information.

7.2.3.4 SDLog: Sensitivity Detector in Software Logs

We propose SDLog, a deep learning-based framework designed to detect sensitive attributes in software logs. SDLog can determine whether a word in a log message corresponds to sensitive information and, if so, assign it to the appropriate category (e.g., username). Researchers and developers can leverage SDLog as an alternative to regular expression-based approaches for identifying sensitive data that needs to be anonymized.

We formulate the detection task as a sequence tagging problem, a well-established approach in the field of NLP [341, 342]. Sequence tagging is commonly used for NER; for example, to identify people, organizations, and locations in a sentence. Similarly, in our context, given a log message, SDLog classifies each word as either non-sensitive or sensitive, and labels sensitive words with their corresponding categories.

To build SDLog, we leverage CodeBERT, a pre-trained large language model released by Microsoft [176]. CodeBERT extends the BERT architecture to handle both natural language and programming languages. It is a bimodal model trained jointly on paired natural language and code data. Specifically, it was trained on six popular programming languages: Python, Java, JavaScript, PHP, Ruby, and Go, using a combination of masked language modeling and replaced token detection objectives. CodeBERT consists of 12 Transformer encoding layers with a total of 125 million parameters. It supports a variety of tasks, including code-to-code translation, code-to-text generation, text-to-code synthesis, and text-to-text tasks. Additionally, it has demonstrated strong performance on tasks such as code search and automatic code documentation generation, making it a suitable model for applications involving both source code and natural language. We choose CodeBERT as the backbone of SDLog because it has been pre-trained on programming languages in addition to natural language, allowing it to better model the semi-structured, code-like patterns that are often present in software logs. Unlike models such as RoBERTa [329] or the original BERT [175], which are exclusively pre-trained on natural language corpora, CodeBERT is specifically optimized to handle syntax, identifiers, and structural elements commonly found in code and, by extension, in log messages.

Figure 7.3 presents an overview of our approach to identify sensitive attributes in software logs. We implement SDLog as described in the following sections.

Pre-tokenization Process. Before applying the pre-trained tokenizer of the CodeBERT model, we follow the same procedure discussed in Section 7.2.3.3, splitting each log message

Table 7.2 Proportion of sensitive attributes in the Benchmark Dataset

Dataset	**# Templates**	**Sensitive (%)**	**Net (%)**	**MAC (%)**	**File path (%)**	**ID (%)**	**URL (%)**	**Username (%)**	**Config (%)**
Android	166	10.2	0.0	0.0	0.6	9.6	0.0	0.0	0.0
Apache	6	66.7	16.7	0.0	33.3	33.3	0.0	0.0	0.0
BGL	120	22.5	6.7	3.3	12.5	3.3	0.0	0.0	0.0
Hadoop	114	58.8	14.0	0.0	5.3	42.1	1.8	0.0	5.3
HDFS	14	100.0	71.4	0.0	14.3	100.0	0.0	0.0	0.0
HealthApp	75	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
HPC	46	26.1	15.2	0.0	0.0	21.7	0.0	0.0	0.0
Linux	118	28.8	7.6	0.0	2.5	6.8	0.0	2.5	15.3
Mac	341	18.2	5.3	0.6	7.3	1.8	4.4	0.0	0.6
OpenSSH	27	88.9	70.4	0.0	0.0	22.2	0.0	48.1	0.0
OpenStack	43	95.3	16.3	0.0	27.9	65.1	0.0	0.0	16.3
Proxifier	8	100.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0
Spark	36	47.2	8.3	0.0	2.8	25.0	11.1	0.0	11.1
Thunderbird	149	26.8	6.0	4.0	6.0	2.7	0.0	6.0	9.4
Windows	50	20.0	0.0	0.0	10.0	10.0	0.0	0.0	2.0
Zookeeper	50	36.0	22.0	0.0	6.0	8.0	0.0	0.0	0.0
Median	**50**	**32.4**	**11.2**	**0.0**	**5.6**	**9.8**	**0.0**	**0.0**	**0.0**

using whitespace as the delimiter. As a result, each word is treated as an individual token, and these tokens are the smallest units processed by the model.

Classification Head. A single fully connected layer is attached to the output representations of the CodeBERT structure to project each token vector onto the IOB label space. A softmax layer then converts these mappings into probability distributions over the label classes. The entire model, including both the classifier and the underlying encoder, is fine-tuned end-to-end using a cross-entropy loss computed on the ground-truth token-level annotations.

SDLog Hierarchy. SDLog consists of two specialized pre-trained transformer-based models, each optimized for different levels of granularity in sensitive attribute detection. The first model, a large-scale transformer, is fine-tuned on log data annotated with eight distinct categories of sensitive information: net, MAC, file path, ID, URL, username, and configuration details. This model is trained using the full Benchmark Dataset, enabling it to generalize across a broad range of attribute types and structural patterns observed in real-world logs. The second model is a lightweight transformer specifically fine-tuned to perform fine-grained classification within the broader net category. Unlike the first model, this one is trained exclusively on the Net Benchmark Dataset. It learns to distinguish between network-related entities, including IP addresses, port numbers, and host names. This hierarchical modeling approach allows SDLog to first identify coarse-grained sensitive spans and then apply a

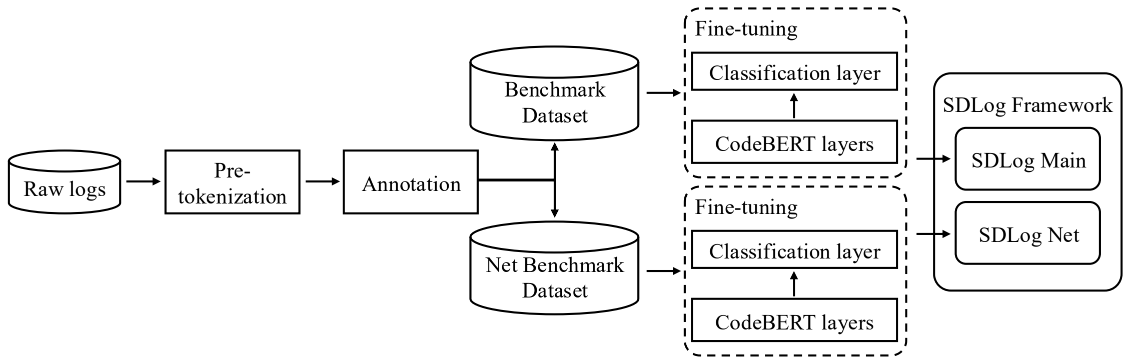


Figure 7.3 An overall diagram of SDLog framework

more targeted classification to resolve ambiguous network-related entities, improving precision in scenarios with complex or overlapping attribute types. Both models follow the same procedure and use the same hyperparameters.

Hyperparameter Settings. We fine-tune a transformer-based token classification model, initialized from the *microsoft/codebert-base* checkpoint [176], to perform NER over log data. The model is optimized using the AdamW optimizer with a learning rate of 5e-5 and a weight decay of 0.01. Considering the size of the log dataset, training is conducted for 2 epochs with a linear learning rate scheduler. Evaluation is performed every 300 training steps, and checkpoints are saved at the same interval. To promote generalization and avoid overfitting, the best-performing model, based on validation loss, is retained at the end of training. These hyperparameter choices follow standard practices for adapting pre-trained language models to token-level classification tasks [343].

Experimental Environment. All experiments are conducted on a workstation equipped with an 11th Gen Intel Core i7-11700K CPU (8 cores, 16 threads, up to 5.0 GHz), 32 GB of RAM, and an NVIDIA GeForce GTX 1060 GPU with 3 GB of memory. The system runs a 64-bit Linux environment with Python 3.12, and all models are trained using PyTorch [344] and the Hugging Face Transformers library [345].

7.3 Study Approach and Results

This section presents the details of our RQs and their results. We organize each RQ by its motivation, approach, and results.

7.3.1 RQ1. How effective are regex-based approaches in identifying sensitive attributes in software logs?

7.3.1.1 Motivation

Identifying sensitive attributes is a crucial first step in most log anonymization pipelines, as it determines which log attributes should be anonymized. Regular expressions have traditionally been the accepted method for this task in both research and industry. However, despite their widespread adoption, there is a notable lack of standardized regex patterns even for commonly identified attributes such as IP addresses and URLs, which can lead to inconsistencies in detection. Furthermore, the effectiveness of regex-based methods across various attribute types and log formats remains largely unexplored. A systematic evaluation of these limitations can provide a clearer understanding of current practices, highlight specific weaknesses, and support the case for alternative approaches.

7.3.1.2 Approach

We perform a comprehensive search across three research databases, initially identifying 185 relevant articles. We review each article and its replication package (if available) according to inclusion and exclusion criteria to extract regex patterns that identify sensitive attributes in software logs. In addition, we collect regex patterns from three industry partners, which they use in their anonymization pipelines. Our selection process adheres to the systematic methodology proposed by Wohlin et al. [207] and is detailed below.

Searching Research Databases. Our literature search is conducted across three widely recognized academic databases: IEEE Xplore Digital Library [208], ACM Digital Library [209], and Engineering Village [346], which itself aggregates three databases of Compendex, Inspec, and Knovel. We aim to identify conference and journal papers that use regex patterns for detecting sensitive attributes in software logs. To achieve this, we formulate the following search query:

("Abstract": log) AND ("Abstract": privacy OR "Abstract": sensitiv OR "Abstract": anonymiz* OR "Abstract": log pars*) AND ("Full Text": regex OR "Full Text": "regular expression")*

This query targets articles that contain the terms *log* and *regular expression* or *regex* while also falling within the domains of privacy, log anonymization, or log parsing. We include "log pars*" keyword because many log parsers use regex patterns as a preprocessing step [273,347].

Applying this query, we find 133 articles from the IEEE Xplore Digital Library, 47 from the ACM Digital Library, and 38 from Engineering Village, amounting to a total of 218 articles. After removing duplicate articles, we get a total of 185 papers for further analysis.

Inclusion and Exclusion Criteria. We focus on identifying papers that use regex to detect one or more sensitive attributes discussed in Section 7.2.3.1. We review tables, figures, and code snippets of each paper, searching for keywords such as "regex" and "regular expression". Additionally, we check whether the article includes a replication package. If available, we review it and look for signs of regex patterns, such as the presence of "import re" in Python projects.

We exclude articles that focus on data types other than software logs (e.g., search logs) or those that use regex patterns to identify attributes outside the scope of our defined sensitive attributes (e.g., timestamps). Additionally, we do not consider patterns that are specific to one software system. For example, Li et al. [348] use "blk_-?\d+" to detect block IDs in *Hadoop* logs and "core\.\d+" to identify core IDs in *BGL* logs. However, we do not consider these regex patterns for the ID attribute, as they are specific to particular systems.

Industry Regexes. In addition to the academic literature, we also collect regex patterns from three of our industry partners. These partners are large-scale organizations, each with more than 1,000 employees, where they manage extensive log datasets derived from supercomputers, operating systems, and server application environments. The regexes provided by these partners are part of their anonymization pipelines, designed to detect sensitive attributes within their software logs prior to data sharing, either internally or externally. We apply the same inclusion and exclusion criteria discussed in Section 7.3.1.2 to these regex patterns, capturing those that detect sensitive attributes while discarding patterns customized to specific systems or use cases (e.g., a server name pattern unique to a company).

Regex Extraction. We extract regex patterns from research papers, their replication packages, and industry partners, organizing them by attribute type (e.g., IP address and URL). Each article, replication package, or industry partner may provide regex patterns for one or more log attributes.

Evaluating Regex Patterns. To systematically evaluate the effectiveness of the extracted regex patterns, we run each regex pattern individually against the Benchmark Dataset, which includes 32,000 logs from 16 different datasets. For each execution, we compute standard evaluation metrics including precision, recall, and F1-score, based on token-level ground truth annotations. True positives are counted when the regex-matched text overlaps with the labeled sensitive attributes; false positives are counted when a match occurs outside that sensitive attribute region; and false negatives capture labeled sensitive attributes that were not detected using the regex pattern. Therefore, in this evaluation, precision measures how many of the matches found by the regex patterns were correct, and recall measures how many of the true labeled attributes the regex patterns successfully found. This analysis enables a detailed comparison of regex patterns, highlighting their strengths and weaknesses in identifying different types of structured information within heterogeneous logs.

7.3.1.3 Results

No common regex ground truth exists to identify log sensitive attributes. Table 7.3 summarizes the complete set of regex patterns identified through our systematic search. In total, we collected 41 distinct regex patterns across 8 different attribute types, namely IP addresses, MAC addresses, file paths, IDs, URLs, usernames, ports, and configuration details. We did not find any regex patterns for host names. Among the found regexes, 17 patterns specifically target IP addresses and are used by 20 different sources. Similarly, for other attributes such as file paths, IDs, URLs, and usernames, each source appears to have developed its own set of regex patterns. This leads to our first key observation: there is no unified or widely adopted set of regex patterns that researchers and practitioners rely on. Instead, every article, project, or company that requires the identification of sensitive attributes within software logs seems to independently curate and maintain its own collection of regex patterns.

Regex patterns show large performance variability for the same attribute. Our evaluation reveals that regex patterns designed for the same attribute can exhibit vastly different levels of performance depending on subtle variations in their construction. For instance, among the regexes for IP addresses, F1-scores range from extremely low values such as 0.0%, 1.1%, and 9.1%, to relatively high values such as 88.4% and 88.5%. Similarly, for regexes designed to detect URLs, we observe F1-scores as low as 0.0%, 1.4%, and 1.8%, alongside patterns achieving F1-scores up to 95.1%. These results reveal two major challenges in designing and using regex patterns: every small detail in the pattern matters, and regex patterns often lack generalizability across different log formats and datasets.

