How Industry Tackles Anomalies during Runtime: Approaches and Key Monitoring Parameters

Monika Steidl*, Benedikt Dornauer†, Michael Felderer‡,
Rudolf Ramler§, Mircea-Cristian Racasan¶, Marko Gattringer||
*†‡*University of Innsbruck*, Innsbruck, Austria
§*Software Competence Center Hagenberg GmbH*, Hagenberg, Austria
‡*German Aerospace Center (DLR), Institute of Software Technology*, Cologne, Germany
¶*c.c.com Moser GmbH*, Germany
||*Gepardec IT Services*, Wien, Austria
†‡*University of Cologne*, Cologne, Germany
ORCID: *0000-0002-3410-7637, †0000-0002-7713-4686,‡0000-0003-3818-4442,
§0000-0001-9903-6107,¶0009-0008-7938-3126,||0000-0003-1659-3624

***Abstract*—Deviations from expected behavior during runtime, known as anomalies, have become more common due to the systems' complexity, especially for microservices. Consequently, analyzing runtime monitoring data, such as logs, traces for microservices, and metrics, is challenging due to the large volume of data collected. Developing effective rules or AI algorithms requires a deep understanding of this data to reliably detect unforeseen anomalies. This paper seeks to comprehend anomalies and current anomaly detection approaches across diverse industrial sectors. Additionally, it aims to pinpoint the parameters necessary for identifying anomalies via runtime monitoring data. Therefore, we conducted semi-structured interviews with fifteen industry participants who rely on anomaly detection during runtime. Additionally, to supplement information from the interviews, we performed a literature review focusing on anomaly detection approaches applied to industrial real-life datasets. Our paper (1) demonstrates the diversity of interpretations and examples of software anomalies during runtime and (2) explores the reasons behind choosing rule-based approaches in the industry over self-developed AI approaches. AI-based approaches have become prominent in published industry-related papers in the last three years. Furthermore, we (3) identified key monitoring parameters collected during runtime (logs, traces, and metrics) that assist practitioners in detecting anomalies during runtime without introducing bias in their anomaly detection approach due to inconclusive parameters.**

***Index Terms*—anomaly detection, runtime monitoring data, parameter extraction, logs, metrics, traces, microservices**

I. Introduction

In the late 1970s and early 1980s, system administrators manually inspected printed audit logs, which often piled up to four to five feet by week's end, to search for suspicious behavior. This might mark the start of software anomaly detection [1]. Nowadays, frequent software changes have become the industry norm for fixing bugs, improving performance, and enhancing user satisfaction with new features [2]. Due to these extensive changes, classical test suites frequently cannot prevent all potential *deviations from expected behavior* [3] at runtime, often described as anomalies. Beyond software issues, system reliance on hardware may also result in system failures or performance declines from processing queue saturation [4]. Mariani et al. [5] even stated that runtime anomalies are becoming the norm rather than the exception in various systems (e.g., ultra-large systems, system of systems, or cloud systems). Therefore, the industry requires effective strategies to preserve the integrity and functionality of its software systems even during runtime.

These effective strategies must handle volatile anomalies with different root causes and varying observable behavior. Currently, there is no consensus on a classical notion of software anomalies during runtime [6]. This makes it hard to consistently identify and communicate anomalies, complicating efforts to ensure software quality, prioritize the handling of specific anomalies, and manage risks based on severity and impact. Thus, it is crucial to understand how the industry interprets, and characterises these anomalies. In addition, gaining more insights into potential anomalies by extending related work [7]–[17] with further experienced real-life anomalies, allows to enhance awareness of further potential anomalies:

> **RQ1:** What are the prevailing interpretations, characteristics, and examples of anomalies within the industry?

Anomaly detection during runtime involves continuously monitoring and analyzing a system's operations to identify the underlying reasons, also known as root cause, for anomalies in time or before the system is affected. Thus, developers frequently find themselves manually examining a huge volume of runtime monitoring data, such as logs, traces (for microservices), or metrics that exhibit big data characteristics [18]. Not only is analyzing this runtime monitoring data labor-intensive due to a large amount of available data and challenging due to the system's complexity (e.g., constant changes in traffic, scaling requirements, complex infrastructures), but behavior that appears anomalous may not always signify an

This work was supported by the Austrian Research Promotion Agency (FFG) in the frame of the ConTest project [888127] and the SmartQuality (SmartDelta) project [890417]. We thank the study participants, including Vaadin, Akkodis, Turkcell Technology, Hoxhunt, and all other anonymized partners, for their valuable insights.

actual anomaly [8], [16], [19]. To ease this cumbersome work, semi-automated approaches to assist developers and operators in detecting software anomalies during runtime exist. These approaches rely on self-defined rules and thresholds and Artificial Intelligence (AI) algorithms [4], [14], [20], [21] to detect anomalies and their respective root cause. We examine companies choices and rationales to understand why they opted for **rule-based** or **AI-based** approaches:

RQ2: What factors influence the selection of anomaly detection approaches in industrial settings?

Both approaches heavily rely on a high-quality dataset and domain-specific knowledge [22], [23] where a consensus on which runtime monitoring parameter indicates an anomaly is missing [24]. In related work, only a few anomaly detection algorithms explicitly indicate which parameters are used as input dataset [10], [14], [16], [25]. Thus, it is difficult to replicate published performance measurements (e.g., accuracy, precision, and recall) of open-source AI-based anomaly detection methods, indicating their high dependency on optimized hyperparameters and a comprehensive monitoring dataset that includes all essential parameters without introducing biases that could skew the results [8].

Parameters are extracted from runtime monitoring data, such as logs, traces (for microservices), and metrics. Logs consist of predefined semi-structured emitted messages with natural language, traces are microservice-specific data types representing the end-to-end execution of a single request, and metrics consist of numeric time-series performance data. By knowing which parameters are commonly considered, anomaly detection approaches could better suit industry needs and improve their effectiveness, leading to more robust and targeted rules or training datasets for AI-based approaches [26]. The importance of recognizing essential parameters derived from runtime monitoring data is often overlooked. For instance, related work has focused on a single type of runtime monitoring data for detecting anomalies in microservices [13], [20], [21], [27]–[30], and focusing on the combination of these gain increased attention [10], [14], [17]. Understanding the relationships between these parameters (e.g., an increased response rate increases CPU usage) and the dependencies among microservices is essential for avoiding false positives. Stable relationships may indicate the absence of anomalies. This understanding allows removing unnecessary parameters to optimize storage and computational resources without relying on extensive manual work [8]. Thus, our RQ is:

RQ3: Which runtime monitoring data is used to identify anomalies by industry?

