Autonomous Monitors for Detecting Failures Early and Reporting Interpretable Alerts in Cloud Operations

Adha Hrusto*
Lund University, Department of Computer Science
Lund, Sweden
System Verification Sweden AB
Malmö, Sweden
adha.hrusto@cs.lth.se

Per Runeson
Lund University, Department of Computer Science
Lund, Sweden
per.runeson@cs.lth.se

Magnus C Ohlsson
System Verification Sweden AB
Malmö, Sweden
magnus.c.ohlsson@systemverification.com

ABSTRACT

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

CCS CONCEPTS

• **Computing methodologies → Machine learning algorithms**; *Information extraction*; • **Software and its engineering → Operational analysis**; **Software defect analysis**; • **Information systems →** *Computing platforms*.

KEYWORDS

cloud, monitoring, anomaly detection, failures

ACM Reference Format:
Adha Hrusto, Per Runeson, and Magnus C Ohlsson. 2024. Autonomous Monitors for Detecting Failures Early and Reporting Interpretable Alerts in Cloud Operations. In *46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3639477.3639712

1 INTRODUCTION

Maintaining cloud-based system operations free from failures is increasingly challenging, considering the complexity of software architectures and the wide variety of cloud deployment options used for developing and operating modern software systems. Streamlining the software development cycles by applying agile and DevOps practices partially addresses this ever-rising concern. Since modern applications rely heavily on underlying cloud infrastructures, managing and monitoring corresponding cloud resources has become essential in providing resilient large-scale software systems. Thus, cloud operations (CloudOps) [2] is another critical component of digital transformation together with DevOps that helps provide more reliable software systems. Reliability can be achieved with run-time monitoring of key performance indicators (KPI) and timely detection of deviations that signalize either a software bug or performance degradation of the cloud platform.

One of the widely used approaches for detecting abnormalities in monitoring data is *anomaly detection* [4]. Early detectability of observations that significantly deviate from the normal pattern might be crucial for preventing the roll-out of severe software failures. Further, an anomaly detection service that provides early warning of impending failure can reduce the costs associated with reliability and downtime [23]. To tackle this, many software development companies nowadays rely on commercial monitoring tools [25] for tracking and visualizing monitoring data, while reporting anomalies as *alerts* via email or communication platforms (e.g., MS Teams or Slack). This is often implemented with manually set alert rules or anomaly detectors provided by the monitoring tool. The approach, however, risks causing information overload for operators due to the diversity and abundance of metric indicators and detection mechanisms that can usually be configured only for single metrics (univariate data). Additionally, generated alert notifications are often not intuitive as they lack clarity and actionable information, making it difficult for operators to understand the underlying issues. This characteristic, which we refer to as *interpretability*, may be enhanced by providing more clear and concise insights on how to approach raised alerts.

Our recent paper [5] partially deals with this concern, where we proposed and implemented an approach for optimization of anomaly detection in multivariate data. The solution based on *LSTM autoencoders* is validated in the context of a microservice-based system. In this paper, we extend the scope of validity by collaborating with a new *case company* and implementing the solution in a new industrial context. The case company is responsible for developing and operating a Product Information Management (PIM) system using Microsoft Azure tools and services. The new system under study has a different software architecture but is distributed in nature and is cloud-based. Moreover, inspired by the latest advancements in the machine learning area, we enhance our solution with an additional deep learning (DL) model based on transformers [27] to be able to capture broader temporal trends in data.

The two DL models, which we refer to as *LSTMAE* and *TRANAE*, are the main part of the *autonomous monitors*, i.e., the cloud solution used for near-real-time monitoring and detecting deviations in metrics, numerical observations of specific aspects of Azure resources. In addition to metrics, logs can provide significant insights into various activities and events in operations. However, the large amount of available logs is not usually favorable for in-depth analysis, considering their complex structure. Further, existing approaches are not applicable to different contexts due to a lack of standardized logging procedures [3]. Instead of combining metrics and log features together for anomaly detection [10, 19], we stick with only metrics analysis while leveraging the pre-trained large language model GPT-3 [17] to support the interpretation of logs and to generate interpretable alerts.

The solution approach was inspired by the practices of the CloudOps teams and the way they were handling and resolving alerts. Detecting deviations in metrics was the first line of defense and then digging deeper into the logs was the next step to find clues for root cause analysis. We designed the autonomous monitors to mimic this behavior by using two DL models to simultaneously detect discrepancies in metrics and the GPT-3 model [17] to give suggestions for root cause analysis, which is crucial for preventing the roll-out of severe failures. The output of the autonomous monitors is an interpretable alert notification, providing a detailed status report, to which we refer as a *smart alert*.