Small differences in regex design have a large impact. To illustrate the importance of fine details, consider two regex patterns intended to detect IP addresses: (1) `((\d+).(\d+).(\d+).(\d+))` and (2) `\d+\.\d+\.\d+\.\d+`. Both patterns look similar, as

Table 7.3 Performance of individual regular expressions in categorizing sensitive attributes on the Benchmark Dataset

	IP address	**P (%)**	**R (%)**	**F1 (%)**	**Source**
1	`(/|)([0-9]+\.){3}[0-9]+(:[0-9]+|)(:|)`	**92.1**	85.1	88.4	[282, 349–351]
2	`([0-9.]+)\s`	11.4	48.8	18.5	[352]
3	`([0-9]+.){3}[0-9]+(:[0-9]+)`	23.0	0.6	1.1	[353]
4	`((\d+).(\d+).(\d+).(\d+))`	32.5	**99.7**	49.0	[354]
5	`(\d)+3\d(:\d+)?`	8.6	9.6	9.1	[355]
6	`(\d+\.){3}\d+(:\d+)?`	**92.1**	85.1	88.4	[356]
7	`^(25[0-5]|2[0-4]\d|[0-1]?\d?\d)(\.(25[0-5]|2[0-4]\d|[0-1]?\d?\d)){3}$`	0.0	0.0	0.0	[357]
8	`(\b\d{1,3}(?:\.\d{1,3}){3}\b)`	**92.1**	85.1	**88.5**	[358]
9	`(\d{1,3}(?:\.\d{1,3}){3}):?\d*`	**92.1**	85.1	**88.5**	[358]
10	`\d+\.\d+\.\d+\.\d+`	**92.1**	85.1	88.4	[348]
11	`(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})[,:  )]`	90.7	78.6	84.2	[359]
12	`[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}`	31.5	**99.7**	47.9	[360]
13	`(/|)(\d+.){3}\d+(:\d+)?`	32.5	**99.7**	49.1	[273]
14	`[0-9]+\.[0-9\.:]*[0-9]`	56.7	85.1	68.1	[361]
15	`(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})`	**92.1**	85.1	**88.5**	Company 1
16	`\b\d{1,3}(?:\.\d{1,3}){2,}\b`	91.8	85.1	88.3	Company 2
17	`(\b\d{1,3}\.)(\d{1,3}\.)(\d{1,3}\.)(\d{1,3}\b)`	**92.1**	85.1	**88.5**	Company 3
	MAC address	**P (%)**	**R (%)**	**F1 (%)**	**Source**
1	`^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$`	0.0	0.0	0.0	[273]
	File path	**P (%)**	**R (%)**	**F1 (%)**	**Source**
1	`((((?<!\w)[A-Z,a-z]:)|(\.{1,2}\\))([^\b%\/\\|:\n\"]*))|(\"\2([^%\/\\|` `:\n\"]*)\")|((?<!\w)(\.{1,2})?(?<!\/)(\/((\\\b)|[^\b%\\|:\n\"\\\/])+)+\/?)`	59.0	98.3	73.7	[359]
2	`/[\w/.  :-]+`	55.6	99.5	71.3	[362]
3	`(/[^/\s]+)+`	48.1	99.5	64.9	[362]
4	`(([A-Z]:)|)(/\S+)+`	47.8	99.5	64.5	[351]
5	`(/|)(([\w.-]+|\<\*\>)/)+([\w.-]+|\<\*\>)`	**66.7**	98.1	**79.4**	[273]
6	`([A-Za-z]:|\.){0,1}(/|\\)[0-9A-Za-z\-_\.:/\*\+\$#@!\\\?=%&]+(?<![:\.])`	48.1	**100.0**	65.0	[361]
7	`\/(\S+)`	47.8	99.5	64.5	Company 3
	ID	**P (%)**	**R (%)**	**F1 (%)**	**Source**
1	`(?:UUID|GUID|version|id)[\\=:\"\'\s]*\b[a-fA-F0-9]{8}-` `[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}\b`	0.0	0.0	0.0	[362]
2	`<([^>]+)>`	2.6	0.1	0.2	[362]
3	`[pP]id[:|-|=|\s/]*(\d+)`	97.7	1.3	2.5	[359]
4	`[uU]id[:|-|=|\s/]*(\d+)`	**99.8**	**23.5**	**38.1**	[359]
	URL	**P (%)**	**R (%)**	**F1 (%)**	**Source**
1	`[A-Za-z\.]+://[A-Za-z0-9\.\/\+#@:_\-]+(?<![:\.])`	91.4	99.2	**95.1**	[361]
2	`(https?://\S+)`	**100.0**	39.1	56.2	[359]
3	`https?://[^\s#]+#[A-Za-z0-9\-\=\+]+`	0.0	0.0	0.0	[362]
4	`http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\).]|` `(?:\%[0-9afA-F][0-9a-fA-F]))+`	**100.0**	39.1	56.2	[362]
5	`([\w-]+\.)+[\w-]+(:\d+)?`	0.9	**100.0**	1.8	[363]
6	`(\S+\.\S+(\.\S+)+(:\d+)?)|(\w+-\w+(-\w+)+)`	0.7	**100.0**	1.4	[351]
7	`\bhttps?://(www.)?[a-zA-Z0-9-]+(.[a-zA-Z]{2,})+(:[0-9]{1,5})?(/[^\s]*)?\b`	**100.0**	31.2	47.6	[273]
	Username	**P (%)**	**R (%)**	**F1 (%)**	**Source**
1	`user( | )[A-Za-z0-9]+(?<!request)(?!  methods)`	**42.0**	25.3	31.6	[361]
2	`user\:\s(\w+)`	0.0	0.0	0.0	[364]
3	`r?[uU]ser[:|-|=|\s/]*<(\w+)>|r?[uU]ser[:|-|=|\s/]*(\w+)`	35.2	**72.0**	**47.3**	[359]
	Port	**P (%)**	**R (%)**	**F1 (%)**	**Source**
1	`[pP]ort[=:  |:|=|:  |\s/]*(\d1,5)`	**96.0**	**8.1**	**15.0**	[359]
	Configuration details	**P (%)**	**R (%)**	**F1 (%)**	**Source**
1	`size\s+(\d+)`	**19.2**	**14.2**	**16.3**	[358]

each aims to match four numeric blocks separated by dots. However, their performance differs considerably. The first pattern achieves a very high recall of 99.7%, successfully detecting almost all IP addresses present in the dataset. Nevertheless, its precision is low, 32.5%, indicating that it also incorrectly identifies many non-IP strings as IP addresses. In contrast, the second pattern yields a high precision of 92.1%, correctly labeling true IP addresses in most cases, but has a slightly lower recall of 85.0%. Consequently, the first pattern reaches an F1-score of 49.0%, while the second achieves a much higher F1-score of 88.4%. This example shows how small differences in regex structure can significantly impact precision, recall, and overall effectiveness.

Regex patterns often lack generalizability. Generalizability presents another critical limitation. For example, the regex pattern *^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$* completely failed to detect any MAC addresses in the Benchmark Dataset. This failure is due to the pattern's assumption that the MAC address must be the only content in the log message. In real-world logs, however, attributes are typically embedded among various other strings. With a small modification to the pattern and changing it to *\b([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})\b*, we observe a dramatic improvement, with a precision of 98.8%, recall of 100.0%, and F1-score of 99.4%. A similar issue was observed with an IP address pattern *^(25[0-5]|2[0-4]\d|[0-1]?\d?\d)(\.(25[0-5]|2[0-4]\d|[0-1]?\d?\d)){3}$*. By changing it to *\b(25[0-5]|2[0-4]\d|[0-1]?\d?\d)(\.(25[0-5]|2[0-4]\d|[0-1]?\d?\d)){3}\b*, the initial F1-score of 0.0% increases to 88.5%. These findings emphasize that regex patterns often fail to generalize across different log structures and datasets unless they are explicitly designed to handle common variations in log formatting.

Regex patterns can achieve reasonable results in certain attributes that follow common patterns (e.g., IP addresses or file paths. Looking at the best-performing regex patterns in Table 7.3, shown in bold, we find that regexes are capable of achieving reasonable results for specific types of sensitive attributes, particularly IP addresses, MAC addresses (with the improved pattern discussed in the last paragraph), file paths, and URLs. The best F1-score in these attributes are 88.5%, 99.4%, 79.4%, and 95.1%, respectively. This relatively high performance is largely due to the fact that these attributes often follow strict and predictable patterns that remain consistent across a variety of log formats and datasets. Nevertheless, it is important to highlight that these figures reflect the best-performing regex patterns. The overall performance across all regexes for a given attribute shows more variability; for example, the median F1-score for file paths is 65.0% and for URLs is 47.6%, indicating that many regex patterns still struggle to generalize effectively even within structured attribute types.

However, regex patterns are not well-suited for detecting certain attributes that are constrained to rigid structures (e.g., IDs or usernames). As shown in Table 7.3, regex patterns generally perform poorly for detecting certain attributes such as IDs, usernames, ports, and configuration details. This is primarily due to the absence of a consistent and rigid structure for these attributes, which makes them difficult for regex patterns to capture effectively. The data reflects this, with fewer researchers and practitioners developing regex patterns for these attributes. For example, while we identified 17 distinct regex patterns for detecting IP addresses, only 4, 3, 1, and 1 patterns were found for IDs, usernames, ports, and configuration details, respectively. We also did not find any regex patterns for host names. The lack of consistent patterns for these attributes is widely recognized, and as a result, fewer efforts are made to address them with regex. Hence, these attributes are often overlooked in detection efforts, with anonymization pipelines typically focusing on more structured attributes such as IP addresses and file paths. For those who do attempt to use regex for these attributes, the results are often of low quality. For instance, the best-performing regex pattern for detecting IDs achieved an F1-score of just 38.1%, while the best pattern for detecting usernames had an F1-score of 47.3%. The performance for detecting ports and configuration details is even worse, with F1-scores of 15.0% and 16.3%, respectively. These attributes often appear in a variety of formats, making them even harder to detect. For example, usernames may or may not be preceded by the keyword *user*, complicating the task for regex patterns designed with specific assumptions. Similarly, ports can be located in various positions within a log; immediately after an IP address, immediately after a host name, or embedded in text, making it a challenging task for strict regex patterns to distinguish them from other integers.

Summary

Our analysis shows that there is no common ground truth for regex patterns to detect sensitive attributes in software logs; instead, researchers and practitioners tend to develop their own patterns, leading to inconsistency and fragmentation. Regex performance varies significantly even within the same attribute type, with small design differences causing large impacts on precision and recall. While regex can achieve reasonable results for well-structured attributes such as IP addresses, it performs poorly for less structured attributes such as IDs, usernames, ports, and configuration details. Overall, regex patterns often lack generalizability across datasets and are not well-suited for attributes without strict formatting.

7.3.2 RQ2. How effective is SDLog in identifying sensitive attributes in software logs?

7.3.2.1 Motivation

Although regular expressions provide a structured, rule-based method for detecting sensitive information in software logs, they often fall short when applied to the wide range of log formats and datasets, limiting their practical utility. In contrast, machine learning methods, such as SDLog, have the potential to address these shortcomings by leveraging contextual and structural information within the data. To demonstrate the effectiveness of SDLog, We evaluate its ability to accurately detect sensitive attributes in previously unseen log datasets. Because log formats vary significantly across systems in terms of syntax, terminology, and structure, testing the model's ability to generalize is essential. By comparing SDLog to the best-performing regex patterns, we establish a performance baseline and assess whether a learning-based solution is suitable for deployment in dynamic, heterogeneous logging environments.

7.3.2.2 Approach

We evaluate the effectiveness of SDLog in identifying sensitive attributes and compare its results with the best-performing regex patterns found in RQ1.

Baseline As our baseline, we select the best-performing regex pattern for each attribute from Table 7.3, based on the highest F1-score. In cases where multiple regex patterns for the same attribute achieve identical F1-scores, we randomly choose one. This results in a set of eight regex patterns, each corresponding to one of the following attributes: IP address, MAC address, file path, ID, URL, username, port, and configuration details. Using these patterns, we design an anonymization pipeline that simulates real-world usage. In contrast to our evaluation on RQ1 where we ran each regex pattern separately in order to find the best regex patterns, in this RQ and to build our regex baseline, each log line is passed through all eight regexes. When a pattern detects a match, the matched value is replaced with a placeholder (e.g., an IP match is replaced with *$IP*).

Prior research has suggested that the order in which regex patterns are applied can impact the outcome [273]. For example, if a loosely written ID regex partially matches an IP address and is executed before the IP regex, the true IP may be incorrectly masked as *$ID*. However, if the IP regex is executed first, it correctly identifies and replaces the IP address. To evaluate the impact of regex execution order, we design a sampling strategy. Out of the

8! possible permutations of the eight regex patterns, we randomly sample 500 orders and run each through our anonymization pipeline. Finally, we report the minimum and maximum precision, recall, and F1-score values observed for each sensitive attribute across these different orderings.

SDLog Evaluation We use cross-validation to evaluate the performance of SDLog. Given the variability in syntax and structure across software logs, this step is critical for evaluating the model's performance on unseen data. For the first, large-scale model of SDLog, we use *Leave-One-Out Cross-Validation (LOOCV)* [365], training the model on all datasets except one and testing it on the excluded dataset. This procedure is repeated until each dataset has been used as the test set once. By conducting LOOCV over our diverse set of logs, we obtain a rigorous and comprehensive evaluation of SDLog's effectiveness in identifying sensitive attributes across varying log formats and sources. For the second model focused on net sub-attribute classification, we use 4-fold cross-validation [366], splitting the 16 datasets into 12 for training and 4 for testing in each round. 4-fold cross validation is more suitable than LOOCV for the second model, as three datasets contain no net attributes, making k-fold validation a better fit for balanced evaluation.

We evaluate SDLog through a two-step process. First, we measure its ability to detect sensitive attributes in datasets it has not encountered during training. These attributes correspond to the nine categories outlined in Section 7.2.3.1. The goal is to determine whether SDLog can accurately locate these sensitive attributes. Second, we evaluate SDLog's ability to classify the detected attributes into their respective sensitive categories. This two-tiered evaluation ensures that SDLog not only generalizes well to new data but also effectively distinguishes between different types of sensitive data.

7.3.2.3 Results

The order of regex patterns significantly affects performance. Table 7.4 presents the results of applying our top-performing regex patterns for detecting sensitive attributes in software logs, executed 500 times with randomized pattern orders. Despite the distinct syntactic structures among the eight sensitive categories, the performance of regex-based sensitive attribute detection varies notably based on pattern ordering. In some cases, the variation is minor; for example, the F1-score for file paths ranges from 79.3% to 80.5%, and for IP addresses from 88.5% to 88.9%. However, for URL pattern, the F1-score fluctuates drastically, from as low as 16.6% up to 95.1%, representing a 78.5% difference. This highlight

Table 7.4 Performance of regular expression pipeline in categorizing sensitive attributes on the Benchmark Dataset with randomized orders (500 runs)

		Precision (%)		**Recall (%)**		**F1 (%)**	
Attribute	**Best Regex Pattern**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
IP	(\b\d{1,3}(?:\.\d{1,3}){3}\b)	92.1	93.0	85.1	85.1	88.5	88.9
MAC	\([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})\	98.6	98.6	100.0	100.0	99.3	99.3
File path	(/|)(([\w.-]+|\<*\>)/)+([\w.-]+|\<*\>)	66.6	68.4	97.7	98.0	79.3	80.5
ID	[uU]id[:|-|=|\s/]*(\d+)	99.8	99.8	23.5	23.5	38.0	38.0
URL	[A-Za-z\.]+://[A-Za-z0-9\.\/\+#@:_\-]+ (?<![:\.])	70.6	94.6	9.4	99.2	16.6	95.1
Username	r?[uU]ser[:|-|=|\s/]*<(\w+)>|r? [uU]ser[:|-|=|\s/]*(\w+)	36.6	36.6	72.0	72.0	48.5	48.5
Port	[pP]ort[=: |:|=|: |\s/]*(\d1,5)	96.0	96.2	8.1	8.1	15.0	15.0
Configuration	size\s+(\d+)	19.2	19.2	14.2	14.2	16.3	16.3

three key observations: (1) the sequence in which regex patterns are applied has a substantial impact on overall performance, (2) the optimal ordering is likely dataset-dependent and must be tuned manually through trial and error, and (3) regex-based approaches lack robustness and reliability due to their sensitivity to manual configurations, raising concerns about their scalability and trustworthiness in real-world log datasets.