The remainder of the paper is structured as follows: Section II describes the study design of the literature review and semi-structured interviews. Section III addresses and discusses the RQ 1-3. Afterward, Section IV discusses potential threats, followed by a summary and future work in Section V.

II. Study design

We applied two research methods to answer our defined research questions. Firstly, we extended an existing (A) literature review with additional anomaly detection approaches for microservices related to industry use cases and conducted (B) semi-structured interviews. We followed the methods regarding the Empirical Standards for Software Engineering Research v2.0 [31]. For detailed information, please refer to the replication package in [32]. Due to confidentiality and company regulations, we cannot disclose the full recorded interviews and transcripts.

A. Extended Literature Review

In February 2022, Soldani and Brogi [18] published the first structured overview of current research regarding anomaly detection during runtime, root cause analysis, and required algorithms, specifically addressing the context of microservices. The additional monitoring data type for microservices, traces, is highly relevant for anomaly detection due to its popularity in the last years [19]. However, their stated methodology was a survey without systematically collecting literature based on a search string, which we aim to extend upon with our approach.

Thus, we extended their list of anomaly detection approaches with the search term "*anomaly detection runtime monitoring microservices*" where the search was executed via Google Scholar. We started in June 2023 to gain further insights into current research gaps. To consider the latest research advances, we again proceeded with the same literature study from December 2023 to January 2024, where we verified previously obtained literature and added new ones published within this timeframe. We included peer-reviewed anomaly detection algorithms for microservices published after 2015. To assess if our search string includes relevant keywords to overlap with the papers cited in Soldani and Brogi [18], we compared our findings with their papers where we were able to identify exactly 50%. However, we discovered 30 additional papers published before February 2022 that Soldani and Brogi did not include. In total, we found 92 papers focusing on anomaly detection for microservices. Of these, 36 papers were considered for further argumentation in this paper due to their evaluation based on real-life datasets and industry relevance. We excluded work that evaluated their approach with benchmark systems due to their artificial setting and injected anomalies. We provide details about the selection process and the included and excluded papers in the replication package [32].

To minimize selection bias, the second author executed a blind review of the included and excluded papers and categorized industry-relevant papers, where we achieved an inter-rater reliability of 94.7%. To resolve discrepancies, we discussed the decisions until we reached an agreement.

B. Semi-structured Interviews in Various Domains

We followed the guidelines by Runeson and Höst [33] for our semi-structured interviews and relied on recommendations for software engineering interviews by Hove and Anda [34].

We explicitly focused on interview participants from various company sizes (balancing between small and medium enterprises and large companies), microservice architecture (for tracing information), domains (holistic overview), and experiences to gain domain unspecific insights (heterogeneous sample), as outlined in Table I. We used purposive sampling to contact companies from the authors' network that rely on anomaly detection during runtime. The goal was to gain insights into experienced anomalies aiming to show that anomalies and their effect are highly volatile and unpredictable, factors influencing the selection of anomaly detection approaches, and key monitoring parameters. Thus, the goal is not to provide statistical inference but to gain and discuss qualitative insights. Therefore, we asked for skilled experts who either work as developers, data, or DevOps engineers or ensure software quality. These interview participants need to be involved in the manual or automated identification of anomalies happening during runtime based on logs, traces (for microservices), or metrics. Furthermore, along with our request, we attached our interview guidelines.

	ID		Size[1]	Specific Domain	Experience	Years	RU	AI[1]	MS[1]
Finance	B	🇹🇷	LC	Finance Payment B2B Provider	Linux-Sysadmin and R&D manager	13 & 2	✓	~	
	F	🇯🇵	LC	Global Financial Services Group	Data Engineer and Scientist for Log Anomalies	14	✓		
	M	🇦🇹	MC	Financial Service Provider	Product Owner for Software Quality, before Load Tester	18	✓	~	✓
	N				Load Tester and Release Automation Engineer	5	✓	~	✓
Web	A	🇮🇳	SC	End-user Web Application	DevOps engineer	20	✓		
	G	🇫🇮	MC	Human Risk Management Platform	Junior Security Engineer skilled in AI anomaly detection	3	✓	✓	✓
	J				Site Reliability Engineer for Cloud Applications	3	✓	✓	✓
	L	🇫🇮	MC	Web Application Framework Provider	Product Developer	14	✓		✓
Hardware	D	🇦🇹	SC	Traffic Analysis System Provider	Head of Software Engineering	16	✓		✓
	I	🇦🇹	LC	Manufacturing of Machinery	Team Lead with focus on quality assurance for SPS	10+	✓		
	O	🇩🇪	LC	Global Digital Engineering Company	Embedded Software Team Lead	9	✓		
Other	C	🇦🇹	MC	Java/JEE Solutions and Cloud Tech.	Cloud Architect	5	✓	~	
	E	🇹🇷	LC	Tele. Company and Network Provider	Senior DevOps Engineer	3	✓	~	✓
	H	🇺🇸	LC	Software Observability Platform	System and DevOps Engineer	5	✓	✓	✓
	K				Senior Data Scientist for univariate time series	4	✓	✓	✓

TABLE I
LIST OF INTERVIEW PARTICIPANTS WITH IDS ASSIGNED ALPHABETICALLY AND LISTED CHRONOLOGICALLY.

In total, we interviewed 15 participants from 12 different companies. Based on their products and services, the participants can be assigned to four categories: Finance software (software solutions for the financial sector), Web service (web-based service for end users), Technology hardware (hardware-based software or embedded systems), and Others (ranging from services to develop, integrate, and maintain custom Java/-JEE solutions and cloud technologies, telecommunication, software observability), illustrating the variety of domains.

[1]Company Size based on OECD [35]: Small Comp. [employees≤49] (SC), Medium Comp. [50≤employees≤249] (MC), Large Comp. [employees≥250] (LC). Rule-based (RU) approach. AI-based (AI) approach [∼ = integrated Dynatrace]. Microservice (MS).

Before the semi-structured interviews, we pilot-tested the guidelines with two individuals. This process allowed us to identify potential misunderstandings or comprehension difficulties and make necessary adjustments to the interview guidelines. The first two authors were present during the interviews, one responsible for guiding the interview and the other for asking additional clarification questions while taking notes. Every interview lasted at least 30 and up to 60 minutes and was conducted between October 2023 and February 2024.

C. *Data Extraction*

For both RQ1 and RQ2, we applied inductive coding by extracting relevant information from the transcribed interviews, done independently, and afterward applied color schemes.