Contributions of the paper are threefold:

- **C1:** A novel approach for monitoring metrics and reporting *interpretable* alerts based on two state-of-the-art anomaly detection DL techniques and the GPT-3 language model, which is evaluated in a real-world setting.
- **C2:** Implementation and evaluation of a deep transformer network [27] in an industrial context, which was previously evaluated only on publicly available datasets.
- **C3:** Case-based generalization of two case studies resulted in assessing opportunities for *smart alerts* in multiple contexts.

2 BACKGROUND AND RELATED WORK

Commercial cloud providers gained tremendous popularity in the last decade, especially among software development companies that seek scalable and flexible platforms for the deployment of their software. This means that such cloud resources can be dynamically allocated to accommodate varying workloads, providing enhanced performance when needed. However, cloud infrastructures are not immune to failures due to increasing computing demands, growing complexity of system architectures, and cost-effective but low-quality commodity hardware used in cloud data centers [15]. Consequently, software systems relying on these cloud resources are also at risk of underperforming in operations [26]. Run-time failures in cloud-based software systems can be identified on the *application level* (e.g., service unavailability or slow response times), *platform level* (e.g., errors in deployment configuration settings), or *infrastructure level* (e.g., physical servers within a data center fail) [25]. This multilayer failure source makes it even more challenging to timely identify and resolve any potential threat to performance and reliability degradation [4].

Continuous monitoring is thus of high importance for tracking the health status of each system component in the cloud. Tamburri et al. [25] conducted a survey in over 70 different organizations, searching for monitoring practices in the industry. The key findings revealed that software development organizations are not fully aware of the potential benefits of monitoring data analysis, while 90% of respondents identified that one of the main challenges is the lack of standardization in the realm of monitoring tools. This implies that many organizations use a minimum of two or more different monitoring tools as they struggle to find a unified tool to fulfill all their needs regarding collecting, transferring, processing, storing, and visualizing monitoring data [25]. The same study reports that most adopted monitoring tools include Google Analytics, Nagios, Grafana, Amazon CloudWatch [24], Microsoft Azure Monitor [5], and ELK (Elastic, Logstash, Kibana) [8]. Many of these tools include some alert mechanism for current or forecasted observations with predefined thresholds [28].

Existing approaches for detecting failures based on monitoring data are mainly inspired either by anomaly- or signature-based strategies [15]. Signature-based approaches identify behavior associated with specific faults, aiming to detect similar faults during runtime, while anomaly-based approaches identify normal system behavior and aim to detect deviations from these norms. Since faulty behavior is unknown and hard to reproduce in many industrial contexts, anomaly detection approaches gained wider acceptance and usage. Thus, in this paper, we further focus only on anomaly detection techniques, designed for unlabeled data and based on machine learning.

The metrics, as an essential part of monitoring data, are usually represented as a time series, a sequence of data points recorded at specific time intervals. For this type of data, many anomaly detection techniques were implemented and evaluated in the context of cloud computing systems [6, 7, 13, 21]. However, as time series are widely used to model data in various domains (e.g., finance or healthcare), a number of approaches for anomaly detection were also proposed and evaluated on publicly available datasets or benchmarks. Some of the approaches that gained wide popularity include autoencoders [14], deep convolutional neural network (CNN) [16], GANs [12], and lately very attractive transformers-based networks [27]. In practice, the choice of which technique to use depends on the specific characteristics of the data and anomaly detection tasks. Often, a combination of these methods or hybrid models can yield better results. Logs, as the other significant part of monitoring data, were also of interest for analysis and detecting anomalies, standalone or in combination with metrics [11, 19].

There is still a significant number of anomaly detection approaches that were not evaluated in industrial contexts of cloud-based systems. Thus, we take that opportunity to deploy *TRANAE* in operations and integrate it into the monitoring platform of the system under study. Additionally, we implement a custom monitor as we want to perform inference periodically and report interpretable alerts, which provide more valuable information than just a binary value, usually provided by inbuilt anomaly detectors within a monitoring platform [28].