SDLog can accurately detect sensitive information in software logs, achieving an overall F1-score of 92.9%. Table 7.5 summarizes SDLog's performance on the Benchmark Dataset, reporting precision (i.e., the proportion of correctly predicted sensitive attributes among all predictions), recall (i.e., the proportion of actual sensitive attributes correctly identified), F1-score (i.e., the harmonic average of precision and recall), and support (i.e., the number of predictions per dataset). The support values vary considerably across datasets, depending on how many sensitive attributes they contain. For example, the *HealthApp* dataset contains no actual sensitive information (see Table 7.2), yet the model falsely predicted one instance as sensitive. In contrast, SDLog achieved perfect F1-scores of 100% on datasets such as *Apache* and *Proxifier*, where it successfully identified all sensitive attributes. To compute the overall performance, we aggregate true positives, false positives, and false negatives across all testing datasets from the cross-validation to build a unified confusion matrix. Overall, SDLog achieved a precision of 94.6%, recall of 91.2%, and F1-score of 92.9%.

SDLog demonstrates strong performance in categorizing most sensitive attribute types. Looking at Table 7.6, SDLog achieves high precision, recall, and F1-scores for the majority of sensitive categories. For example, it correctly detects 97.8% of net attributes and is accurate 99.5% of the time. Similarly, it identifies 94.8% of file paths with a precision of 99.9%. For more challenging attributes, such as ID, username, and configuration details, which lack consistent structure and vary significantly across datasets, SDLog still performs well, achieving F1-scores of 88.7%, 84.8%, and 50.4% respectively. Although the model achieves strong overall performance, its lower scores on MAC and URL attributes can be primarily explained by the limited availability of labeled examples: only 70 instances of MAC and 128 instances of URL are present across the entire Benchmark Dataset, with even fewer examples available during training. Moreover, these two attributes appear in just three datasets (check Table 7.2), and their values likely vary significantly between training and testing sets. This scarcity and inconsistency make it challenging for the model to learn generalizable patterns. With more diverse labeled data, we expect SDLog's performance on these attributes to improve substantially. In contrast, categories such as Username and net have significantly more training examples (e.g., 1,623 and 13,851 respectively), enabling the model to learn their patterns more effectively. Despite these limitations, SDLog proves to be a highly effective tool for detecting and categorizing sensitive attributes in logs. Future work can address lower-performing categories by incorporating additional labeled data and targeted fine-tuning, further enhancing SDLog's coverage and robustness.

SDLog's second-level granularity achieves near-perfect categorization of net sub-attributes. As presented in Table 7.7, the second, lightweight model of SDLog is dedicated to distinguishing between the sub-attributes of the net attribute, namely, IP addresses, port numbers, and host names. Trained with a high number of support for each attribute, SDLog achieves high F1-scores of 99.7% for IP addresses, 100.0% for Port numbers, and 99.6% for host names. This two-stage approach, first detecting net-related entities, then categorizing the sub-attributes of it, proves effective in handling attributes that frequently appear in similar context and overlapping or compound forms (e.g., IP:port or host:port).

SDLog significantly outperforms regular expressions in detecting sensitive attributes in software logs. A comparison between SDLog (Tables 7.6 and 7.7) and the regex-based approach (Table 7.4) reveals that SDLog achieves higher accuracy in identifying and classifying sensitive attributes. Please note that in Table 7.4, the *Max* values represent the highest performance achieved by the regular expression patterns, based on the optimal ordering and selection of the best-performing pattern for each attribute. Even under these optimal conditions, regex can detect IP addresses with 85.1% recall and 93.0% precision. In contrast, SDLog detects 97.8% of net attributes with 99.5% precision and further classifies IP addresses with 99.4% recall and 100.0% precision in the second granularity level, resulting in over a 10% improvement in F1-score. This result is especially important, as IP addresses are widely regarded as the most sensitive attribute in software logs, recognized as sensitive by academia, industry, and privacy regulations [310].

Table 7.5 SDLog performance in detecting sensitive attributes across the Benchmark Dataset

Dataset	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Support**
Android	80.5	89.9	84.9	313
Apache	100.0	100.0	100.0	1481
BGL	100.0	86.3	92.6	175
Hadoop	98.8	80.0	88.4	2082
HDFS	93.1	95.1	94.1	4417
HealthApp	0.0	0.0	0.0	1
HPC	100.0	68.8	81.5	369
Linux	99.8	99.7	99.7	3874
Mac	62.3	49.9	55.4	577
OpenSSH	92.0	91.6	91.8	5363
OpenStack	100.0	88.8	94.1	3559
Proxifier	100.0	100.0	100.0	3042
Spark	62.3	64.2	63.2	2162
Thunderbird	97.1	86.5	91.5	980
Windows	99.8	99.0	99.4	1207
Zookeeper	99.9	99.8	99.8	1271
Overall	**94.6**	**91.2**	**92.9**	**30873**

For file paths, regex performs well in recall (98.0%) but poorly in precision (68.4%), yielding a relatively modest F1-score. SDLog, by contrast, balances both dimensions effectively, achieving 99.9% precision and 94.8% recall, improving the F1-score by 16.8%. Port number detection is another area where regex underperforms, with a recall of just 8.1% and F1-score of 15.0%. On the other hand, SDLog first achieves an F1-score of 98.6% on detecting the net attribute, and then, achieves an F1-score of 100.0% on distinguishing the port number out of the net attribute. Host name detection is handled with near-perfect accuracy by SDLog, while no regex pattern was found for this attribute.

For attributes lacking a fixed format or structure, such as username, ID, and configuration details, SDLog significantly outperforms regular expressions. While regex achieves only 38.0% F1-score for IDs, 48.5% for usernames, and 16.3% for configuration details, SDLog improves these scores to 88.7%, 84.8%, and 50.4%, respectively, improving F1-score by 30%-50%. These results demonstrate that SDLog not only outperforms regex across all categories, but particularly excels where rule-based methods are most limited.

Table 7.6 SDLog performance in categorizing sensitive attributes on the Benchmark Dataset

Attribute	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Support**
Net	99.5	97.8	98.6	13851
MAC	100.0	40.0	57.1	70
File path	99.9	94.8	97.3	2868
ID	86.0	91.5	88.7	9745
URL	0.0	0.0	0.0	128
Username	99.8	73.7	84.8	1623
Configuration	95.7	34.2	50.4	1049

Table 7.7 SDLog performance in categorizing net attribute on the Net Benchmark Dataset

Attribute	**Precision (%)**	**Recall (%)**	**F1 (%)**	**Support**
IP	100.0	99.4	99.7	8922
Port	100.0	100.0	100.0	7168
Host name	100.0	99.2	99.6	6013

Summary

Our results demonstrate that SDLog outperforms regex-based approaches in detecting sensitive information in software logs. Regex performance is highly sensitive to the ordering of patterns, making it unreliable and difficult to scale, particularly for attributes with non-structured formats. In contrast, SDLog achieves robust and accurate sensitive attribute detection across diverse datasets, with an overall F1-score of 92.9%. SDLog especially excels in handling attributes without predefined structure, such as username, ID, and configuration details, highlighting its effectiveness and generalization capability in real-world log datasets.

7.3.3 RQ3. How effective is SDLog in identifying sensitive attributes when fine-tuned on a target dataset?

7.3.3.1 Motivation

While RQ2 shows that SDLog outperforms regular expressions, especially in complex or unstructured attributes, its performance can still be limited for certain categories that lack sufficient labeled examples or structural diversity in training. Our goal, however, is to develop a sensitive attribute detection pipeline that can be practically adopted by organizations, researchers, and practitioners as a reliable alternative to the commonly used regular expressions in anonymization workflows. In this research question, we explore how fine-tuning SDLog with a subset of logs from the target dataset can improve its sensitive attribute detection performance. We evaluate the impact of different amounts of fine-tuning data on both overall sensitive attribute detection performance and attribute-level categorization, using the same evaluation procedure as in RQ2.

7.3.3.2 Approach

To evaluate how fine-tuning improves SDLog's ability to detect sensitive attributes in a new dataset, we simulate a practical scenario where only a small subset of labeled log samples from the target dataset is available for adaptation. Specifically, we consider three fine-tuning sizes: 20, 50, and 100 logs. For this purpose, we begin with the base models trained during RQ2 (i.e., models trained on all datasets except the target one), and further fine-tune them using a small, labeled subset from the target dataset.

For each dataset in our Benchmark Dataset, which contains 2,000 logs, we reserve the first 100 logs that contain at least one sensitive attribute as the fine-tuning set. The remaining 1,900 logs are used as the fixed test set. For example, in RQ2, to evaluate SDLog on the *Apache* dataset, we trained the model on the other 15 datasets and tested it on all 2,000 *Apache* logs. In this research question, we extend that setup by additionally fine-tuning the pre-trained SDLog model using 20, 50, or 100 *Apache* logs before testing it, each time using the same fixed 1,900-log test set for consistency.

This fine-tuning and evaluation process is repeated for each dataset in the Benchmark Dataset. At the end of this process, we report the impact of different fine-tuning sizes (20, 50, and 100 logs) on the model's performance. We evaluate both its ability to detect the presence of sensitive information and its accuracy in categorizing the specific type of each sensitive attribute. For this RQ, we exclude the *HealthApp* dataset from our evaluation setup, as it does not contain any sensitive attributes in the entire dataset, and therefore, it cannot be used for fine-tuning SDLog.

7.3.3.3 Results

Fine-tuning SDLog with target-specific data enhances its sensitive attribute detection performance. Table 7.8 summarizes the results of sensitive attribute detection across the Benchmark Dataset when SDLog is fine-tuned using 20, 50, and 100 labeled log samples from the target dataset. In general, increasing the number of fine-tuning examples leads to improved performance. Specifically, in 13 out of 15 datasets, fine-tuning with 100 logs results in equal or higher F1-scores compared to using only 20 logs. Notably, when

Table 7.8 Performance of fine-tuned SDLog for individual target datasets (with 20, 50, and 100 logs) in detecting sensitive attributes across the Benchmark Dataset

	20			**50**			**100**		
Dataset	**P (%)**	**R (%)**	**F1 (%)**	**P (%)**	**R (%)**	**F1 (%)**	**P (%)**	**R (%)**	**F1 (%)**
Android	92.8	99.0	95.8	88.0	99.0	93.6	86.7	100.0	92.9
Apache	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
BGL	85.7	92.3	88.9	98.5	100.0	99.2	85.5	100.0	92.2
Hadoop	80.6	84.5	82.5	81.4	92.5	86.8	82.8	99.6	90.4
HDFS	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
HPC	97.1	98.1	97.6	54.3	98.1	69.9	96.3	100.0	98.1
Linux	99.9	99.6	99.8	99.5	99.6	99.6	99.6	99.6	99.6
Mac	67.0	58.5	62.5	68.3	85.6	76.0	91.1	96.2	93.6
OpenSSH	100.0	99.7	99.8	100.0	99.8	99.9	100.0	99.8	99.9
OpenStack	100.0	88.7	94.0	100.0	95.0	97.4	100.0	99.8	99.9
Proxifier	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Spark	82.6	97.9	89.6	99.7	97.9	98.8	97.0	97.9	97.5
Thunderbird	96.9	87.7	92.1	98.1	87.6	92.6	92.4	95.3	93.8
Windows	100.0	99.2	99.6	100.0	99.2	99.6	100.0	99.2	99.6
Zookeeper	85.4	100.0	92.1	91.7	99.9	95.6	88.8	99.9	94.0
Overall	**96.2**	**96.3**	**96.3**	**96.9**	**98.1**	**97.5**	**97.4**	**99.5**	**98.4**

fine-tuned with 100 samples, SDLog achieves an F1-score above 90% and a recall above 95% on all datasets. This indicates that, across the entire Benchmark Dataset, SDLog, when fine-tuned with just 100 logs, can accurately detect at least 95% of all sensitive attributes. Overall, using 100 fine-tuning logs, SDLog achieves an average precision of 97.4% (up by 1.2% compared to using 20 logs), a recall of 99.5% (up by 3.2% compared to using 20 logs), and an F1-score of 98.4% (a 2.1% improvement compared to using 20 logs).

Fine-tuning SDLog with target-specific logs significantly improves the categorization of sensitive attributes. Table 7.9 shows the categorization performance of SDLog after fine-tuning with varying amounts of labeled data from the target dataset. The results show that fine-tuning greatly enhances the model's ability to distinguish between different types of sensitive attributes. As the number of fine-tuning examples increases, from 20 to 100 logs, SDLog demonstrates consistent improvement gains across all attribute types. Even for attributes that already had high performance using 20 logs as fine-tuning, such as net, file path, ID, and username, fine-tuning with 100 logs yields additional improvements in F1-score ranging from 0.4% to 2.3%. More substantial gains are observed in challenging attributes. For instance, comparing the 20 log samples and 100 log samples for fine-tuning, the F1-score for URL increases from 76.9% to 93.4% (a 16.5% improvement), for MAC address from 54.5% to 95.9% (a 41.4% boost), and for configuration details, the most irregular and unstructured category, from 45.4% to 95.1% (a 49.7% increase). Notably, with just 20 fine-tuning logs, 4 out of 7 attributes already surpass a 90% F1-score, and when using 100 logs, all 7 attributes exceed 90% F1-score, highlighting the strong impact of even modest fine-tuning on SDLog's classification capabilities.

Summary

Our results show that fine-tuning SDLog on a small, labeled subset of the target dataset significantly improves its performance in both detecting and categorizing sensitive attributes. With as few as 100 fine-tuning samples, SDLog accurately identifies 99.5% of sensitive attributes in Benchmark Dataset and achieve an F1-score of 98.4%. Fine-tuning also enhances the model's ability to classify specific attribute types, with special improvements for more challenging categories such as configuration details. When fine-tuned with 100 samples, SDLog reaches an F1-score exceeding 90% across all categories of sensitive attributes. These findings highlight the practical value of lightweight fine-tuning in adapting SDLog to new environments and improving its effectiveness.

7.4 Discussion

7.4.1 When are regular expressions suitable for detecting sensitive attributes in software logs?

Combining results from our first two research questions, we identify several inherent limitations of regular expressions for detecting sensitive attributes in software logs. First, there is no standardized ground truth for regex patterns; for example, we found 17 distinct patterns for identifying IP addresses, a relatively well-defined format. Second, performance varies widely across patterns; some regexes for IP detection gained extremely low F1-scores (e.g., 0.0%, 1.1%, and 9.1%). Third, even small syntactic differences between similar-looking regexes can lead to large performance variations. Fourth, regexes often lack generalizability across datasets, requiring developers to craft dataset-specific patterns, which limits reuse and scalability. Fifth, regexes perform poorly on attributes that have no defined format and are context-dependent, such as username, ID, and configuration details. These attributes are often ignored or weakly captured in practice. Finally, in real-world pipelines where multiple regexes are applied sequentially, the overall performance is highly sensitive to their execution order; an order that must be optimized per dataset but is not known in advance.

Despite these limitations, regexes can still be effective in certain controlled scenarios. Our results show that when the most effective patterns are carefully selected and applied in an optimal sequence, regexes can perform well on attributes with clearly defined structures, such as MAC addresses or URLs, achieving F1-scores above 95%. However, for more ambiguous

Table 7.9 Performance of fine-tuned SDLog for individual target datasets (with 20, 50, and 100 logs) in categorizing sensitive attributes on the Benchmark Dataset

	20			**50**			**100**		
Attribute	**P (%)**	**R (%)**	**F1 (%)**	**P (%)**	**R (%)**	**F1 (%)**	**P (%)**	**R (%)**	**F1 (%)**
Net	96.5	99.5	98.0	96.1	99.8	97.9	96.9	99.9	98.4
MAC	94.7	38.3	54.5	100.0	74.5	85.4	92.2	100.0	95.9
File path	99.1	95.9	97.5	99.4	98.3	98.8	99.4	98.9	99.1
ID	95.4	99.4	97.4	96.8	99.8	98.3	97.6	99.9	98.7
URL	100.0	62.5	76.9	100.0	62.5	76.9	99.0	88.4	93.4
Username	94.2	98.0	96.1	99.8	98.0	98.9	97.5	99.4	98.4
Configuration	97.2	29.6	45.4	96.7	62.7	76.1	96.5	93.7	95.1

attributes such as username, configuration details, or ID, regexes remain unreliable even under ideal conditions. We also did not find any usable regex patterns for detecting host names.