In particular, for RQ1, we were interested in seeing how the collected interview data compares to available standards based on [36] and if there exist common interpretations and characteristics among the industry. The main identified codes are *argue that it is difficult to define anomaly, deviation of expectation, outliers of specific trends, synonym, negative effect, unforeseen or not reproducible, examples.*

For RQ2, we were interested in the type(s) of anomaly detection approaches and their specific evaluation. Therefore, the main identified codes are *rule-based, AI-based, both* and *advantage, disadvantage*.

For RQ3, we executed deductive coding of the collected papers from the literature review and transcribed interviews because we wanted to base the identified data on preconceived categories that are based on theory and existing knowledge. Soldani and Brogi [18] categorized the input for anomaly detection approaches into logs, traces, and metrics. Therefore, the main identified codes for logs are *static part, error/warnings, others*. For traces, the codes are *HTTP status code, depth of microservice invocation path, similarity between control flow graph, response time*, and for metrics, the codes are *queues, CPU, memory, network traffic, disk, energy consumption*. In RQ3, we present the overlap between literature and interviews, also considering the differences in stated parameters between literature and interview findings.

For all three RQs, if discrepancies arose, the authors resolved them through discussion where respective codes were constantly reviewed and adapted to allow an accurate representation of the data. Furthermore, we evaluated the coded results by consolidating with the interview participants afterward. We provide all interview materials, including questions and coding procedures, in the replication package [32].

III. FINDINGS AND DISCUSSION

In the following, we illustrate and discuss interpretations, characteristics, and examples of anomalies by industry (RQ1) and further elaborate existing anomaly detection approaches in the industry with their advantages and shortcomings (RQ2). These approaches rely on key monitoring parameters to gain insights into deviant behavior. Therefore, (RQ3) gathers industry-relevant parameters.

RQ1: Interpretations, Characteristics, and Examples of Anomalies Within Industry

Even today, the original definition of ”anomaly” from the *IEEE Standard 1012 of 1990* [3] persists in active standards such as the *ISO/IEC/IEEE 24765:2017 Systems and Software Engineering – Vocabulary*. It reads as follows:

”Anything observed in the documentation or operation of the software that deviates from expectations based on previously verified software products or reference documents.” [3]

Some participants [G, H, K, L] *argue that it is difficult to define anomaly* for them like the one definition above. Nevertheless, through our research, we have identified commonalities among the 15 interview participants and show their resemblances.

Similarly to the IEEE definition, the aspect of *deviations from expectations* is an essential characteristic of anomalies, also mentioned by several participants [B, C, D, G, J, K, M]. This term is also expressed in a similar manner as abnormal behavior [F] or described as behavior that is not desired [I].

An alternative perspective is a mathematical viewpoint, characterizing abnormal data as *outliers from specific trends* [E, M, H, J, M, N]. This primarily is described as data points that are outside of general patterns, such as unusual extreme values, exceeded thresholds, or abnormal decreases/increases. Participant [G] mentioned a company-internal standard definition based on specific metrics, including latency, request rates, error counts, system availability, and response times.

A recurring observation from the interviews was the depiction of anomalies using *synonyms*, particularly failure [B, C, D, K, F] or error [C, J, M, N]. In 2010, the standardization committee [36] classified software anomalies, acknowledging that the original term's broad meaning led to imprecision and impeded effective communication. Despite concerted efforts to establish precise definitions, nearly every industry interview revealed that these terms were frequently used in a manner inconsistent with their formal definitions and often used interchangeably.

Another interesting observation by two participants is that abnormal behavior is classified as an anomaly in their understanding when it results in a *negative effect*. Participant [B] defined it as any catastrophic failure. In a similar context, participant [D] linked it to disruptions in organizational operations, which means, for instance, decreased user satisfaction, blocked processes, or similar repercussions. Such circumstances need fast action [A,B,C]. Participant [A] noted that when something occurs more frequently or happens in critical situations, they place greater emphasis on resolving it.

As outlined in Table II, the wide variety of underlying reasons or root causes responsible for the observed anomaly makes it difficult to identify anomalies. For instance, the root causes might differ, but the final observed anomaly is similar, such as in the anomaly examples [L-1] and [O-1], indicating a deviation in the performance of I/O operations. The volatility complicates the development of an accurate and universal anomaly classification that should help in predicting *unforeseeable or not reproducible* [B, C, K, L, M, O] anomalies.

Examples
B-1 - Utilizing a NetApp storage system with Network File System (NFS), the company established shared storage for all Virtual Machine (VM)s that ran their services. One routine involved archiving all software versions. This particular action led to an increase in the number of inodes over time, resulting in higher response times across all services.
C-1 - A memory leak due to improper garbage collection occurred during a software release. After some days in production, the application experienced an unexpected failure. This issue remained undetected until an analysis exposed a continuous increase in memory.
D-1 - For some microservices they have self-defined rules, e.g. Random-Access Memory (RAM). If a microservice reaches a specific threshold, the service is killed and restarted, executed automatically by Kubernetes.
F-1 - During an orchestrating process executed on a VM, the system unexpectedly exhausted its heap space. The system continued to function without any immediate errors or signs of failure. The only noticeable impact was a substantial decrease in response speed.
F-2 - In a client-server interaction, the user requested some calculation handing over an input entry. The number entered was so large (probably because a key was pressed for too long) that it caused the system to crash. A log with Japanese text showed up, indicating a memory overflow error.
J-1 - In a range of scenarios, an observable aggregation of diverse jobs within the processing queue has been observed. This accumulation has manifested in extended processing times, similar to increased request times. For instance, JS-backend libraries have often been the root cause of such queuing abnormal behavior.
L-1 - Logging was used to trace abnormal states for comprehensibility and traceability. After implementing a new logging mechanism for anomaly detection, the system failed due to excessive I/O operations.
M-1 Some applications in their company suffer from poor software elasticity. For instance, the system can only support 500 users at a time. If the 501st user tries to log in, the system becomes slow, possibly unstable, and prone to errors. Eventually, it can crash.
O-1 A frequently executed routine on a CPU, accounting for only 2% of operations, leading to excessive use of flash memory, hitting hardware limits. This decreased I/O performance after 10,000 writing cycles, leading to long-term performance degradation.

TABLE II
ANOMALY EXAMPLES STATED BY INDUSTRIAL PARTICIPANTS.

Takeaways RQ1: Anomalies in Industry

Most interview participants describe an anomaly as a deviation from expectations where fast resolution is essential to minimize negative effects, which complies with IEEE definitions. Identified examples in Table II demonstrate an extract of the variety of often unforeseeable anomalies discovered by interview participants.