3 RESEARCH APPROACH

Similar to our previous work [5], we use the design science paradigm [20] to frame our research at the case company as it aims to *improve an area of practice*. The main contributions of such research are *technological rules*, i.e., prescriptive recommendations for practice. Since they are context dependent, our goal in this study is to extend the validity of already validated prescription for practice, which maps the challenge of *alert management* to the solution for *anomaly detection in multivariate data* [5]. By alert management, we refer to any activity related to reporting, interpreting, or resolving alerts. Insufficient engagement in regard to these activities will eventually cause *alert flooding* and burden operations teams. To avoid this undesirable consequence, we explore possibilities of improving alert management in a new industrial context while also considering the latest scientific contributions in the area of ML to advance the solution design for more efficient monitoring.

To reach our goals, we conducted four main activities of the design science cycle [20]:

- *Problem conceptualization* — By closely collaborating with the DevOps and CloudOps teams, which included interviews and observations of their practices, we managed to confirm that alert management and consequently alert flooding was one of the main issues in operations. Thus, we continued with a more detailed analysis of monitoring data used for designing the alert mechanism. Surveying literature and previous experience in similar studies enabled the formulation of problem constructs. Metrics and logs representing multivariate time series and a chronological record of events from operations were identified as critical data for which we needed to envision a matching solution.
- *Solution design* — We design a solution for monitoring and reporting interpretable alerts by replicating the current practices of the team and creating an automated workflow that consists of *monitoring*, *detection*, and *interpretation*. The monitoring part includes pulling the data from different Azure resources periodically and creating a feature vector, formatted for use with DL models. Following the latest advancements in the machine learning world, we use deep *transformer network* [27] for anomaly detection, simultaneously with *LSTM autoencoders*, which was the only DL model in our previous work [5]. Since logs were used by the CloudOps teams in later stages to investigate detected anomalies, similarly, we employ the GPT-3 model [17] to analyze and interpret the logs whenever an anomaly is detected. These procedures of monitoring, detection, and interpretation of alerts are integrated into the monitoring system, which we refer to as the *autonomous monitors*.
- *Instantiation* — The proposed solution design of the autonomous monitors is implemented using Microsoft Azure services since the system under study is also developed and operated using the same cloud platform. We leveraged solutions for registering and deploying ML models, building and deploying event-driven code, and automating workflows.
- *Empirical validation* — For validation of implemented solutions, we used feedback from DevOps and CloudOps teams on reported alert notifications in the MS Teams channel. We draw conclusions by analyzing qualitative data collected in a survey using questionnaires and quantitative data representing the performance of the ML models for anomaly detection.