7.4.2 What are the advantages of using a learning-based model such as SDLog for detecting sensitive attributes in software logs?

Adopting a deep learning model such as SDLog offers several key advantages over traditional regex-based methods. Unlike regexes, SDLog does not rely on hand-crafted patterns and can identify a wide range of sensitive attributes, including those that lack rigid or well-defined formats (e.g., usernames, IDs, and configuration details). Additionally, SDLog eliminates the problem of dependencies on the order of rules. Since it uses a unified model for all attribute types, it avoids conflicts and interferences that often arise when multiple regexes are applied sequentially. One of SDLog's key advantages is its generalizability: it can serve as a pre-trained model for sensitive attribute detection in software logs, eliminating the need for developers to manually design and validate regex patterns for every new environment. Furthermore, SDLog benefits from contextual understanding, allowing it to detect sensitive information based not just on format, but also on surrounding semantics, something regex patterns are inherently incapable of. In summary, SDLog offers a more robust, flexible, and scalable solution for sensitive data detection in software logs.

7.4.3 How easy is it to integrate SDLog into real-world log anonymization pipelines?

As shown in the results of RQ2, even the plain SDLog model outperforms the best-performing regular expressions. Integrating SDLog into an anonymization pipeline is straightforward. SDLog, trained on all 16 datasets of the Benchmark Dataset, is publicly available on Hugging

Face [315,316], and can be used like any standard pre-trained language model such as BERT. After formatting the logs appropriately, developers can run SDLog with a simple API call. Moreover, RQ3 shows that fine-tuning SDLog on as few as 20 to 100 labeled log lines from the target dataset significantly boosts performance. Based on our experience, labeling 100 logs typically takes only 2–3 days, and fine-tuning the model takes about a few minutes on a standard PC with a consumer-grade graphics card. To support adoption, we also provide the scripts needed to prepare the raw dataset and fine-tune the SDLog. By releasing our models and tools, we aim to make SDLog an accessible and practical alternative to regex-based methods in real-world anonymization pipelines.

7.5 Threats to Validity

Internal validity. The definition of *sensitive* information may vary across organizations, individuals, and legal jurisdictions. To mitigate this, we adopted the definition proposed in Chapter 6, which combines perspectives on sensitivity from academic literature, industrial practice, and privacy regulations.

External validity. Although we evaluated SDLog on 16 publicly available log datasets that have been widely used in prior research on software logs [335–337], the results may not generalize to all logging environments, such as those used in enterprise or government environments. Nonetheless, we demonstrated that fine-tuning SDLog on a small portion of a new dataset leads to substantial performance gains, suggesting the method is adaptable to unseen domains.

Additionally, SDLog may be less effective at detecting sensitive attributes that were not present in the training data, such as geolocation coordinates or API keys. However, our fine-tuning experiments show that even attributes that were underrepresented during pre-training can be accurately detected after domain-specific fine-tuning.

Construct validity. The ground truth annotations were created manually by the first two authors. They may include subjective judgment regarding what constitutes sensitive information. To mitigate potential bias, we followed a systematic annotation approach and measured inter-rater agreement, achieving a high reliability after two rounds of independent coding followed by discussion sessions.

Another potential threat lies in the selection of regex patterns, which may not reflect the most effective rules an experienced practitioner could design. To mitigate this, we systematically collected patterns from three academic databases and integrated regexes from three industry collaborators. We found a total of 41 regex patterns across eight sensitive attribute types and benchmarked only the top-performing patterns in our comparisons.

There is also a risk of overfitting, particularly when SDLog is fine-tuned on small datasets, potentially causing it to learn dataset-specific artifacts. We addressed this by employing an early stopping strategy during training to prevent overfitting.

7.6 Chapter Summary

We introduce SDLog, a deep learning-based framework for detecting sensitive attributes in software logs, a critical first step for log anonymization. While traditional approaches rely heavily on manually designed regular expressions, these methods suffer from limited generalizability and poor performance on unstructured or context-dependent attributes. SDLog overcomes these limitations by leveraging contextual understanding through pre-trained language models, enabling it to generalize across diverse log formats and structures. In our evaluation across 16 datasets, SDLog consistently outperforms the best-performing regex patterns in identifying sensitive information. When fine-tuned with as few as 100 samples from a target dataset, SDLog achieves near-perfect detection, correctly identifying 99.5% of sensitive attributes. Our approach could easily be integrated into real-world log anonymization pipelines.

CHAPTER 8 CONCLUSION

This chapter provides a summary and conclusion of the thesis, addressing its limitations and proposing directions for future research.

8.1 Summary

In recent years, the concept of AIOps has gained attention as a promising and transformative approach for addressing the growing complexity and scale of modern IT environments. AIOps leverages big data technologies and AI techniques to automate and improve various areas of IT Ops. Similar to any AI system, the effectiveness of AIOps solutions heavily depends on access to high-quality, relevant, and diverse data. However, there is a notable shortage of public and accessible AIOps datasets. This thesis aims to address the shortage problem of publicly available data in the field of AIOps.

Chapter 4 presents our study on open-source AIOps projects. The main goal of this study was to assess if the lack of public AIOps datasets is a limitation for real-world AIOps projects. We combine both quantitative and qualitative analyses to understand the state of AIOps solutions. We compare the state of AIOps projects with two set of baselines, showing that AIOps projects are receiving a lot of attention in open-source community and are growing fast. We also determine the most common input data, techniques, and goals of AIOps pipelines. We also observe that the shortage of public and accessible AIOps datasets is indeed a challenge for open-source AIOps projects.

To mitigate the shortage of AIOps datasets, generally two main approaches exist: (1) Generating synthetic data by simulating real-world scenarios, or (2) Acquiring real-world operational data from IT organizations.

In Chapter 5, we focus on the first approach, generating synthetic data. To generate synthetic data, researchers and practitioners try to replicate real-world scenarios and behaviors. Typically, they begin by analyzing real datasets to uncover different patterns and then apply various models and techniques to produce new, synthetic data. One of the main sources of real-world data that is used for this purpose is web application workloads. Therefore, in Chapter 5, we conduct a study to first identify public web application datasets, then examine the applications and trends of web application workloads in the research community, and finally, characterize these workloads. We uncover three daily and three weekly patterns that are common in public web application workloads. Since these patterns and insights were derived from real-world datasets, they can be used in realistic workload generation by replicating the behavioral trends observed in actual operational data. Future work can leverage these findings to develop realistic workload generation pipelines.

While synthetic data has its benefits and applications, it comes with many limitations such as oversimplified view of complex systems, sophistication of synthetic data generation approaches, and the tendency of stakeholders to real-world data due to its direct reflection of true operational contexts. On the other hand, while real-world data has numerous benefits over synthetic data, it comes with privacy and security risks. Presence of PII and quasi-identifiers in IT Ops data is a significant concern for managing and sharing this data. To protect sensitive information within IT environments, anonymizing is a crucial step to prevent unauthorized access to user, system, or infrastructure details. However, anonymizing IT Ops data comes with three major challenges: (1) Understanding the sensitive information that needs to be anonymized, (2) Detecting sensitive information within vast amount of IT data, and (3) Effectively anonymizing sensitive attributes while handling the utility-privacy trade-off. To overcome these challenges, we perform two studies, one to define the sensitive attributes in software logs, and another to design a framework to detect sensitive information in software logs. We focus on software logs because as discovered in Chapter 4, the most common data type in AIOps techniques is software logs.

In Chapter 6, we perform an analysis to understand the sensitive information of software logs from different perspectives of regulatory frameworks, research literature, and industry practices. By combining the results from these perspectives, we come up with nine attributes, including IP addresses, MAC addresses, and configuration details, that are generally sensitive in software logs. Building upon this definition, in Chapter 7, we introduce SDLog, a deep learning-based framework for detecting sensitive attributes in software logs. SDLog is a replacement for regular expressions. Traditionally, almost all anonymization techniques rely on regular expressions to detect sensitive information. However, regular expressions come with many challenges and limitations such as limited generalizability and poor performance on unstructured or context-dependent attributes. SDLog, on the other hand, leverages contextual understanding to detect sensitive information and is able to generalize across diverse log formats and structures. Our results show that SDLog consistently outperforms the best-performing regular expression patterns. When fine-tuned with as few as 100 samples from a target dataset, SDLog achieves near-perfect detection, correctly identifying 99.5% of sensitive attributes. All materials related to our studies, including code, data, and models, have been made publicly available so that future work can use, replicate, or extend them.

8.2 Key Findings

The main objective of this thesis was to address the shortage of publicly available datasets for AIOps by contributing to both synthetic data generation and privacy-preserving data sharing. In order to achieve this objective, we performed several studies. In this section, the key findings of this thesis are summarized.

- **AIOps is an emerging field and is growing rapidly.** Our results show that the speed of growth for AIOps applications is faster than that for ML and General projects on GitHub. Besides, open-source AIOps projects are receiving more attention and are also more popular and active than the baselines.
- **Software logs are the most common data type in AIOps pipelines.** By analyzing open-source AIOps projects, we find that software logs are the most commonly used input data, followed by performance metrics and network-traffic data. We also find that classical machine learning techniques are the most used analysis techniques, followed (far behind) by deep learning and statistical analysis. Also, the most popular goal of the AIOps projects are anomaly detection, followed (far behind) by monitoring.
- **The shortage of public and accessible AIOps datasets is indeed a challenge for AIOps solutions.** The results from two of our studies confirm that both open-source AIOps projects (Chapter 4) and AIOps literature (Chapter 5) suffer from lack of public and accessible AIOps datasets. They both consistently use a limited number of datasets that are often restricted in size, scope, complexity, or are outdated.
- **Current synthetic data generation approaches use simplistic modeling which does not capture real-world workload patterns.** Our analysis of 12 real-world web application workloads reveal three daily and three weekly patterns that are consistent across all the datasets. For the task of synthetic data generation, existing studies commonly use simplistic patterns such as random, steady, or linear [205,248]. However, we find that real-world patterns are more complex, follow polynomial models and are non-monotonic and non-linear.
- **There is no universally established definition of sensitivity in software logs.** By analyzing privacy regulations, research literature, and industry practices, we find that definitions of sensitivity in software logs vary, and there is currently no established definition of which sensitive attributes should be anonymized prior to dataset sharing. Based on our analyses of software log privacy from multiple perspectives, we consider

IP addresses, MAC addresses, host names, file paths, IDs, URLs, usernames, port numbers, and configuration details as generally sensitive information in software logs.

- **Regular expressions have many limitations in detecting sensitive information.** Identifying sensitive information is the first step in the anonymization process. Although regular expressions are widely used in both academia and industry for this purpose, they have several limitations. Our results reveal that regex patterns are manual-intensive, often lack generalizability across datasets, and are not well-suited for attributes without strict formatting.
- **A learning-based approach, SDLog, can identify up to 99.5% of sensitive information in software logs.** We introduce SDLog, a deep-learning framework for detecting sensitive information in software logs. Our results show that SDLog consistently outperforms regex-based approaches in detecting sensitive information, achieving robust and accurate sensitive attribute detection across diverse datasets, with an overall F1-score of 92.9%. When fine-tuned with as few as 100 samples from the target dataset, SDLog accurately identifies 99.5% of sensitive attributes and achieve an F1-score of 98.4%.

8.3 Limitations

While this thesis aims to address the shortage problem of publicly available data in the field of AIOps, it comes with a set of limitations.

Labeling and categorization: In three out of our four studies, we needed to categorize and label projects, articles, and log attributes. This manually labeling process could introduce potential bias to the results of our studies. To mitigate this risk, we always involved at least two coders in the process, followed a systematic methodology, and measured inter-rater agreement between coders. For all the studies, we achieved a high reliability ratio, indicating the categorizations were not produced by one person; rather, they reflect a consistent and shared understanding among multiple independent coders.

Generalizability: A key limitation of our studies lies in the reliance on open-source projects and publicly available datasets. These resources do not necessarily reflect the full spectrum of AIOps use cases, particularly those involving proprietary datasets or industrial AIOps pipelines. As a results, our findings may not generalize to all real-world scenarios, especially in enterprise environments where operational data is often confidential or significantly larger in scale. To mitigate this limitation, we tried to include a diverse set of datasets and projects that approximate real-world conditions.

Data collection: Another limitation across our studies relates to potential biases introduced during data selection and the definitions adopted throughout the analysis. Relying on specific keywords to find AIOps projects or to find related articles may lead to the deletion of relevant data sources which could impact our findings. Additionally, foundational concepts such as *sensitivity* in data attributes may carry different meanings across industries, regions, and individuals. To mitigate these risks, we always performed systematic searches and filtering procedures and clearly documented the criteria and definitions used in each study to ensure transparency and reproducibility. We have also shared all the materials of our studies for future validation or extension.

8.4 Future Research

This thesis is a first step towards a better collaboration between academia and industry to create, design, and share AIOps datasets that can be used by researchers in the emerging field of AIOps. The findings of this thesis open a wide range of opportunities for future research.

- Most of our studies focus on open-source AIOps projects and public AIOps datasets to perform different analyses and conclude different findings. However, differences may exist between open-source and publicly available datasets or pipelines and the proprietary solutions developed within companies. Thus, an important extension of this thesis is to investigate AIOps pipelines developed within companies through potential collaborations with industry partners.

- In the study presented in Chapter 5, we analyzed daily and weekly patterns across public web application workloads. An extension of this study could be the possibility of exploring different temporal granuralities, for example seasonal or yearly patterns.

- In Chapters 6 and 7, we focused on privacy in software logs, as software logs are the most common data types in AIOps pipelines. Future work can extend our results to include other types of AIOps data, including software traces or network-related data.

- In order to share a sensitive dataset, three steps of understanding the sensitive attributes, detecting the sensitive attributes, and anonymizing the detected attributes are needed. While we focused on the first two steps, future work could work on the third aspect and develop an anonymization tool that is specific to software logs, handling the utility-privacy trade-off.

- There is an opportunity to develop a privacy score that could evaluate the sensitivity of log entries based on their attributes and determine if sharing them would pose a privacy risk.
- Future work can further enhance SDLog's capabilities by incorporating additional log datasets and increasing coverage for underrepresented attribute types.
- Another possible direction for future research is the development of standardized benchmarks for AIOps datasets. Establishing such benchmarks would support fair comparison of AIOps techniques and promote reproducibility across the community.
- Future work could also explore multi-modal synthetic data generation by developing models that simultaneously simulate multiple types of AIOps data (e.g., logs, metrics, and traces), capturing correlations and dependencies across different operational dimensions.
- Although this thesis focuses on software systems, the approaches proposed for synthetic data generation and privacy-preserving data sharing can be extended to other domains such as edge computing, IoT, and cybersecurity, where similar challenges in telemetry data quality and privacy exist.