RQ2: Anomaly Detection Approaches in Industrial Settings

We distinguish two main approaches to identify anomalies and their root causes used by the interviewees (see Table I) and mentioned by industry papers. Anomalies detected by

- ***rule-based approaches*** rely on thresholds derived from extensive domain knowledge and company-specific insights (e.g., a range of 10% to 30% [D]), pattern matches (e.g., within a specific time frame [F]), or statistical principles (e.g., quantile regression [K]), while
- ***AI-based approaches*** use supervised and unsupervised AI models [18] to detect patterns to identify deviations.

Every participant, spanning various company sizes, state to use self-defined *rule-based* approaches. On the contrary, adopting self-developed *AI-based* approaches was done only by two companies [1st company: G, J, 2nd company: H, K], even though in the last three years, 80% of identified industry papers (e.g., [11], [14], [15], [37]) successfully applied various self-developed AI-based algorithms. Four companies, however, do not develop their own AI models but rely on commercially available AI-algorithms, e.g., *Dynatrace* used by [B, C, E, M, N]. One of the drawbacks mentioned with *Dynatrace* was the number of false positives [B, C, E, M, N]. Participants [B, M, N] receive emails when an anomaly is detected, but false positive alerts caused them to ignore these emails.

Half of the participants [B, C, D, M, N, J, O] mentioned that they prefer rule-based approaches due to the widespread availability of established monitoring tools that gather essential monitoring parameters and visualize them. Furthermore, they can define rules for these parameters that trigger alerts. The participants currently opted for this approach because they require less computational costs than AI for training [J] (which requires retraining to adapt to the latest input [G]) and detect anomalies in nearly real-time [D, F, K].

On the contrary, rule-based approaches require extensive domain knowledge and expertise to define valuable rules and thresholds [C, D, F, H, J, M, N, O]. Thus, setting up these rules is time-consuming, based on subjective insights, and might not detect future anomalies. For instance, participant [H] stated that one of their rules detected `sudo` operations. However, the rule did not specify `su` where they missed these anomalies. Participant [G] has experimented at his company with a self-implemented AI approach to identify and detect irregular patterns. Using K-means mainly allowed the company to reasonably detect anomalies in that scope, but their approach was not deemed production-ready due to shortcomings in their training dataset. Thus, to effectively use AI-based approaches, it is crucial to enhance dataset quality by understanding input parameters [G], allowing to maximize AI approaches' potential [F]. Additionally, participant [O] suggested that AI could optimize the dataset size by finding relevant monitoring parameters that do not introduce bias.

Takeaways RQ2: Anomaly Detection Approaches

While research mainly covers new implementations of AI algorithms for anomaly detection, companies use particularly rule-based or commercially available AI-based approaches. The main reasons for the interviewed companies using rule-based approaches are that they have lower computational costs, wider tool support, high adaptability to domain constraints through self-defined rules, and fast anomaly exposure. These rules, however, are tailored to each company and hinge on a deep understanding of the parameters and thresholds involved. Commercial AI-based approaches might offer more flexible solutions to identify patterns and anomalies in various contexts. Nonetheless, the dataset's quality limits the effectiveness of both rule-based and AI-based approaches.

RQ3: Runtime Monitoring Data for Anomaly Detection

As identified in RQ2, the dataset's quality is the foundation of a good anomaly detection approach where using key monitoring parameters and understanding their relationship helps avoid false positives. For instance, participant [B] illustrated that increased user requests resulted in slower response times. Yet, it should not be considered an anomaly as the relationship between these parameters remained unchanged. Therefore, this section elaborates on key monitoring parameters extracted from the systems' runtime data to identify anomalies. We categorized these parameters into **logs, traces,** and **metrics**. **Logs** are predefined semi-structured emitted messages that contain a specific timestamp, verbosity level (such as `INFO`, `WARN`, `DEBUG`, and `ERROR`), and unstructured natural language, including comprehensive information about an event specified by developers [C, F, H] [10]. A **trace** is a microservice-specific data type that represents the end-to-end execution of a single request, traversing through various microservices. A span is a trace segment encompassing more specific metadata (e.g., start and end time). **Metrics** consist of numeric time series data of system instances by collecting various performance data via, for instance, Prometheus, [10] [C, D, E, G, H, J, L]. These three primary monitoring data types can be measured via third-party software. For instance, participants rely on Heroku, Dynatrace, OpenShift, Nagios, Grafana, Kubernetes, Prometheus, Jaeger, GrayLog to collect, track and visualize their runtime monitoring data.

Regardless of the specific third-party software or monitoring data type, various parameters can be calculated from different monitoring data types. For instance, response time can be identified or calculated via logs, traces, or metrics [A, C]. Different errors are visible in logs via their verbosity level and in traces via their HTTP status [J]. Out-of-memory issues can be identified via logs or metrics [38].

We depict the three monitoring data types and their associated key monitoring parameters for anomaly detection in Figure 1, further elaborated in the following subsections.

1) Logs: Half of the participants look at logs to identify anomalous behavior to troubleshoot their applications, and one ([F]) solely relies on logs.

Due to the semi-structured characteristics of logs and the extensive volume [L] [37], [39], four participants [F, H, J, O] extract the **static part of logs** for further processing. The participants then either identify if this log was detected beforehand (true/false) [F] or look at the frequency within a time frame [H, O] [27], [28], [37], [39], [40]. For the frequency, participants [F, J, L] count the number of occurrences of a static part of the log. Participant [F] mentioned two ways to count it, either via event-based or sequence-based approaches. In his company, they rely on event-based counts where the static log gets counted when detected, which is feasible for their single-threaded processes where logs occur in their sequential order. Participant [L] chose a sequence-based approach because they had a multi-threaded application where the order of the logs may change. When only counting the occurrences as done with the event-based approach and without considering the time (fixed or sliding time frame), deadlocks, or the anomaly example [M-1], that dynamic scaling of the thread pool slowed down response times in Open Liberty might not get detected.

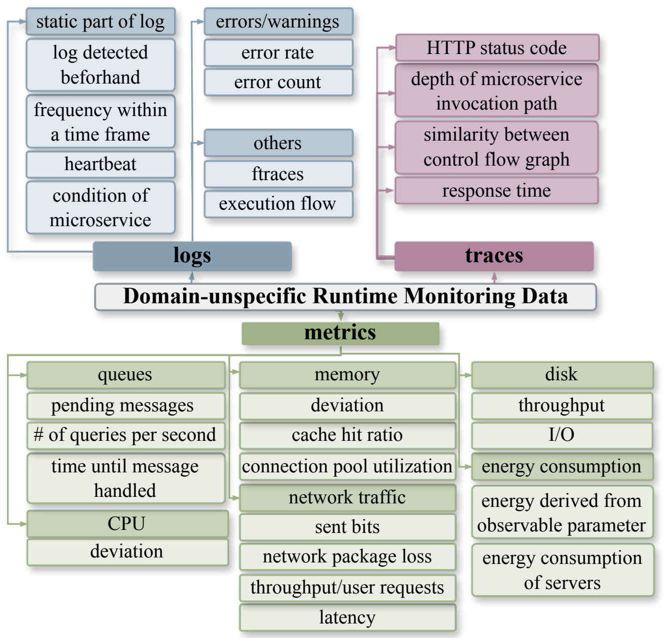

Fig. 1. Runtime monitoring data types categorization into logs, traces, metrics, and associated parameters.