4 PROBLEM CONTEXT

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

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

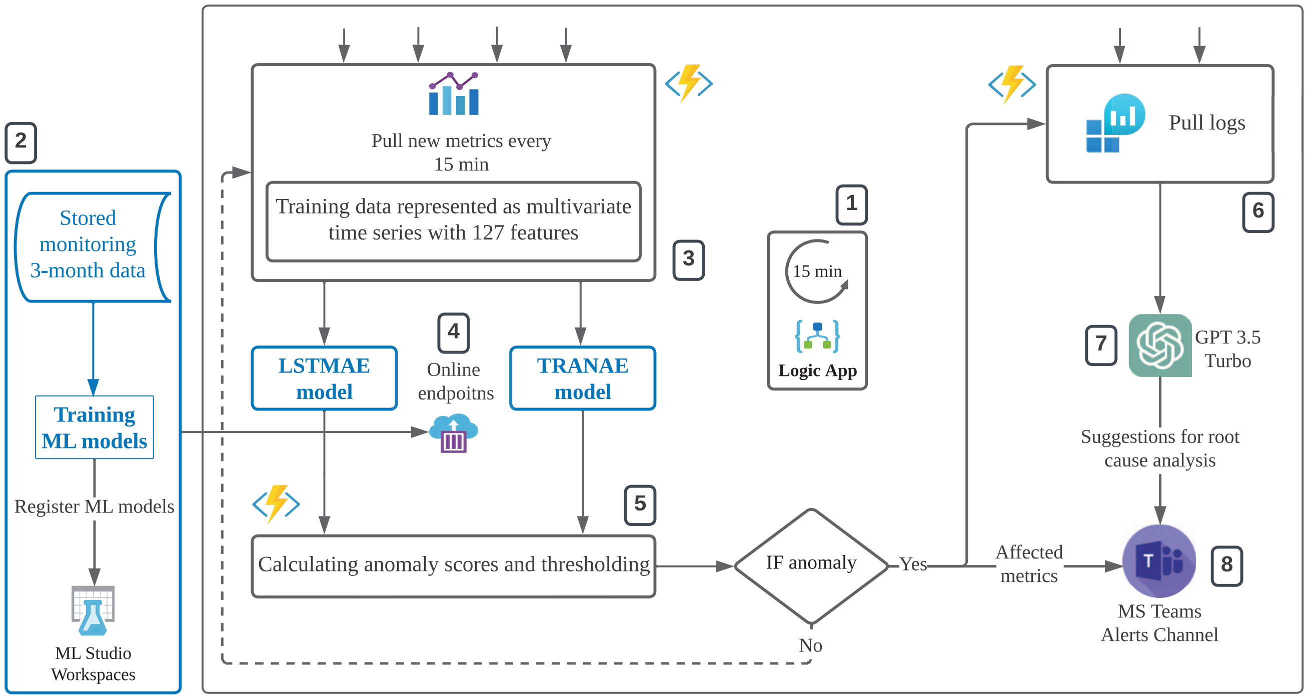

2
Stored monitoring 3-month data
Training ML models
Register ML models
ML Studio Workspaces
Pull new metrics every 15 min
Training data represented as multivariate time series with 127 features
3
LSTMAE model
4
Online endpoitns
TRANAE model
5
Calculating anomaly scores and thresholding
IF anomaly
Yes
No
1
15 min
Logic App
Pull logs
6
7
GPT 3.5 Turbo
Suggestions for root cause analysis
Affected metrics
8
MS Teams Alerts Channel

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

5 AUTONOMOUS MONITORS

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

5.1 Implementation

The main goal of the autonomous monitors shown in Figure 1 is to retrieve a preselected set of metrics periodically and utilize ML models for anomaly detection to detect deviations that significantly deviate from the normal pattern. The most critical metrics are identified with the Peak Over Threshold (POT) method [22], used for learning threshold values for anomaly scores. In case of an anomaly, the next step in the workflow is pulling logs and predicting possible root causes with a GPT-3 language model. The final output of the autonomous monitors is an interpretable alert notification, shown in Figure 2. We refer to this step as *instantiation*, as described in Section 3. The detailed description of each step in the workflow shown in Figure 1 is given below.

(1) **Automated workflow** — The overall solution design of the proposed autonomous monitors is implemented using a Logic App in Azure, a cloud-based service for creating, scheduling, and automating workflows. With its inbuilt *triggers* and *actions*, it is possible to define the condition or frequency of the execution and implement and connect each step.
(2) **Training ML models** — We used 3-month data of metrics for training both *LSTMAE* and *TRANAE* models. A more detailed overview of data and preprocessing is explained in Section 5.1.1. Considering the size of this training dataset, we decided to perform training on our local machine. However, in the case of larger datasets and more complex ML models, the training can be performed on some cloud computing resources. The models were developed in Python using *Pytorch* library [18]. *LSTMAE* was developed by authors while *TRANAE* is an open source solution developed by Tuli et al. [27]. After the training, the models were registered in Azure ML Studio Workspace, for easier managing and deploying.
(3) **Pulling metrics** — In agreement with the team in the case company, we configured the frequency of the automated workflow to be fifteen minutes. The first triggered step is the execution of the four Function Apps, responsible for fetching selected metrics. Azure Functions are lightweight cloud solutions for building and deploying code written in various programming languages. Thus, we developed and deployed Python scripts for accessing metrics via Azure Monitor Metrics and Query APIs and pulling all required data from Application Insights and Log Analytics.
(4) **Deploying ML models** — Both models, *LSTMAE* and *TRANAE* are deployed as online endpoints in Azure ML Workspace.