REFERENCES

[1] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1245–1255, 2013.

[2] S. Nedelkoski, J. Bogatinovski, A. K. Mandapati, S. Becker, J. Cardoso, and O. Kao, "Multi-source distributed system data for ai-powered analytics," in *Service-Oriented and Cloud Computing: 8th IFIP WG 2.14 European Conference, ESOCC 2020, Heraklion, Crete, Greece, September 28–30, 2020, Proceedings 8*. Springer, 2020, pp. 161–176.

[3] Y. Remil, A. Bendimerad, R. Mathonat, and M. Kaytoue, "Aiops solutions for incident management: Technical guidelines and a comprehensive literature review," *arXiv preprint arXiv:2404.01363*, 2024.

[4] Q. Cheng, D. Sahoo, A. Saha, W. Yang, C. Liu, G. Woo, M. Singh, S. Saverese, and S. C. Hoi, "Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges," *arXiv preprint arXiv:2304.04661*, 2023.

[5] A. Dakkak, J. Bosch, and H. Holmstrom Olsson, "Towards aiops enabled services in continuously evolving software-intensive embedded systems," *Journal of Software: Evolution and Process*, vol. 36, no. 5, p. e2592, 2024.

[6] S. Kabinna, C.-P. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan, "Examining the stability of logging statements," *Empirical Software Engineering*, vol. 23, pp. 290–333, 2018.

[7] Y. Dang, Q. Lin, and P. Huang, "Aiops: real-world challenges and research innovations," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 4–5.

[8] P. Prasad and C. Rich, "Market Guide for AIOps Platforms," https://www.gartner.com/en/documents/3892967, 2018, last accessed: 2025/06/20.

[9] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in *Proceedings of the first workshop on machine learning for computing systems*, 2018, pp. 1–8.

[10] S. Nedelkoski, J. Cardoso, and O. Kao, “Anomaly detection from system tracing data using multimodal deep learning,” in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD).* IEEE, 2019, pp. 179–186.

[11] Y. Li, Z. M. Jiang, H. Li, A. E. Hassan, C. He, R. Huang, Z. Zeng, M. Wang, and P. Chen, “Predicting node failures in an ultra-large-scale cloud computing platform: an aiops solution,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 2, pp. 1–24, 2020.

[12] G. Zhao, S. Hassan, Y. Zou, D. Truong, and T. Corbin, “Predicting performance anomalies in software systems at run-time,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–33, 2021.

[13] J. Ding, X. Wang, X. Ye, Y. Ouyang, and Y. Chai, “Ttercl: An onsite real-time alarm root-cause location algorithm,” in *2021 IEEE International Conference on Big Data (Big Data).* IEEE, 2021, pp. 4852–4858.

[14] H. Wang, Z. Wu, H. Jiang, Y. Huang, J. Wang, S. Kopru, and T. Xie, “Groot: An event-graph-based approach for root cause analysis in industrial settings,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 2021, pp. 419–429.

[15] J.-G. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, “Experience report on applying software analytics in incident management of online service,” *Automated Software Engineering*, vol. 24, no. 4, pp. 905–941, 2017.

[16] R. Ding, Q. Fu, J. G. Lou, Q. Lin, D. Zhang, and T. Xie, “Mining historical issue repositories to heal large-scale online service systems,” in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.* IEEE, 2014, pp. 311–322.

[17] P. Notaro, J. Cardoso, and M. Gerndt, “A survey of aiops methods for failure management,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 12, no. 6, pp. 1–45, 2021.

[18] J. Bogatinovski, S. Nedelkoski, A. Acker, F. Schmidt, T. Wittkopp, S. Becker, J. Cardoso, and O. Kao, “Artificial intelligence for it operations (aiops) workshop white paper,” *arXiv preprint arXiv:2101.06054*, 2021.

[19] J. Diaz-De-Arcaya, A. I. Torre-Bastida, G. Zárate, R. Miñón, and A. Almeida, “A joint study of the challenges, opportunities, and roadmap of mlops and aiops: A systematic survey,” *ACM Computing Surveys*, vol. 56, no. 4, pp. 1–30, 2023.

[20] A. Lyons, J. Gamba, A. Shawaga, J. Reardon, J. Tapiador, S. Egelman, and N. Vallina-Rodríguez, “Log:{It’s} big,{It’s} heavy,{It’s} filled with personal data! measuring the logging of sensitive information in the android ecosystem,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 2115–2132.

[21] Z. Li, N. Zhao, S. Zhang, Y. Sun, P. Chen, X. Wen, M. Ma, and D. Pei, “Constructing large-scale real-world benchmark datasets for aiops,” *arXiv preprint arXiv:2208.03938*, 2022.

[22] S. Ahmed, M. Singh, B. Doherty, E. Ramlan, K. Harkin, and D. Coyle, “Ai for information technology operation (aiops): A review of it incident risk prediction,” in *2022 9th International Conference on Soft Computing & Machine Intelligence (ISCMI)*. IEEE, 2022, pp. 253–257.

[23] L. Rijal, R. Colomo-Palacios, and M. Sánchez-Gordón, “Aiops: A multivocal literature review,” *Artificial Intelligence for Cloud and Edge Computing*, pp. 31–50, 2022.

[24] X. Wu, H. Li, and F. Khomh, “What information contributes to log-based anomaly detection? insights from a configurable transformer-based approach,” *Automated Software Engineering*, vol. 32, no. 2, pp. 1–29, 2025.

[25] N. Laptev, S. Amizadeh, and I. Flint, “Generic and scalable framework for automated time-series anomaly detection,” in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 1939–1947.

[26] L. B. N. Laboratory, “The internet traffic archive,” 1998, accessed on May 5, 2025. [Online]. Available: https://ita.ee.lbl.gov/html/traces.html

[27] S. Koussouris, T. Dalamagas, P. Figueiras, G. Pallis, N. Bountouni, V. Gkolemis, K. Perakis, D. Bibikas, and C. Agostinho, “Bridging data and aiops for future ai advancements with human-in-the-loop. the ai-dapt concept,” in *2024 IEEE International Conference on Engineering, Technology, and Innovation (ICE/ITMC)*. IEEE, 2024, pp. 1–8.

[28] G. Steinbuss and K. Böhm, “Benchmarking unsupervised outlier detection with realistic synthetic data,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 4, pp. 1–20, 2021.

[29] S. James, C. Harbron, J. Branson, and M. Sundler, “Synthetic data use: exploring use cases to optimise data utility,” *Discover Artificial Intelligence*, vol. 1, no. 1, p. 15, 2021.

[30] J. Jordon, L. Szpruch, F. Houssiau, M. Bottarelli, G. Cherubin, C. Maple, S. N. Cohen, and A. Weller, “Synthetic data–what, why and how?” *arXiv preprint arXiv:2205.03257*, 2022.

[31] S. Petrescu, F. Den Hengst, A. Uta, and J. S. Rellermeyer, “Log parsing evaluation in the era of modern software systems,” in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 379–390.

[32] D. Viana, R. Teixeira, J. Baptista, and T. Pinto, “Synthetic data generation models for time series: A literature review,” in *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET*. IEEE, 2024, pp. 1–6.

[33] S. Hao, W. Han, T. Jiang, Y. Li, H. Wu, C. Zhong, Z. Zhou, and H. Tang, “Synthetic data in ai: Challenges, applications, and ethical implications,” *arXiv preprint arXiv:2401.01629*, 2024.

[34] D. Rankin, M. Black, R. Bond, J. Wallace, M. Mulvenna, G. Epelde *et al.*, “Reliability of supervised machine learning using synthetic data in health care: model to preserve privacy for data sharing,” *JMIR medical informatics*, vol. 8, no. 7, p. e18910, 2020.

[35] Y. Fu, M. Yan, J. Xu, J. Li, Z. Liu, X. Zhang, and D. Yang, “Investigating and improving log parsing in practice,” in *Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering*, 2022, pp. 1566–1577.

[36] A. Figueira and B. Vaz, “Survey on synthetic data generation, evaluation methods and gans,” *Mathematics*, vol. 10, no. 15, p. 2733, 2022.

[37] P. A. Osorio-Marulanda, G. Epelde, M. Hernandez, I. Isasa, N. M. Reyes, and A. B. Iraola, “Privacy mechanisms and evaluation metrics for synthetic data generation: A systematic review,” *IEEE Access*, 2024.

[38] A. Varanda, L. Santos, R. L. d. C. Costa, A. Oliveira, and C. Rabadão, “Log pseudonymization: Privacy maintenance in practice,” *Journal of Information Security and Applications*, vol. 63, p. 103021, 2021.

[39] E. Union, “General data protection regulation,” 2022, accessed on May 5, 2025. [Online]. Available: https://gdpr-info.eu/

[40] U. D. of Health and H. Services, “Health insurance portability and accountability act,” 1996, accessed on May 5, 2025. [Online]. Available: https://www.hhs.gov/hipaa/index.html

[41] U. S. of California, “California consumer privacy act,” 2018, accessed on May 5, 2025. [Online]. Available: https://oag.ca.gov/privacy/ccpa

[42] T. Brekne, A. Årnes, and A. Øslebø, “Anonymization of ip traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies,” in *International Workshop on Privacy Enhancing Technologies*. Springer, 2005, pp. 179–196.

[43] T. Brekne and A. Årnes, “Circumventing ip-address pseudonymization.” in *Communications and Computer Networks*, 2005, pp. 43–48.

[44] L. Sweeney, “k-anonymity: A model for protecting privacy,” *International journal of uncertainty, fuzziness and knowledge-based systems*, vol. 10, no. 05, pp. 557–570, 2002.

[45] A. Aleroud, F. Yang, S. C. Pallaprolu, Z. Chen, and G. Karabatis, “Anonymization of network traces data through condensation-based differential privacy,” *Digital Threats: Research and Practice (DTRAP)*, vol. 2, no. 4, pp. 1–23, 2021.

[46] Gartner, “It operations,” https://www.gartner.com/en/information-technology/glossary/it-operations, 2023, last accessed: 2025/06/20.

[47] J. Iden and T. R. Eikebrokk, “Implementing it service management: A systematic literature review,” *International Journal of Information Management*, vol. 33, no. 3, pp. 512–523, 2013.

[48] P. Patel, A. H. Ranabahu, and A. P. Sheth, “Service level agreement in cloud computing,” 2009.

[49] A. Levin, S. Garion, E. K. Kolodner, D. H. Lorenz, K. Barabash, M. Kugler, and N. McShane, “Aiops for a cloud object storage service,” in *2019 IEEE international congress on big data (BigDataCongress)*. IEEE, 2019, pp. 165–169.

[50] R. McCreadie, J. Soldatos, J. Fuerst, M. F. Argerich, G. Kousiouris, J.-D. Totow, A. C. Nieto, B. Q. Navidad, D. Kyriazis, C. Macdonald *et al.*, “Leveraging data-driven infrastructure management to facilitate aiops for big data applications and operations,” in *Technologies and Applications for Big Data Value*. Springer, 2021, pp. 135–158.

[51] A. Gulenko, A. Acker, O. Kao, and F. Liu, “Ai-governance and levels of automation for aiops-supported system administration,” in *2020 29th International Conference on Computer Communications and Networks (ICCCN).* IEEE, 2020, pp. 1–6.

[52] Y. Lyu, G. K. Rajbahadur, D. Lin, B. Chen, and Z. M. Jiang, “Towards a consistent interpretation of aiops models,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 1, pp. 1–38, 2021.

[53] Q. Wong, “Facebook’s massive outage costs the company an estimated $60 million in revenue,” https://www.cnet.com/tech/mobile/facebooks-massive-outage-costs-the-company-an-estimated-60-million-in-revenue/, 2021, last accessed: 2025/06/20.

[54] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, “Metric selection and anomaly detection for cloud operations using log and metric correlation analysis,” *Journal of Systems and Software*, vol. 137, pp. 531–549, 2018.

[55] B. T. Sloss, M. Dahlin, V. Rau, and B. Beyer, “The calculus of service availability: You’re only as available as the sum of your dependencies.” *Queue*, vol. 15, no. 2, pp. 49–67, 2017.

[56] D. Galin, *Software quality assurance: from theory to implementation.* Pearson education, 2004.

[57] M. R. Lyu, “Software reliability engineering: A roadmap,” in *Future of Software Engineering (FOSE’07).* IEEE, 2007, pp. 153–170.

[58] R. Hat, “What is sre (site reliability engineering)?” https://www.redhat.com/en/topics/devops/what-is-sre, 2020, last accessed: 2025/06/20.

[59] J. Hallur, “The future of sre: Trends, tools, and techniques for the next decade,” *International Journal of Science and Research (IJSR)*, vol. 13, no. 9, pp. 1688–1698, 2024.

[60] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955,” *AI magazine*, vol. 27, no. 4, pp. 12–12, 2006.

[61] P. McCorduck and C. Cfe, *Machines who think: A personal inquiry into the history and prospects of artificial intelligence.* AK Peters/CRC Press, 2004.

[62] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, “Deep blue,” *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[63] J. McCarthy *et al.*, “What is artificial intelligence,” 2007.

[64] N. J. Nilsson, *The quest for artificial intelligence.* Cambridge University Press, 2009.

[65] T. M. Mitchell and T. M. Mitchell, *Machine learning.* McGraw-hill New York, 1997, vol. 1, no. 9.

[66] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[67] V. Nasteski, “An overview of the supervised machine learning methods,” *Horizons. b*, vol. 4, no. 51-62, p. 56, 2017.

[68] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[69] N. Patwardhan, S. Marrone, and C. Sansone, “Transformers in the real world: A survey on nlp applications,” *Information*, vol. 14, no. 4, p. 242, 2023.

[70] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[71] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[72] J. R. Jim, M. A. R. Talukder, P. Malakar, M. M. Kabir, K. Nur, and M. F. Mridha, “Recent advancements and challenges of nlp-based sentiment analysis: A state-of-the-art review,” *Natural Language Processing Journal*, p. 100059, 2024.

[73] H. Wang, H. Wu, Z. He, L. Huang, and K. W. Church, “Progress in machine translation,” *Engineering*, vol. 18, pp. 143–153, 2022.

[74] K. Nassiri and M. Akhloufi, “Transformer models used for text-based question answering systems,” *Applied Intelligence*, vol. 53, no. 9, pp. 10 602–10 635, 2023.

[75] S. Shen, J. Zhang, D. Huang, and J. Xiao, “Evolving from traditional systems to aiops: design, implementation and measurements,” in *2020 IEEE International Conference*

on Advances in Electrical Engineering and Computer Applications (AEECA). IEEE, 2020, pp. 276–280.

[76] Y. Lyu, H. Li, M. Sayagh, Z. M. Jiang, and A. E. Hassan, "An empirical study of the impact of data splitting decisions on the performance of aiops solutions," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–38, 2021.

[77] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao *et al.*, "Predicting node failure in cloud service systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 480–490.

[78] B. A. Brahmandam, "Beyond devops: The evolution toward intelligent it operations with aiops and mlops," 2025.

[79] P. Notaro, J. Cardoso, and M. Gerndt, "A systematic mapping study in aiops," in *Service-Oriented Computing–ICSOC 2020 Workshops: AIOps, CFTIC, STRAPS, AI-PA, AI-IOTS, and Satellite Events, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings.* Springer, 2021, pp. 110–123.