In addition, thread names or their ID could be included in the log line [L].

However, when calculating the frequency, participant [F] mentioned that some parts of the software might not be used as regularly as others, resulting in different numbers of counts, where zero occurrences might also be feasible. Regular heartbeats are recorded to counteract overseeing an involuntary count of zero logs. Participant [O] further indicates that looking at the condition (e.g., running, idle) of a microservice via logs over time allows him to identify if the anticipated conditions occurred as expected.

Another common parameter extracted from logs is looking at **errors or warnings**. According to participants [E, G, J], error-specific parameters include error rate (calculated as the number of errors divided by the total number of occurred verbosity levels indicated in percentage) and the counted number of errors within a specific time frame [G, O]. Participant [J] does not differentiate the type of error when counting the occurrences. However, participant [F] states that they analyzed this parameter in the log message because the term *error* does not necessarily signify a system deviation (e.g., error handling routine started) as indicated in the anomaly description [F-2]. This example also shows that the absence of the English term *error* in the logs message does not indicate the absence of it. **Other** potential approaches involve extracting ftraces to understand kernel operations, especially since Participant [I] utilizes Programmable Logic Controller (PLC) as their programming language. Furthermore, logs can also capture the execution flow of inter and intra services similarly to traces where task or transaction IDs tie logs together [27], [28].

2) Traces: Participants [D, G, J, M, N] explicitly monitored microservice architectures and, therefore, could also rely on tracing information.

Similar to the error parameter extracted from logs, HTTP status code allows interview participants [A, H, J, N] and [10], [16] to extract the frequency of errors from endpoints, such as 500 (internal server error) or generate insights into the availability of services [26], such as 2xx (successful), or 4xx (client error).

For calculating further parameters, a control flow graph based on information regarding traces, microservices, and their relation to each other is recreated [D, J], where literature often specifies the graph as a directed acyclic graph [12], [15], [20], [21], [30], [41], [42]. However, participant [N] stated that generating the graph manually requires extensive system knowledge.

Based on this graph, parameters such as the **depth of the microservice invocation path** can be calculated [D, J]. Participant [J] stated that this parameter is essential when diving deeper into the root cause of an anomaly. For instance, more spans than anticipated might indicate an anomaly.

Industry papers revealed further parameters not identified during the interviews. For instance, one parameter summarises the **similarity between the control flow graph** of the microservices invocation pattern, such as if the in and outgoing dependencies remain the same for similar requests or microservices are missing within a trace [11], [43], [44].

Furthermore, the **response time** calculated via traces is an indicator for anomalous behavior [11], [20], [29], [30]. For instance, several industry papers look at the deviation of the response time when finding a matching invocation path [15], [20], [21], [27], [28], [30]. Response times can be measured for the whole request (calculated by the time difference between the incoming request and the application's response) [E, N] or for the processing time of each microservice [15], [21]. Furthermore, logs can also indicate a request and response where the time between these logs is measured [N]. Participant [K] also collects idle times within the response times and latency [12]. As in the anomaly example [F-1] with the exhausted heap space or example [B-1] with accumulated inodes, deviations in system-wide response time could be an early indicator of an accumulating anomaly and can prevent system failures as in the anomaly example [M-1]. If the response time increases, the load for one service might be too high [A, C]. The response times of external systems, such as a database and other systems, are of interest [E] [26].

3) Metrics: All participants mentioned several metric parameters. We assume metrics are mentioned that often because these are already available, are intuitive, monitored over time with a fixed interval, and do not require additional calculations. Furthermore, we assume that because all participants use rules and thresholds, they have more experience with metrics.

Several participants indicated that they observe different parameters regarding their **queues** [F, J, L]. For instance, in the anomaly example [J-1], a processing queue aggregated too many jobs, indicating an anomaly. Thus, participant [F] analyzed how many pending messages were in the queue and looked at the number of resolved queries per second within a queue [17]. Participants [A, E] use a similar metric but focus on the time it takes until the server handles the message in the queue (queue time).

In terms of CPU, a deviation or unusual pattern could signal an anomaly [C, F, H] [14], [16]. In the case of participants [C], a sudden or gradual increase may indicate a potential anomaly. However, a decrease in CPU utilization might also indicate an anomaly because participant [H] mentioned that their CPUs are almost 100% utilized in their normal production environment. The CPU-related parameter might further be split into CPU user usage, CPU system usage, CPU wait, or CPU throttling [H, K] [14]. However, participant [A] did not consider CPU usage because this signals when resource scaling is necessary.

When referring to **memory**, participants [C, D, J] and [14] stated that they also look at deviations of memory and memory dump. In the anomaly example [C-1], a memory leak from improper garbage collection was detected by monitoring memory consumption. Participant [D] monitors their sensor memory so that they do not allocate more resources during runtime than during testing. Kubernetes takes over the resource allocation via soft and hard thresholds where services might get restarted several times [D-1].

In addition, [H] indicated that the cache hit ratio, in combination with response time, gives insights into anomalous behavior. Participants [E, H, N] stated that they look at connection pool utilization and if there is an unusual amount of these connections, for instance, to a database service.

Participant [H] mentioned **network traffic**, where a change in the transmitted bits or network package loss might indicate an anomaly. Lee et al. [14] emphasized using network throughput, and two participants focused on increases in network traffic from increased user requests or timeout messages [J, L]. Participant [E] further identifies latency regarding the network or network throughput [14], [16].

Disk parameters are also a good indication for identifying a deviating behavior, as participants [F, H] indicated. For instance, [H] looks at disk throughput and disk I/O, such as I/O wait, idle, and the device read speed [14]. Participant [I] relies on Linux-based metrics measured via `iotop`, watching I/O usage information output by the Linux kernel to measure stress and increased load on hard disks.

During the literature review, we identified that **energy consumption** is not considered so far for anomaly detection. Thus, we explicitly asked our interview participants if they already employed this parameter in their anomaly detection approach. So far, one participant has used it to measure their hardware's energy consumption. However, they stated that they have not tried to measure their processors' energy consumption, for instance, with Running Average Power Limit (RAPL) [I]. Several participants also do not measure this type of metric but do believe that this might be a potential indication for anomalies and are eager to explore this additional parameter [C, D, F, K, L].