[80] Z. Zhong, Q. Fan, J. Zhang, M. Ma, S. Zhang, Y. Sun, Q. Lin, Y. Zhang, and D. Pei, "A survey of time series anomaly detection methods in the aiops domain," *arXiv preprint arXiv:2308.00393*, 2023.

[81] J. Xue, R. Birke, L. Y. Chen, and E. Smirni, "Managing data center tickets: Prediction and active sizing," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* IEEE, 2016, pp. 335–346.

[82] ——, "Spatial–temporal prediction models for active ticket managing in data centers," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 39–52, 2018.

[83] J.-G. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, "Software analytics for incident management of online services: An experience report," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 2013, pp. 475–485.

[84] C. Luo, J.-G. Lou, Q. Lin, Q. Fu, R. Ding, D. Zhang, and Z. Wang, "Correlating events with time series for incident diagnosis," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1583–1592.

[85] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 ninth IEEE international conference on data mining.* IEEE, 2009, pp. 149–158.

[86] B. Sharma, P. Jayachandran, A. Verma, and C. R. Das, "Cloudpd: Problem determination and diagnosis in shared dynamic clouds," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* IEEE, 2013, pp. 1–12.

[87] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng, "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *Proceedings of the 2015 internet measurement conference*, 2015, pp. 211–224.

[88] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[89] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[90] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy, "Inferring models of concurrent systems from logs of their behavior with csight," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 468–479.

[91] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2828–2837.

[92] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *ACM SIGCOMM computer communication review*, vol. 34, no. 4, pp. 219–230, 2004.

[93] ——, "Mining anomalies using traffic feature distributions," *ACM SIGCOMM computer communication review*, vol. 35, no. 4, pp. 217–228, 2005.

[94] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection and classification using distributed tracing and deep learning," in *2019 19th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGRID).* IEEE, 2019, pp. 241–250.

[95] G. Nguyen, S. Dlugolinsky, V. Tran, and Á. López García, "Network security aiops for online stream data monitoring," *Neural Computing and Applications*, vol. 36, no. 24, pp. 14 925–14 949, 2024.

[96] Y. Zhang, Z. Guan, H. Qian, L. Xu, H. Liu, Q. Wen, L. Sun, J. Jiang, L. Fan, and M. Ke, "Cloudrca: a root cause analysis framework for cloud computing platforms," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 4373–4382.

[97] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu *et al.*, "Towards intelligent incident management: why we need it and how we make it," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1487–1497.

[98] A. Saha and S. C. Hoi, "Mining root cause knowledge from cloud service incident investigations for aiops," in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, 2022, pp. 197–206.

[99] Y. Li, X. Zhang, S. He, Z. Chen, Y. Kang, J. Liu, L. Li, Y. Dang, F. Gao, Z. Xu *et al.*, "An intelligent framework for timely, accurate, and comprehensive cloud incident detection," *ACM SIGOPS Operating Systems Review*, vol. 56, no. 1, pp. 1–7, 2022.

[100] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark," in *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*. IEEE, 2015, pp. 38–44.

[101] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A large collection of system log datasets for ai-driven log analytics," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 355–366.

[102] P. Balasubramanian, J. Seby, and P. Kostakos, "Transformer-based llms in cybersecurity: An in-depth study on log anomaly detection and conversational defense mechanisms," in *2023 IEEE International Conference on Big Data (BigData)*. IEEE, 2023, pp. 3590–3599.

[103] D. Pathak, M. Verma, A. Chakraborty, and H. Kumar, "Self adjusting log observability for cloud native applications," in *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*. IEEE, 2024, pp. 482–493.

[104] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the tenth european conference on computer systems*, 2015, pp. 1–17.

[105] G. Inc., "Google traces," 2025, accessed on May 5, 2025. [Online]. Available: https://github.com/google/cluster-data

[106] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 153–167.

[107] M. Inc., "Azure traces," 2025, accessed on May 5, 2025. [Online]. Available: https://github.com/Azure/AzurePublicDataset

[108] A. Inc., "Alibaba traces," 2025, accessed on May 5, 2025. [Online]. Available: https://github.com/alibaba/clusterdata

[109] F. Li and B. Hu, "Deepjs: Job scheduling based on deep reinforcement learning in cloud data center," in *Proceedings of the 4th International Conference on Big Data and Computing*, 2019, pp. 48–53.

[110] L. Sliwko, "Cluster workload allocation: A predictive approach leveraging machine learning efficiency," *IEEE Access*, 2024.

[111] A.-I. Tuns and A. Spătaru, "Cloud service failure prediction on google's borg cluster traces using traditional machine learning," in *2023 25th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2023, pp. 162–169.

[112] U. of California, "Kdd cup 1999 data," 1999, accessed on May 5, 2025. [Online]. Available: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[113] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

[114] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2003, pp. 220–237.

[115] A. Vasudevan, E. Harshini, and S. Selvakumar, "Ssenet-2011: a network intrusion detection system dataset and its comparison with kdd cup 99 dataset," in *2011 second asian himalayas international conference on internet (AH-ICI).* IEEE, 2011, pp. 1–5.

[116] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS).* IEEE, 2015, pp. 1–6.

[117] T. M. Corporation, "Cve program," 2025, accessed on May 5, 2025. [Online]. Available: https://www.cve.org/

[118] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, no. 2018, pp. 108–116, 2018.

[119] H. Murtaza, M. Ahmed, N. F. Khan, G. Murtaza, S. Zafar, and A. Bano, "Synthetic data generation: State of the art in health care domain," *Computer Science Review*, vol. 48, p. 100546, 2023.

[120] A. Bauer, S. Trapp, M. Stenger, R. Leppich, S. Kounev, M. Leznik, K. Chard, and I. Foster, "Comprehensive exploration of synthetic data generation: A survey," *arXiv preprint arXiv:2401.02524*, 2024.

[121] Y. Lu, M. Shen, H. Wang, X. Wang, C. van Rechem, T. Fu, and W. Wei, "Machine learning for synthetic data generation: a review," *arXiv preprint arXiv:2302.04062*, 2023.

[122] F. K. Dankar and M. Ibrahim, "Fake it till you make it: Guidelines for effective synthetic data generation," *Applied Sciences*, vol. 11, no. 5, p. 2158, 2021.

[123] M. Hernandez, G. Epelde, A. Alberdi, R. Cilla, and D. Rankin, "Synthetic data generation for tabular health records: A systematic review," *Neurocomputing*, vol. 493, pp. 28–45, 2022.

[124] S. Mascaro, A. E. Nicholso, and K. B. Korb, "Anomaly detection in vessel tracks using bayesian networks," *International Journal of Approximate Reasoning*, vol. 55, no. 1, pp. 84–98, 2014.

[125] A. Haque, A. DeLucia, and E. Baseman, "Markov chain modeling for anomaly detection in high performance computing system logs," in *Proceedings of the Fourth International Workshop on HPC User Support Tools*, 2017, pp. 1–8.

[126] T. Liu, J. Tang, G. Vietri, and S. Wu, "Generating private synthetic data with genetic algorithms," in *International Conference on Machine Learning*. PMLR, 2023, pp. 22 009–22 027.

[127] Z. Wan, Y. Zhang, and H. He, "Variational autoencoder based synthetic data generation for imbalanced learning," in *2017 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2017, pp. 1–7.

[128] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[129] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang, "Diffusion models: A comprehensive survey of methods and applications," *ACM Computing Surveys*, vol. 56, no. 4, pp. 1–39, 2023.

[130] B. Kaabachi, J. Despraz, T. Meurers, K. Otte, M. Halilovic, B. Kulynych, F. Prasser, and J. L. Raisaro, "A scoping review of privacy and utility metrics in medical synthetic data," *NPJ digital medicine*, vol. 8, no. 1, p. 60, 2025.

[131] A. Goncalves, P. Ray, B. Soper, J. Stevens, L. Coyle, and A. P. Sales, "Generation and evaluation of synthetic patient data," *BMC medical research methodology*, vol. 20, pp. 1–40, 2020.

[132] N. P. C. of the People's Republic of China, "Personal information protection law," 2021, accessed on May 5, 2025. [Online]. Available: https://personalinformationprotectionlaw.com/

[133] ——, "Data security law," 2021, accessed on May 5, 2025. [Online]. Available: https://digichina.stanford.edu/work/translation-data-security-law-of-the-peoples-republic-of-china/

[134] P. C. of Canada, "Personal information protection and electronic documents act," 2000, accessed on May 5, 2025. [Online]. Available: https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/

[135] N. C. of Brazil, "General data protection law," 2018, accessed on May 5, 2025. [Online]. Available: https://www.gov.br/anpd/pt-br/centrais-de-conteudo/outros-documentos-e-publicacoes-institucionais/lgpd-en-lei-no-13-709-capa.pdf

[136] A. J. Slagell, K. Lakkaraju, and K. Luo, “Flaim: A multi-level anonymization framework for computer and network logs.” in *LISA*, vol. 6, 2006, pp. 3–8.

[137] Y. Smamash, “Truncation method of reduction: a viable alternative,” *Electronics letters*, vol. 17, no. 2, pp. 97–99, 1981.

[138] K. Mivule and B. Anderson, “A study of usability-aware network trace anonymization,” in *2015 Science and Information Conference (SAI)*. IEEE, 2015, pp. 1293–1304.

[139] E. Boschi and B. Trammell, “Ip flow anonymization support,” Tech. Rep., 2011.

[140] S. He, J. Zhu, P. He, and M. R. Lyu, “Experience report: System log analysis for anomaly detection,” in *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

[141] Widodo, E. K. Budiardjo, and W. C. Wibowo, “Privacy preserving data publishing with multiple sensitive attributes based on overlapped slicing,” *Information*, vol. 10, no. 12, p. 362, 2019.

[142] J. Soria-Comas, J. Domingo-Ferrer, D. Sanchez, and S. Martinez, “t-closeness through microaggregation: Strict privacy with enhanced utility preservation,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 11, pp. 3098–3110, 2015.

[143] M. Lab, “Studying AIOps Projects on GitHub Replication Package,” https://github.com/roozbehaghili/studying_aiops_github, 2023, last accessed: 2025/06/20.

[144] GitHub, “GitHub,” https://github.com/, 2025, last accessed: 2025/06/20.

[145] S. Li, Y. Wu, Y. Liu, D. Wang, M. Wen, Y. Tao, Y. Sui, and Y. Liu, “An exploratory study of bugs in extended reality applications on the web,” in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 172–183.

[146] M. Openja, M. M. Morovati, L. An, F. Khomh, and M. Abidi, “Technical debts and faults in open-source quantum software systems: An empirical study,” *Journal of Systems and Software*, vol. 193, p. 111458, 2022.

[147] F. Majidi, M. Openja, F. Khomh, and H. Li, “An empirical study on the usage of automated machine learning tools,” in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 59–70.

[148] A. M. Dakhel, M. C. Desmarais, and F. Khomh, "Dev2vec: Representing domain expertise of developers in an embedding space," *Information and Software Technology*, vol. 159, p. 107218, 2023.

[149] P. L. Foalem, F. Khomh, and H. Li, "Studying logging practice in machine learning-based applications," *arXiv preprint arXiv:2301.04234*, 2023.

[150] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *IEEE Transactions on software engineering*, no. 7, pp. 733–743, 1986.

[151] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data mining and knowledge discovery*, vol. 15, no. 1, pp. 55–86, 2007.

[152] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data mining and knowledge discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[153] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM sigmod record*, vol. 29, no. 2, pp. 1–12, 2000.

[154] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, 2017.

[155] R. Artstein and M. Poesio, "Inter-coder agreement for computational linguistics," *Computational linguistics*, vol. 34, no. 4, pp. 555–596, 2008.

[156] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[157] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[158] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang, "An empirical study on tensorflow program bugs," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 129–140.

[159] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 510–520.

[160] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, "A comprehensive study on challenges in deploying deep learning based software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 750–762.

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

[162] J. Li, L. Dai, F. Tan, H. Shen, Z. Wang, B. Sheng, and P. Hu, "Cdx-net: Cross-domain multi-feature fusion modeling via deep neural networks for multivariate time series forecasting in aiops," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2022, pp. 4073–4077.

[163] A. Di Stefano, A. Di Stefano, G. Morana, and D. Zito, "Prometheus and aiops for the orchestration of cloud-native applications in ananke," in *2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE).* IEEE, 2021, pp. 27–32.

[164] I. GitHub, "To create integrations, retrieve data, and automate your workflows, build with the github rest api." https://docs.github.com/en/rest, 2022.

[165] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[166] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[167] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.

[168] J. Romano, J. D. Kromrey, J. Coraggio, J. Skowronek, and L. Devine, "Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and cohen'sd indices the most appropriate choices," in *annual meeting of the Southern Association for Institutional Research.* Citeseer, 2006, pp. 1–51.

[169] S. H. Khandkar, "Open coding," *University of Calgary*, vol. 23, p. 2009, 2009.

[170] C. Wohlin and A. Aurum, "Towards a decision-making structure for selecting a research design in empirical software engineering," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1427–1455, 2015.

[171] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: a critical review and guidelines," in *Proceedings of the 38th International conference on software engineering*, 2016, pp. 120–131.

[172] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: deep learning-based system log analysis for anomaly detection," *arXiv preprint arXiv:2107.05908*, 2021.

[173] B. Chen and Z. M. Jiang, "A survey of software log instrumentation," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[174] J. Svoboda, I. Ghafir, V. Prenosil *et al.*, "Network monitoring approaches: An overview," *Int J Adv Comput Netw Secur*, vol. 5, no. 2, pp. 88–93, 2015.

[175] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[176] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[177] S. J. Wilson, "Data representation for time series data mining: time domain approaches," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 9, no. 1, p. e1392, 2017.

[178] M. Lovrić, M. Milanović, and M. Stamenković, "Algoritmic methods for segmentation of time series: An overview," *Journal of Contemporary Economic and Business Issues*, vol. 1, no. 1, pp. 31–53, 2014.

[179] A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. Devanbu, "On the naturalness of software," *Communications of the ACM*, vol. 59, no. 5, pp. 122–131, 2016.

[180] Z. Tu, Z. Su, and P. Devanbu, "On the localness of software," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 269–280.

[181] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 178–189.

[182] K. Yao, H. Li, W. Shang, and A. E. Hassan, “A study of the performance of general compressors on log files,” *Empirical Software Engineering*, vol. 25, pp. 3043–3085, 2020.

[183] R. Ding, Q. Fu, J.-G. Lou, Q. Lin, D. Zhang, J. Shen, and T. Xie, “Healing online service systems via mining historical issue repositories,” in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE ’12, 2012, pp. 318–321.

[184] A. Riska, E. Smirni, and G. Ciardo, “Analytic modeling of load balancing policies for tasks with heavy-tailed distributions,” 2000.

[185] A. Shahidinejad, M. Ghobaei-Arani, and M. Masdari, “Resource provisioning using workload clustering in cloud computing environment: a hybrid approach,” *Cluster Computing*, vol. 24, no. 1, pp. 319–342, 2021.

[186] W. Foundation, “Analytics datasets: Pageviews,” n.d., accessed on April 8, 2024. [Online]. Available: https://dumps.wikimedia.org/other/pageviews/readme.html

[187] G. Urdaneta, G. Pierre, and M. Van Steen, “Wikipedia workload analysis for decentralized hosting,” *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, 2009.

[188] S. A. Chowdhury and D. J. Makaroff, “Category-based user interaction with online user-generated videos: workload characterization.” in *CASCON*, 2014, pp. 367–370.

[189] R. N. Calheiros, R. Ranjan, and R. Buyya, “Virtual machine provisioning based on analytical performance and qos in cloud computing environments,” in *2011 International Conference on Parallel Processing*. IEEE, 2011, pp. 295–304.

[190] D. Zhou, H. Chen, K. Shang, G. Cheng, J. Zhang, and H. Hu, “Cushion: A proactive resource provisioning method to mitigate slo violations for containerized microservices,” *IET Communications*, vol. 16, no. 17, pp. 2105–2122, 2022.

[191] Z. Amekraz and M. Y. Hadi, “An adaptive workload prediction strategy for non-gaussian cloud service using arma model with higher order statistics,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 646–651.

[192] N. Roy, A. Dubey, and A. Gokhale, “Efficient autoscaling in the cloud using predictive models for workload forecasting,” in *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 500–507.

[193] J. Dogani, F. Khunjush, and M. Seydali, “K-agrued: A container autoscaling technique for cloud-based web applications in kubernetes using attention-based gru encoder-decoder,” *Journal of Grid Computing*, vol. 20, no. 4, p. 40, 2022.

[194] M. Imdoukh, I. Ahmad, and M. G. Alfailakawi, “Machine learning-based auto-scaling for containerized applications,” *Neural Computing and Applications*, vol. 32, pp. 9745–9760, 2020.

[195] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, “Flexible and efficient decision-making for proactive latency-aware self-adaptation,” *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 13, no. 1, pp. 1–36, 2018.

[196] ——, “Efficient decision-making under uncertainty for proactive self-adaptation,” in *2016 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2016, pp. 147–156.

[197] L. N. Bairavasundaram, M. Sivathanu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “X-ray: A non-invasive exclusive caching mechanism for raids,” *ACM SIGARCH Computer Architecture News*, vol. 32, no. 2, p. 176, 2004.

[198] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, “Changes in web client access patterns: Characteristics and caching implications,” *World Wide Web*, vol. 2, pp. 15–28, 1999.

[199] T. Shi, H. Ma, G. Chen, and S. Hartmann, “Auto-scaling containerized applications in geo-distributed clouds,” *IEEE Transactions on Services Computing*, 2023.

[200] M. Abdullah, W. Iqbal, A. Mahmood, F. Bukhari, and A. Erradi, “Predictive autoscaling of microservices hosted in fog microdata center,” *IEEE Systems Journal*, vol. 15, no. 1, pp. 1275–1286, 2020.

[201] M. C. Calzarossa, L. Massari, and D. Tessera, “Workload characterization: A survey revisited,” *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–43, 2016.

[202] S. Shishira, A. Kandasamy, and K. Chandrasekaran, “Workload characterization: Survey of current approaches and research challenges,” in *Proceedings of the 7th international conference on computer and communication technology*, 2017, pp. 151–156.

[203] P. Singh, P. Gupta, K. Jyoti, and A. Nayyar, “Research on auto-scaling of web applications in cloud: survey, trends and future directions,” *Scalable Computing: Practice and Experience*, vol. 20, no. 2, pp. 399–432, 2019.

[204] C. Qu, R. N. Calheiros, and R. Buyya, “Auto-scaling web applications in clouds: A taxonomy and survey,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–33, 2018.

[205] M. Masdari and A. Khoshnevis, “A survey and classification of the workload forecasting methods in cloud computing,” *Cluster Computing*, vol. 23, no. 4, pp. 2399–2424, 2020.

[206] M. Lab, “Web Application Workloads Replication Package,” https://github.com/mooselab/web-app-workloads, 2024, last accessed: 2025/06/20.

[207] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering.* Springer Science & Business Media, 2012.

[208] IEEE, “IEEE Xplore Digilat Library,” https://ieeexplore.ieee.org/, 2025, last accessed: 2025/06/20.

[209] ACM, “ACM Digilat Library,” https://dl.acm.org/, 2025, last accessed: 2025/06/20.

[210] T. Wu, M. Pan, and Y. Yu, “A long-term cloud workload prediction framework for reserved resource allocation,” in *2022 IEEE International Conference on Services Computing (SCC).* IEEE, 2022, pp. 134–139.

[211] V. K. Jayakumar, S. Arbat, I. K. Kim, and W. Wang, “Cloudbruno: A low-overhead online workload prediction framework for cloud computing,” in *2022 IEEE International Conference on Cloud Engineering (IC2E).* IEEE, 2022, pp. 188–198.

[212] J. Kumar and A. K. Singh, “Performance assessment of time series forecasting models for cloud datacenter networks’ workload prediction,” *Wireless Personal Communications*, vol. 116, no. 3, pp. 1949–1969, 2021.

[213] ——, “Cloud datacenter workload estimation using error preventive time series forecasting models,” *Cluster Computing*, vol. 23, no. 2, pp. 1363–1379, 2020.

[214] F. Ebadifard and S. M. Babamir, “Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment,” *Cluster Computing*, vol. 24, pp. 1075–1101, 2021.

[215] G. Saravanan and A. Santhosh Babu, “Workload prediction for enhancing power efficiency of cloud data centers using optimized self-attention-based progressive generative adversarial network,” *International Journal of Communication Systems*, vol. 37, no. 1, p. e5634, 2024.

[216] R. Karthikeyan, V. Balamurugan, R. Cyriac, and B. Sundaravadivazhagan, "Cosco2: Ai-augmented evolutionary algorithm based workload prediction framework for sustainable cloud data centers," *Transactions on Emerging Telecommunications Technologies*, vol. 34, no. 1, p. e4652, 2023.

[217] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of www client-based traces," Technical Report TR-95-010, Boston University Department of Computer Science, Tech. Rep., 1995.

[218] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: Evidence and possible causes," *IEEE/ACM Transactions on networking*, vol. 5, no. 6, pp. 835–846, 1997.

[219] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 800–813, 2018.

[220] F. Tahir, M. Abdullah, F. Bukhari, K. M. Almustafa, and W. Iqbal, "Online workload burst detection for efficient predictive autoscaling of applications," *IEEE Access*, vol. 8, pp. 73 730–73 745, 2020.

[221] Z. Amekraz and M. Y. Hadi, "Canfis: a chaos adaptive neural fuzzy inference system for workload prediction in the cloud," *IEEE Access*, vol. 10, pp. 49 808–49 828, 2022.

[222] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of youtube network traffic at a campus network–measurements, models, and implications," *Computer networks*, vol. 53, no. 4, pp. 501–514, 2009.

[223] ——, "Watch global, cache local: Youtube network traffic at a campus network: measurements and implications," in *Multimedia Computing and Networking 2008*, vol. 6818. SPIE, 2008, pp. 35–47.

[224] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Efficient resource provisioning for elastic cloud services based on machine learning techniques," *Journal of Cloud Computing*, vol. 8, no. 1, pp. 1–18, 2019.

[225] M. Abdullah, W. Iqbal, J. L. Berral, J. Polo, and D. Carrera, "Burst-aware predictive autoscaling for containerized microservices," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1448–1460, 2020.

[226] J. J. Prevost, K. Nagothu, B. Kelley, and M. Jamshidi, “Prediction of cloud data center networks loads using stochastic and neural models,” in *2011 6th International Conference on System of Systems Engineering.* IEEE, 2011, pp. 276–281.

[227] D. G. Feitelson, “Metrics for mass-count disparity,” in *14th IEEE International Symposium on Modeling, Analysis, and Simulation.* IEEE, 2006, pp. 61–68.

[228] Y. Lei, Y. Gong, S. Zhang, and G. Li, “Research on scheduling algorithms in web cluster servers,” *Journal of Computer Science and Technology*, vol. 18, pp. 703–716, 2003.

[229] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey, “Cloudinsight: Utilizing a council of experts to predict future cloud application workloads,” in *2018 IEEE 11th international conference on cloud computing (CLOUD).* IEEE, 2018, pp. 41–48.

[230] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, “Self-adaptive workload classification and forecasting for proactive resource provisioning,” in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013, pp. 187–198.

[231] V. Jaiman, S. B. Mokhtar, V. Quéma, L. Y. Chen, and E. Rivìere, “Héron: Taming tail latencies in key-value stores under heterogeneous workloads,” in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS).* IEEE, 2018, pp. 191–200.

[232] M. Arlitt and T. Jin, “A workload characterization study of the 1998 world cup web site,” *IEEE network*, vol. 14, no. 3, pp. 30–37, 2000.

[233] M. F. Arlitt and C. L. Williamson, “Web server workload characterization: The search for invariants,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 24, no. 1, pp. 126–137, 1996.

[234] J. Kumar and A. K. Singh, “Performance evaluation of metaheuristics algorithms for workload prediction in cloud environment,” *Applied Soft Computing*, vol. 113, p. 107895, 2021.

[235] L. B. N. Laboratory, “Worldcup98 trace,” n.d., accessed on April 8, 2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/WorldCup.html

[236] ——, “Nasa trace,” 1995, accessed on April 8, 2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html

[237] ——, “Saskatchewan trace,” 1995, accessed on April 8, 2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/Sask-HTTP.html

[238] ——, “Calgary trace,” 1995, accessed on April 8, 2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/Calgary-HTTP.html

[239] ——, “Epa trace,” 1995, accessed on April 8, 2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html

[240] ——, “Clarknet trace,” 1995, accessed on April 8, 2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html

[241] R. Rocket, “Retailrocket trace,” 2015, accessed on April 8, 2024. [Online]. Available: https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset

[242] L. B. N. Laboratory, “Boston trace,” 1995, accessed on April 8, 2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/BU-Web-Client.html

[243] ——, “Sdsc trace,” 1995, accessed on April 8, 2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/SDSC-HTTP.html

[244] Y. G. Michael Zink, Kyoungwon Suh and J. Kurose, “Youtube traces from the campus network,” 2008, accessed on April 8, 2024. [Online]. Available: https://traces.cs.umass.edu/index.php/Network/Network

[245] C. U. of Madrid, “Complutense university of madrid trace,” 2018, accessed on April 8, 2024. [Online]. Available: https://drive.google.com/file/d/1a94djn7dRQAhciPsBk1SLIzsfC2EuPf4/view

[246] M. Attia, M. Arafa, E. Sallam, and M. Fahmy, “Application of an enhanced self-adapting differential evolution algorithm to workload prediction in cloud computing,” *Int. J. Inf. Technol. Comput. Sci.*, vol. 11, no. 8, pp. 33–40, 2019.

[247] A. Ali-Eldin, J. Tordsson, E. Elmroth, and M. Kihl, “Workload classification for efficient auto-scaling of cloud resources,” *Department of Computer Science, Umea University, Umea, Sweden, Tech. Rep*, vol. 2, 2013.

[248] V. Persico, D. Grimaldi, A. Pescape, A. Salvi, and S. Santini, “A fuzzy approach based on heterogeneous metrics for scaling out public clouds,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2117–2130, 2017.

[249] D. Kumar and N. K. Gondhi, “A qos-based reactive auto scaler for cloud environment,” in *2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS)*. IEEE, 2017, pp. 19–23.

[250] A. V. Papadopoulos, A. Ali-Eldin, K.-E. Årzén, J. Tordsson, and E. Elmroth, "Peas: A performance evaluation framework for auto-scaling strategies in cloud applications," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 1, no. 4, pp. 1–31, 2016.

[251] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud computing patterns: fundamentals to design, build, and manage cloud applications.* Springer, 2014, vol. 545.

[252] Z. M. Jiang and A. E. Hassan, "A survey on load testing of large-scale software systems," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1091–1118, 2015.

[253] M. Arlitt, D. Krishnamurthy, and J. Rolia, "Characterizing the scalability of a large web-based shopping system," *ACM Transactions on Internet Technology (TOIT)*, vol. 1, no. 1, pp. 44–69, 2001.

[254] C. Karmaker, "Determination of optimum smoothing constant of single exponential smoothing method: a case study," *International Journal of Research in Industrial Engineering*, vol. 6, no. 3, pp. 184–192, 2017.

[255] Z. Wu, N. E. Huang, S. R. Long, and C.-K. Peng, "On the trend, detrending, and variability of nonlinear and nonstationary time series," *Proceedings of the National Academy of Sciences*, vol. 104, no. 38, pp. 14 889–14 894, 2007.

[256] K.-I. Goh and A.-L. Barabási, "Burstiness and memory in complex systems," *Europhysics Letters*, vol. 81, no. 4, p. 48002, 2008.

[257] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks*, vol. 16, no. 3, pp. 645–678, 2005.

[258] M. Syakur, B. Khotimah, E. Rochman, and B. D. Satoto, "Integration k-means clustering method and elbow method for identification of the best customer profile cluster," in *IOP conference series: materials science and engineering*, vol. 336. IOP Publishing, 2018, p. 012017.

[259] K. R. Shahapure and C. Nicholas, "Cluster quality analysis using silhouette score," in *2020 IEEE 7th international conference on data science and advanced analytics (DSAA).* IEEE, 2020, pp. 747–748.

[260] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and image processing*, vol. 14, no. 3, pp. 227–248, 1980.

[261] M. Müller, "Dynamic time warping," *Information retrieval for music and motion*, pp. 69–84, 2007.

[262] M. Cuturi and M. Blondel, "Soft-dtw: a differentiable loss function for time-series," in *International conference on machine learning.* PMLR, 2017, pp. 894–903.

[263] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[264] A. Ranganathan, "The levenberg-marquardt algorithm," *Tutoral on LM algorithm*, vol. 11, no. 1, pp. 101–110, 2004.

[265] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey, "Empirical evaluation of workload forecasting techniques for predictive cloud resource scaling," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD).* IEEE, 2016, pp. 1–10.

[266] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, J. Guo, C. Sporea, A. Toma, and S. Sajedi, "Using black-box performance models to detect performance regressions under varying workloads: an empirical study," *Empirical Software Engineering*, vol. 25, pp. 4130–4160, 2020.

[267] F. Di Menna, L. Traini, and V. Cortellessa, "Time series forecasting of runtime software metrics: An empirical study," in *Proceedings of the 2024 ACM/SPEC International Conference on Performance Engineering*, 2024.

[268] W. Chen, K. Ye, Y. Wang, G. Xu, and C.-Z. Xu, "How does the workload look like in production cloud? analysis and clustering of workloads on alibaba cluster trace," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS).* IEEE, 2018, pp. 102–109.

[269] T. Barik, R. DeLine, S. Drucker, and D. Fisher, "The bones of the system: A case study of logging and telemetry at microsoft," in *Proceedings of the 38th International Conference on Software Engineering Companion*, 2016, pp. 92–101.

[270] H. Li, W. Shang, B. Adams, M. Sayagh, and A. E. Hassan, "A qualitative study of the benefits and costs of logging from developers' perspectives," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2858–2873, 2020.

[271] M. Cinque, D. Cotroneo, and A. Pecchia, “Event logs for the analysis of software failures: A rule-based approach,” *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2012.

[272] M. A. Batoun, M. Sayagh, R. Aghili, A. Ouni, and H. Li, “A literature review and existing challenges on software logging practices: From the creation to the analysis of software logs,” *Empirical Software Engineering*, vol. 29, no. 4, p. 103, 2024.