However, participants [G, H, I, J] argued that cloud-based deployment makes it impossible to physically measure energy consumption, and cloud providers do not provide these parameters. For instance, AWS or Google bills based on hourly usage without providing insights into energy consumption. Participant [D] also mentioned lacking tools to monitor server energy consumption.

Contrary, Participant [C] believes measuring energy could be feasible. Participants [D, M, N] suggested that already observable metrics, like CPU or memory, could assist with calculating this parameter. However, optimization strategies might distort the parameter when considering energy consumption. Thus, when an increase in energy outside of the threshold is monitored, a potential anomaly might have occurred.

Participants [K, L], however, suggested that considering server energy consumption could provide insights into how other systems affect the software under test. However, accurate measurement and interpretation are crucial, as participant [B] noted significant server and VM energy consumption fluctuations. Thus, it is challenging to map energy consumption to anomalies due to many influencing factors (e.g., energy use by unrelated monitoring tools on the same server) [B, H]. In addition, participant [C] noted modern solutions targeting energy reduction by shutting down unused applications and restarting them upon demand. Consequently, zero energy usage should not be considered an anomaly in such scenarios. In embedded systems, measuring energy is intricate due to the infinitesimally small units (e.g., Milliamperes) where temperature might influence energy consumption [O].

4) General Remarks: We recognized that the mentioned parameters do not differ based on the participants' domain, used approach, or whether they monitor a microservice or monolithic architecture [B, D, H, M, N]. A monolith can be seen as one microservice where logs or metrics, for instance, cannot be assigned to one specific service but are measured for the whole monolith (e.g., when deployed via a container [B]). Also, microservice-based systems can first look at system-wide parameters, diving deeper into more specific information for each microservice [D]. However, it is essential to note that orchestration tools, such as Kubernetes, might influence the system behavior and, thus, ultimately, the observed parameters due to different scheduling of resources or automated scaling [H]. Thus, looking at parameters in combination and their correlation is essential to avoid identifying an anomaly although the system performs as expected.

Moreover, as discussed by participants [H, I, K], an optimal sampling interval is crucial, avoiding excessive fluctuation when monitoring runtime data too frequently. Participant [H] and [26] note that a wide sampling interval might miss deviations, and participant [K] indicated that the interval must be chosen sensibly to prevent excessive data collection. Participant [M] added that the aggregation of parameters over a time frame (e.g., average, etc.) also influences how good anomalies are displayed.

Takeaways RQ3: Key Monitoring Parameters

Runtime monitoring data consists of three data types - logs, traces, and metrics. Figure 1 illustrates the key monitoring parameters to identify anomalies.

IV. Threats to Validity

This section discusses the Threats to Validity according to Wohlin et al. [45] and illustrates how we mitigated them.

We avoid a lack of **Internal Validity** by using data triangulation via a literature study and interviews to provide a definition on anomalies during runtime, identify anomaly detection approaches that base their insights on runtime monitoring data types and their respective parameters. Therefore, we extended the literature study by [18] and included 36 industry papers. To mitigate the papers‘ selection bias, a second author conducted a blind review as indicated in Section II-A. Furthermore, we gained insights into 12 companies via 15 interview participants, where we mitigated coding bias by discussing the generated codes and assignment thereof with two authors. However, the sample size consists of 15 participants, which might limit the generalizability of the findings. However, this study aims to gain in-depth insights and understand the participants' experiences and perspectives. The focus was on exploring detailed data rather than achieving statistical generalizability. We considered that this sample size could emphasize atypical responses, which we mitigated by providing the number of participants who made this statement. Furthermore, four interview participants have under five years of experience in this area, which might seem to render them less suitable. However, not only did the contacted company representative explicitly forward us these contacts, but also participants [G, H, K] focus on research regarding enhancing their companies' anomaly detection approach.

We enhanced the **External Validity** by selecting interview participants from different domains (Finance, Web, Hardware, Others), company sizes, and experiences. Although participants mentioned different root causes of anomalies, especially for the hardware domain, the observed anomalies and parameters were the same as in other domains. Thus, no significant differences in the anomaly detection strategies and parameters could be identified when analyzing a specific domain. In addition, the literature reviews case studies further enhance the applicability of anomaly detection approaches and parameters in various domains.

To minimize the threat to **Construct Validity**, we identified that numerous parameters could be derived from various runtime monitoring data types depending on the interview participants' monitoring strategies, complicating their classification. Thus, we provided detailed parameter descriptions with additional input when the parameter might get extracted from other monitoring data types and calculation methods to address this (e.g., response time calculated via logs or traces, errors identified via logs and traces).

Regarding **Conclusion Validity**, the obtained key runtime parameters are more difficult to extract from AI algorithms presented in published literature because the authors nearly always did not provide a dataset, code to the algorithm, or discuss their chosen input. Furthermore, AI-based approaches do not necessarily require dissecting the dataset into single parameters but entrust the AI model to make sense of the monitoring data. In addition, interview participants define rules on their parameters, providing more knowledge of key monitoring parameters for their anomaly detection.

V. Conclusion

Software systems must function reliably in industry, although anomalies are becoming more prevalent. Analyzing runtime monitoring data, including logs, traces for microservices, and metrics, is challenging due to the required excessive knowledge of the system and the huge amount of collected monitoring data. Detecting these unpredictable anomalies as soon as possible avoids more serious system failures. Therefore, it's crucial to understand anomalies in various domains, how they are detected in the industry, and which key monitoring parameters extracted from runtime monitoring data depict anomalies to enhance anomaly detection methods.

Therefore, we extended a literature survey, resulting in 36 relevant industry papers, and executed 15 semi-structured interviews across various domains. RQ1 identified that our industrial interview participants describe an anomaly as a deviation from expectations that has outliers of thresholds in monitoring data and negatively affects the company. These anomalies are also referred to as errors and failures where the interview participants provided us with several examples. Regarding RQ2, our analysis revealed that all twelve companies applied rule-based anomaly detection, whereas two companies developed an internal AI-based approach and four companies relied on commercial AI-based approaches. Notably, the existing literature predominantly focuses on AI-based approaches. For all approaches, a deep understanding of the key monitoring parameters and thresholds is essential to improve both anomaly detection approaches. Thus, based on RQ3, we provide a concise summary of key monitoring parameters (collected in Figure 1) extracted from the runtime monitoring data types.

As identified, it is essential to understand and improve the quality of the collected datasets by understanding key monitoring parameters, their relationships, and their influences on the system or microservice. Therefore, **future work** will establish a statistical model based on these parameters that provides insights into the parameters' relationship and necessity. Furthermore, cause-effect relations of an anomaly and the resulting system behavior should be calculated. To validate the explainability model, we will employ a benchmark system, TrainTicket [7], with injected anomalies in a controlled environment and a case study of real-world data comprising logs, Jaeger, and Prometheus data. Given energy consumption's potential as a new anomaly-detection parameter, we will consider *RAPL* and try to measure Watt with an external voltage metering tool. Based on the new insights, we can make informed decisions on their dataset and propose additional rules for detecting anomalies in the company.

References

[1] R. Kemmerer and G. Vigna, "Intrusion detection: a brief history and overview," *Computer*, vol. 35, no. 4, pp. supl27–supl30, 2002.

[2] B. Dornauer, M. Felderer, J. Weinzerl, M.-C. Racasan, and M. Hess, "Sohist: A tool for managing technical debt through retro perspective code analysis," in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 184–187. [Online]. Available: https://doi.org/10.1145/3593434.3593460

[3] I. S. Board, "Ieee standard glossary of software engineering terminology," *Office*, vol. 121990, p. 1, 1990. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=159342

[4] M. S. Islam, W. Pourmajidi, L. Zhang, J. Steinbacher, T. Erwin, and A. Miranskyy, "Anomaly detection in a large-scale cloud platform," *Proceedings - International Conference on Software Engineering*, pp. 150–159, may 2021.

[5] L. Mariani, M. Pezzè, O. Riganelli, and R. Xin, "Predicting failures in multi-tier distributed systems," *Journal of Systems and Software*, vol. 161, p. 110464, mar 2020.

[6] D. Samariya and A. Thakkar, "A comprehensive survey of anomaly detection algorithms," *Annals of Data Science*, vol. 10, pp. 829–850, 2023. [Online]. Available: https://doi.org/10.1007/s40745-021-00362-9

[7] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji *et al.*, "Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, feb 2021.

[8] M. Steidl, M. Gattringer, M. Felderer, R. Ramler, and M. Shahriari, "Requirements for Anomaly Detection Techniques for Microservices," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 13709 LNCS, pp. 37–52, 2022.

[9] F. Silva, V. Lelli, I. Santos, and R. Andrade, "Towards a Fault Taxonomy for Microservices-Based Applications," *ACM International Conference Proceeding Series*, pp. 247–256, oct 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3555228.3555245

[10] S. Zhang, P. Jin, Z. Lin, Y. Sun, B. Zhang *et al.*, "Robust Failure Diagnosis of Microservice System through Multimodal Data," *IEEE Transactions on Services Computing*, vol. 16, no. 6, pp. 3851–3864, feb 2023. [Online]. Available: https://arxiv.org/abs/2302.10512v2

[11] Z. Xie, H. Xu, W. Chen, W. Li, H. Jiang *et al.*, "Unsupervised Anomaly Detection on Microservice Traces through Graph VAE," *ACM Web Conference 2023 - Proceedings of the World Wide Web Conference, WWW 2023*, pp. 2874–2884, apr 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3543507.3583215

[12] Z. Xie, C. Pei, W. Li, H. Jiang, L. Su *et al.*, "From Point-wise to Group-wise: A Fast and Accurate Microservice Trace Anomaly Detection Approach," *ESEC/FSE 2023 - Proceedings of the 31st ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1739–1749, nov 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3611643.3613861

[13] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan *et al.*, "CloudRanger: Root cause identification for cloud native systems," *Proceedings - 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018*, pp. 492–502, jul 2018.

[14] C. Lee, T. Yang, Z. Chen, Y. Su, Y. Yang, and M. R. Lyu, "Heterogeneous Anomaly Detection for Software Systems via Semi-supervised Cross-modal Attention," *Proceedings - International Conference on Software Engineering*, pp. 1724–1736, feb 2023. [Online]. Available: https://arxiv.org/abs/2302.06914v1

[15] S. Zhang, Z. Pan, H. Liu, P. Jin, Y. Sun *et al.*, "Efficient and robust trace anomaly detection for large-scale microservice systems," *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pp. 69–79, 2023.

[16] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang *et al.*, "Practical Root Cause Localization for Microservice Systems via Trace Analysis," *2021 IEEE/ACM 29th International Symposium on Quality of Service, IWQOS 2021*, jun 2021.

[17] D. Liu, C. He, X. Peng, F. Lin, C. Zhang *et al.*, "MicroHECL: High-efficient root cause localization in large-scale microservice systems," *Proceedings - International Conference on Software Engineering*, pp. 338–347, may 2021.

[18] J. Soldani and A. Brogi, "Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey," *ACM Computing Surveys*, vol. 55, no. 3, p. 39, feb 2022. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3501297

[19] B. Li, X. Peng, Q. Xiang, H. Wang, T. Xie *et al.*, "Enjoy your observability: an industrial survey of microservice tracing and analysis," *Empirical Software Engineering*, vol. 27, no. 1, pp. 1–28, jan 2022. [Online]. Available: https://link.springer.com/article/10.1007/s10664-021-10063-9

[20] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen *et al.*, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, vol. 2020-Octob, pp. 48–58, oct 2020.

[21] M. Jin, A. Lv, Y. Zhu, Z. Wen, Y. Zhong *et al.*, "An Anomaly Detection Algorithm for Microservice Architecture Based on Robust Principal Component Analysis," *IEEE Access*, 2020.

[22] A. Ikram, S. Chakraborty, S. Mitra, S. K. Saini, S. Bagchi, and M. Kocaoglu, "Root Cause Analysis of Failures in Microservices through Causal Discovery," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[23] J. Chen, F. Liu, J. Jiang, G. Zhong, D. Xu *et al.*, "TraceGra: A trace-based anomaly detection for microservice using graph deep learning," *Computer Communications*, vol. 204, pp. 109–117, apr 2023.

[24] M. de Silva, S. Daniel, M. Kumarapeli, S. Mahadura, L. Rupasinghe, and C. Liyanapathirana, "Anomaly Detection in Microservice Systems Using Autoencoders," *4th International Conference on Advancements in Computing, ICAC 2022 - Proceeding*, pp. 488–493, 2022.

[25] M. V. Mäntylä, M. M. Fi, and Y. Wang, "LogLead – Fast and Integrated Log Loader, Enhancer, and Anomaly Detector," nov 2023. [Online]. Available: https://arxiv.org/abs/2311.11809v2

[26] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "AutoMAP: Diagnose Your Microservice-based Web Applications Automatically," *The Web Conference 2020 - Proceedings of the World Wide Web Conference, WWW 2020*, pp. 246–258, apr 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3366423.3380111

[27] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng, and J. Xu, "LogSed: Anomaly Diagnosis through Mining Time-Weighted Control Flow Graph in Logs," *IEEE International Conference on Cloud Computing, CLOUD*, vol. 2017-June, pp. 447–455, sep 2017.