[273] Q. Qin, R. Aghili, H. Li, and E. Merlo, “Preprocessing is all you need: Boosting the performance of log parsers with a general preprocessing framework,” *arXiv preprint arXiv:2412.05254*, 2024.

[274] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, “Sherlog: error diagnosis by connecting clues from run-time logs,” in *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*, 2010, pp. 143–154.

[275] P. Jain, M. Gyanchandani, and N. Khare, “Big data privacy: a technological perspective and review,” *Journal of Big Data*, vol. 3, pp. 1–25, 2016.

[276] X. Gu and K. Dong, “Pd-pan: Prefix-and distribution-preserving internet of things traffic anonymization,” *Electronics*, vol. 12, no. 20, p. 4369, 2023.

[277] F. McSherry and R. Mahajan, “Differentially-private network trace analysis,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 123–134, 2010.

[278] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon, “Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme,” in *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* IEEE, 2002, pp. 280–289.

[279] T. Barbaro, Michael; Zeller Jr, “A face is exposed for aol searcher no. 4417749,” 2006, accessed on September 11, 2024. [Online]. Available: https://www.nytimes.com/2006/08/09/technology/09aol.html

[280] A. Cooper, “A survey of query log privacy-enhancing techniques from a policy perspective,” *ACM Transactions on the Web (TWEB)*, vol. 2, no. 4, pp. 1–27, 2008.

[281] A. Narayanan and V. Shmatikov, “How to break anonymity of the netflix prize dataset,” *arXiv preprint cs/0610105*, 2006.

[282] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

[283] Y. Li, A. Slagell, K. Luo, and W. Yurcik, "Canine: A combined conversion and anonymization tool for processing netflows for security," in *International conference on telecommunication systems modeling and analysis*, vol. 21. Citeseer, 2005.

[284] M. Mohammady, L. Wang, Y. Hong, H. Louafi, M. Pourzandi, and M. Debbabi, "Preserving both privacy and utility in network trace anonymization," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 459–474.

[285] GDPRhub, "Cjeu - c-582/14 - patrick breyer," 2023, accessed on September 11, 2024. [Online]. Available: https://gdprhub.eu/index.php?title=CJEU_-_C-582/14_-_Patrick_Breyer

[286] I. O. for Standardization, "Iso/iec 27001:2022," 2022, accessed on September 11, 2024. [Online]. Available: https://www.iso.org/standard/27001

[287] C. Han, K. Sun, H. Tang, Y. Wu, and X. Zhang, "Aft-anon: A scaling method for online trace anonymization based on anonymous flow tables," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–7.

[288] L. D. Manocchio, S. Layeghy, D. Gwynne, and M. Portmann, "A configurable anonymisation approach for network flow data: Balancing utility and privacy," *Computers and Electrical Engineering*, vol. 118, p. 109465, 2024.

[289] A. Slagell and W. Yurcik, "Sharing computer network logs for security and privacy: A motivation for new methodologies of anonymization," in *Workshop of the 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks, 2005.* IEEE, 2005, pp. 80–89.

[290] M. Foukarakis, D. Antoniades, and M. Polychronakis, "Deep packet anonymization," in *Proceedings of the Second European Workshop on System Security*, 2009, pp. 16–21.

[291] M. Foukarakis, D. Antoniades, S. Antonatos, and E. P. Markatos, "Flexible and high-performance anonymization of netflow records using anontool," in *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007.* IEEE, 2007, pp. 33–38.

[292] G. Minshall, "Tcpdpriv," 2005, accessed on September 11, 2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/tcpdpriv.html

[293] D. Plonka, "ip2anonip," 2003, accessed on September 11, 2024. [Online]. Available: https://pages.cs.wisc.edu/%7Eplonka/ip2anonip/

[294] M. Burkhart, D. Schatzmann, B. Trammell, E. Boschi, and B. Plattner, "The role of network trace anonymization under attack," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 5–11, 2010.

[295] J. King, K. Lakkaraju, and A. Slagell, "A taxonomy and adversarial model for attacks against network log anonymization," in *Proceedings of the 2009 ACM symposium on Applied Computing*, 2009, pp. 1286–1293.

[296] P. Silva, E. Monteiro, and P. Simoes, "Privacy in the cloud: A survey of existing solutions and research challenges," *IEEE access*, vol. 9, pp. 10 473–10 497, 2021.

[297] A. Majeed, S. Khan, and S. O. Hwang, "Toward privacy preservation using clustering based anonymization: recent advances and future research outlook," *IEEE Access*, vol. 10, pp. 53 066–53 097, 2022.

[298] A. Agrawal, A. Dixit, N. A. Shettar, D. Kapadia, V. Agrawal, R. Gupta, and R. Karlupia, "Delog: A high-performance privacy preserving log filtering framework," in *2019 IEEE International Conference on Big Data (Big Data).* IEEE, 2019, pp. 1739–1748.

[299] T. Li, S. Zhang, Z. Dang, Y. Xiao, and Z. Ma, "Triplelp: Privacy-preserving log parsing based on blockchain," in *2023 IEEE 14th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP).* IEEE, 2023, pp. 1–6.

[300] B. A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-based software engineering and systematic reviews.* CRC press, 2015, vol. 4.

[301] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to advanced empirical software engineering.* Springer, 2008, pp. 63–92.

[302] K. Wang and M. M. H. Khan, "Performance prediction for apache spark platform," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems.* IEEE, 2015, pp. 166–173.

[303] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using random indexing and support vector machines," *Journal of Systems and Software*, vol. 86, no. 1, pp. 2–11, 2013.

[304] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications (short paper)," in *2008 The Eighth International Conference on Quality Software.* IEEE, 2008, pp. 181–186.

[305] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 353–366.

[306] M. Landauer, M. Wurzenberger, F. Skopik, G. Settanni, and P. Filzmoser, "Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection," *computers & security*, vol. 79, pp. 94–116, 2018.

[307] P. Ryciak, K. Wasielewska, and A. Janicki, "Anomaly detection in log files using selected natural language processing methods," *Applied Sciences*, vol. 12, no. 10, p. 5089, 2022.

[308] Ł. Korzeniowski and K. Goczyła, "Landscape of automated log analysis: A systematic literature review and mapping study," *IEEE Access*, vol. 10, pp. 21 892–21 913, 2022.

[309] R. Aghili, H. Li, and F. Khomh, "Studying the characteristics of aiops projects on github," *Empirical Software Engineering*, vol. 28, no. 6, p. 143, 2023.

[310] R. Aghili, Q. Qin, H. Li, and F. Khomh, "Understanding web application workloads and their applications: Systematic literature review and characterization," in *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 2024, pp. 474–486.

[311] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A practical guide, 1st ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[312] R. Aghili, H. Li, and F. Khomh, "Protecting privacy in software logs: What should be anonymized?" *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, 2025.

[313] U. Flegel, "Pseudonymizing unix log files," in *International Conference on Infrastructure Security.* Springer, 2002, pp. 162–179.

[314] Y.-D. Lin, P.-C. Lin, S.-H. Wang, I.-W. Chen, and Y.-C. Lai, "Pcaplib: A system of extracting, classifying, and anonymizing real packet traces," *IEEE Systems Journal*, vol. 10, no. 2, pp. 520–531, 2014.

[315] M. Lab, "SDLog Main Model Hugging Face," https://huggingface.co/LogSensitiveResearcher/SDLog_main, 2025, last accessed: 2025/06/20.

[316] ——, "SDLog Net Model Hugging Face," https://huggingface.co/LogSensitiveResearcher/SDLog_net, 2025, last accessed: 2025/06/20.

[317] ——, "SDLog Replication Package," https://github.com/mooselab/SDLog, 2025, last accessed: 2025/06/20.

[318] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[319] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *IEEE transactions on knowledge and data engineering*, vol. 34, no. 1, pp. 50–70, 2020.

[320] C. Aone, "A trainable summarizer with knowledge acquired from robust nlp techniques," *Advances in automatic text summarization*, pp. 71–80, 1999.

[321] D. Mollá, M. Van Zaanen, and S. Cassidy, "Named entity recognition in question answering of speech data," in *Proceedings of the 2007 Australasian Language Technology Workshop.* ALTA, 2007, pp. 57–65.

[322] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel, "Nymble: a high-performance learning name-finder," *arXiv preprint cmp-lg/9803003*, 1998.

[323] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman, "Description of the mene named entity system as used in muc-7," in *Proceedings of the Seventh Message Understanding Conference (MUC-7), Fairfax, Virginia, April 29-May 1, 1998*, 1998.

[324] A. McCallum and W. Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, 2003, pp. 188–191.

[325] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[326] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, cnn architectures, challenges, applications, future directions," *Journal of big Data*, vol. 8, pp. 1–74, 2021.

[327] C. Sun, Z. Yang, L. Wang, Y. Zhang, H. Lin, and J. Wang, "Biomedical named entity recognition using bert in the machine reading comprehension framework," *Journal of Biomedical Informatics*, vol. 118, p. 103799, 2021.

[328] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits, "Is bert really robust? a strong baseline for natural language attack on text classification and entailment," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34-05, 2020, pp. 8018–8025.

[329] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[330] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[331] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[332] E. Mashhadi and H. Hemmati, "Applying codebert for automated program repair of java simple bugs," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 505–509.

[333] J. Huang, D. Tang, L. Shou, M. Gong, K. Xu, D. Jiang, M. Zhou, and N. Duan, "Cosqa: 20,000+ web queries for code search and question answering," *arXiv preprint arXiv:2105.13239*, 2021.

[334] J. Tabassum, M. Maddela, W. Xu, and A. Ritter, "Code and named entity recognition in stackoverflow," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.443/

[335] Z. Jiang, J. Liu, J. Huang, Y. Li, Y. Huo, J. Gu, Z. Chen, J. Zhu, and M. R. Lyu, "A large-scale evaluation for log parsing techniques: How far are we?" in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 223–234.

[336] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, and M. R. Lyu, "Lilac: Log parsing using llms with adaptive parsing cache," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 137–160, 2024.

[337] S. Shan, Y. Huo, Y. Su, Y. Li, D. Li, and Z. Zheng, "Face it yourselves: An llm-based two-stage strategy to localize configuration errors via logs," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 13–25.

[338] L. A. Ramshaw and M. P. Marcus, "Text chunking using transformation-based learning," in *Natural language processing using very large corpora.* Springer, 1999, pp. 157–176.

[339] Z. Li, C. Luo, T.-H. Chen, W. Shang, S. He, Q. Lin, and D. Zhang, "Did we miss something important? studying and exploring variable-aware log abstraction," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 2023, pp. 830–842.

[340] A. R. Chen, T.-H. Chen, and S. Wang, "Demystifying the challenges and benefits of analyzing user-reported logs in bug reports," *Empirical Software Engineering*, vol. 26, pp. 1–30, 2021.

[341] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," 2011.

[342] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[343] Hugging Face, "Token classification with transformers," 2025, accessed on May 5, 2025. [Online]. Available: https://huggingface.co/docs/transformers/tasks/token_classification

[344] A. Paszke, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[345] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[346] ELSEVIER, "Engineering Village Digilat Library," https://www.engineeringvillage.com/, 2025, last accessed: 2025/06/20.

[347] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, “Towards automated log parsing for large-scale log data analysis,” *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2017.

[348] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, “Swisslog: Robust anomaly detection and localization for interleaved unstructured logs,” *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 2762–2780, 2022.

[349] H. Dai, Y. Tang, H. Li, and W. Shang, “Pilar: Studying and mitigating the influence of configurations on log parsing,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 818–829.

[350] S. Yu, P. He, N. Chen, and Y. Wu, “Brain: Log parsing with bidirectional parallel tree,” *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3224–3237, 2023.

[351] J. Xu, Q. Fu, Z. Zhu, Y. Cheng, Z. Li, Y. Ma, and P. He, “Hue: A user-adaptive parser for hybrid logs,” in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 413–424.

[352] J. Zhang and F. Wang, “Research on the construction of log parsing system based on regular expression,” in *2022 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*. IEEE, 2022, pp. 627–630.

[353] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, “Paddy: An event log parsing approach using dynamic dictionary,” in *NOMS 2020-2020 IEEE/IFIP network operations and management symposium*. IEEE, 2020, pp. 1–8.

[354] S. Yu, Y. Wu, Y. Li, and P. He, “Unlocking the power of numbers: Log compression via numeric token parsing,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 919–930.

[355] W. Niu, Z. Li, Z. He, A. Wang, B. Li, and X. Zhang, “Fsmflog: Discovering anomalous logs combining full semantic information and multifeature fusion,” *IEEE Internet of Things Journal*, vol. 11, no. 3, pp. 4442–4453, 2023.

[356] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, “A search-based approach for accurate identification of log message formats,” in *Proceedings of the 26th Conference on Program Comprehension*, 2018, pp. 167–177.

[357] R. Naumiuk and J. Legierski, "Anonymization of data sets from service delivery platforms," in *2014 Federated Conference on Computer Science and Information Systems*. IEEE, 2014, pp. 955–960.

[358] J. Wang, T. Li, R. Zhang, Z. Tang, D. Wu, and Z. Yang, "Vcrlog: Variable contents relationship perception for log-based anomaly detection," in *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2024, pp. 156–167.

[359] Y. Li, Y. Liu, H. Wang, Z. Chen, W. Cheng, Y. Chen, W. Yu, H. Chen, and C. Liu, "Glad: Content-aware dynamic graphs for log anomaly detection," in *2023 IEEE International Conference on Knowledge Graph (ICKG)*. IEEE, 2023, pp. 9–18.

[360] B. Debnath, M. Solaimani, M. A. G. Gulzar, N. Arora, C. Lumezanu, J. Xu, B. Zong, H. Zhang, G. Jiang, and L. Khan, "Loglens: A real-time log analysis system," in *2018 IEEE 38th international conference on distributed computing systems (ICDCS)*. IEEE, 2018, pp. 1052–1062.

[361] Z. Li, Q. Fu, Z. Huang, J. Yu, Y. Li, Y. Lai, and Y. Ma, "Revisiting log parsing: The present, the future, and the uncertainties," *IEEE Transactions on Reliability*, vol. 73, no. 3, pp. 1459–1472, 2024.

[362] Z. Wahab, S. Ahmed, M. N. Rahman, R. Shahriyar, and G. Uddin, "Secret breach prevention in software issue reports," *arXiv preprint arXiv:2410.23657*, 2024.

[363] S. Yu, Y. Wu, Z. Li, P. He, N. Chen, and C. Liu, "Log parsing with generalization ability under new log types," in *Proceedings of the 31st acm joint european software engineering conference and symposium on the foundations of software engineering*, 2023, pp. 425–437.

[364] M. Yadav and D. S. Mishra, "Identification of network threats using live log stream analysis," in *2023 2nd International Conference on Paradigm Shifts in Communications Embedded Systems, Machine Learning and Signal Processing (PCEMS)*. IEEE, 2023, pp. 1–6.

[365] T.-T. Wong, "Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation," *Pattern recognition*, vol. 48, no. 9, pp. 2839–2846, 2015.

[366] T. Fushiki, "Estimation of prediction error by using k-fold cross-validation," *Statistics and Computing*, vol. 21, pp. 137–146, 2011.