[28] T. Jia, P. Chen, L. Yang, Y. Li, F. Meng, and J. Xu, "An Approach for Anomaly Diagnosis Based on Hybrid Graph Model with Logs for Distributed Services," *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, pp. 25–32, sep 2017.

[29] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection and classification using distributed tracing and deep learning," *Proceedings - 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2019*, pp. 241–250, may 2019.

[30] "Anomaly detection from system tracing data using multimodal deep learning," *IEEE International Conference on Cloud Computing, CLOUD*, vol. 2019-July, pp. 179–186, jul 2019.

[31] P. Ralph, N. bin Ali, S. Baltes, D. Bianculli, J. Diaz *et al.*, "Empirical Standards for Software Engineering Research," oct 2020. [Online]. Available: https://arxiv.org/abs/2010.03525v2

[32] M. Steidl and B. Dornauer, "Replication Package - How Industry Tackles Anomalies during Runtime: Approaches and Key Monitoring Parameters," May 2024. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.10637562

[33] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, apr 2009. [Online]. Available: https://link.springer.com/article/10.1007/s10664-008-9102-8

[34] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," *Proceedings - International Software Metrics Symposium*, vol. 2005, pp. 10–23, 2005.

[35] OECD, *Entrepreneurship at a Glance 2017*. OECD, 9 2017.

[36] S. E. S. C. of the IEEE Computer Society, "Ieee std 1044-2009 (revision of ieee std 1044-1993), ieee standard classification for software anomalies," 2010.

[37] M. Catillo, A. Pecchia, and U. Villano, "AutoLog: Anomaly detection by deep autoencoding of system logs," *Expert Systems with Applications*, vol. 191, p. 116263, apr 2022.

[38] J. Huang, Y. Yang, H. Yu, J. Li, and X. Zheng, "Twin Graph-Based Anomaly Detection via Attentive Multi-Modal Learning for Microservice System," *Proceedings - 2023 38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023*, pp. 66–78, 2023.

[39] M. Cinque, R. Della Corte, and A. Pecchia, "Micro2vec: Anomaly detection in microservices systems by mining numeric representations of computer logs," *Journal of Network and Computer Applications*, vol. 208, p. 103515, dec 2022.

[40] H. Shan, Y. Zhang, Y. Chen, X. Xiao, H. Liu *et al.*, "ϵ-Diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms," *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019*, pp. 3215–3222, may 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3308558.3313653

[41] L. Meng, F. Ji, Y. Sun, and T. Wang, "Detecting anomalies in microservices with execution trace comparison," *Future Generation Computer Systems*, vol. 116, pp. 291–301, mar 2021.

[42] T. Wang, W. Zhang, J. Xu, and Z. Gu, "Workflow-Aware Automatic Fault Diagnosis for Microservice-Based Applications with Statistics," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2350–2363, dec 2020.

[43] R. Ding, C. Zhang, L. Wang, Y. Xu, M. Ma *et al.*, "TraceDiag: Adaptive, Interpretable, and Efficient Root Cause Analysis on Large-Scale Microservice Systems," *ESEC/FSE 2023 - Proceedings of the 31st ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1762–1773, nov 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3611643.3613864

[44] A. Bento, J. Correia, R. Filipe, F. Araujo, and J. Cardoso, "Automated Analysis of Distributed Tracing: Challenges and Research Directions," *Journal of Grid Computing*, vol. 19, no. 1, pp. 1–15, mar 2021. [Online]. Available: https://link.springer.com/article/10.1007/s10723-021-09551-5

[45] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

VI. Response letter to the reviewers

Thank you for your concrete and constructive feedback, which we used to enhance and improve our paper. We took all comments of the meta-review, as well as the comments from each reviewer, into consideration and adapted the paper as follows:

A. add background

Due to the constraints of the paper's length, we added more background information directly into the introduction section. Therefore, we provided more information as follows:

- what software anomaly detection during runtime is
- added an explanation why a consensus regarding interpretation, characteristics, and examples of anomalies is important
- additional related work that identifies algorithms for anomaly detection
- details about the runtime monitoring data (logs, traces, metrics)

Thus, in total, we now include information regarding the need for anomaly detection within the industry, what anomaly detection is, a description of approaches that allow developers and operators to detect software anomalies, and a description regarding runtime monitoring data (logs, traces, and metrics). However, this should just give an overview and emphasize the need to define anomalies answered in RQ1, the rationale to understand why rule-based or AI-based approaches are used in industry in RQ2, and the different parameters extracted from runtime monitoring data set in RQ3.

B. provide details on the literature review

We provide further details about the literature review by ...

- describing why we have chosen the paper by Soldani and Brogi [18] as our base for extending the survey by our literature review
- providing information on why the literature study stretched over 8 months

C. add interview questions (e.g., as link to online material)

We further enhanced our replication package [32] with the following information:

- Further details about the interviews (participant selection, interview guidelines, interview questions).
- Description regarding the applied purposive sampling strategy
- Coding Schema and colored highlighting for our data extraction

D. clarify the description of your research method/experimental design: the process of selecting companies and how results were analyzed and related (interviews vs. literature)

Based on your feedback, we described ...

- the identification of companies and respective participants via purposive sampling where we wanted to balance SME and large companies, microservice architectures, and various domains. We stated that for this study the goal was to gain qualitative insights rather than statistical inference.
- the data extraction. Therefore, we added a subsection to the Study Design where we specifically presented our data extraction, once for RQ1 and RQ2, via inductive coding because the main goal was to derive codes from the data without having any predefined codes that could limit the findings and interpretations of the interview participants' statements. For further visibility and understanding of how the results were obtained, we added the color schemes for the codes in the replication package as well. For RQ3, we elaborated on how we derived the codes from interviews and literature and combined these findings.

E. add threats to validity related to participant selection and sample size

We further considered your valuable remark about potential biases due to the sample size. Therefore, we specified that statistical generalizability was not the main goal but to achieve a thorough understanding of the participants' experiences and perspectives. Furthermore, we considered that this sample size could emphasize atypical responses, which we mitigated by stating the number of participants who made this statement. In addition, we stated why four participants have less than five years of experience and that we did not want to exclude them because not only did the people we contacted from the company highly recommend talking to these people, but also three participants were tasked to enhance their companies' anomaly detection approach.