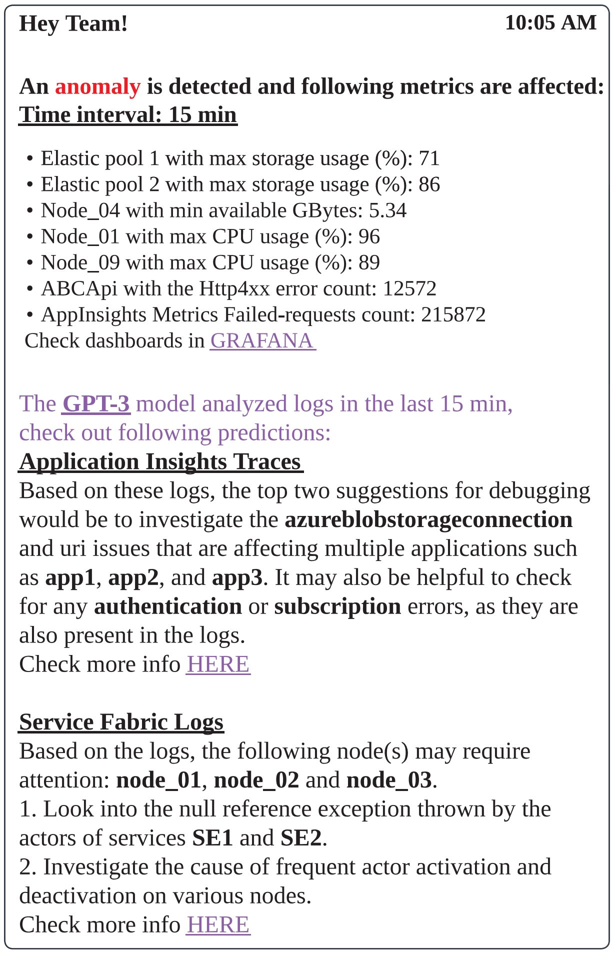
In this way, the models are accessible for real-time inference via REST APIs and can be seamlessly integrated into production workflows. We consumed deployed endpoints by sending the HTTP POST request to the endpoint's URL and providing the necessary input data. The request body included the feature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

(5) **Thresholding** — We used different approaches for each model to determine if reconstructed losses are anomalous. For the *LSTMAE* model we calculate anomaly likelihood as suggested by Islam et al. [6] and Ahmad et al. [1]. For the *TRANAE* model, we use the statistical method, Peak Over Threshold (POT), as it was originally proposed by Tuli et al. [27]. This method is used in extreme value analysis, and it's particularly valuable for detecting rare or extreme values, that are unusually high or low.

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

(7) **GPT-3 predictions** — We leveraged the latest advancements in the ML world by using the pre-trained language model GPT-3 [17] (GPT-3.5 Turbo) for the logs analysis. The main goal was to discover some implications of the identified anomaly in metrics and predict the clues for root cause analysis. The model was deployed within the Azure AI Studio. The input to the model was structured as a series of messages formatted for API communication. The first message contained the 15-minute history of log data, while the second message included a query asking the model to suggest the top two debugging strategies based on the logs, specifically highlighting the affected applications.

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



Hey Team! **10:05 AM**

An anomaly is detected and following metrics are affected:
Time interval: 15 min

- Elastic pool 1 with max storage usage (%): 71
- Elastic pool 2 with max storage usage (%): 86
- Node_04 with min available GBytes: 5.34
- Node_01 with max CPU usage (%): 96
- Node_09 with max CPU usage (%): 89
- ABCApi with the Http4xx error count: 12572
- AppInsights Metrics Failed-requests count: 215872

Check dashboards in GRAFANA

The **GPT-3** model analyzed logs in the last 15 min, check out following predictions:

Application Insights Traces
Based on these logs, the top two suggestions for debugging would be to investigate the **azureblobstorageconnection** and uri issues that are affecting multiple applications such as **app1**, **app2**, and **app3**. It may also be helpful to check for any **authentication** or **subscription** errors, as they are also present in the logs.
Check more info HERE

Service Fabric Logs
Based on the logs, the following node(s) may require attention: **node_01**, **node_02** and **node_03**.
1. Look into the null reference exception thrown by the actors of services **SE1** and **SE2**.
2. Investigate the cause of frequent actor activation and deactivation on various nodes.
Check more info HERE

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

5.1.1 Data. As described in Section 4, the training set consists of metrics that we sorted into four categories. The total number of features considering all categories is 127, where a detailed overview of each category is presented in Table 1. The training data set consists of 8640 samples recorded every fifteen minutes within three months. Some of the features contained null entries, thus we replaced them with zeros to maintain data structures and avoid bias. The next necessary step before the training process is to equalize the scales of features to prevent domination of one of them during the learning process. For the *LSTMAE* model, we used *Scikit-Learn's* normalizer, which performs row-wise normalization on a dataset, independently scaling each row to have a specified norm, typically L1 or L2. For *TRANAE* model, we used a two-step normalization process [27]. Firstly, the data is scaled to a range [-1, 1] by dividing each element (feature) by the maximum absolute value across columns. Then, these values are scaled again to the range [0, 1] by dividing by 2 and adding 0.5. The goal of both approaches is to bring all features on a common scale, allowing ML algorithms to work more effectively and produce reliable and accurate results.

5.1.2 ML models. The *LSTMAE* model, presented in our recent work [5], consists of two components built of LSTM (Long Short Term Memory) stack of layers which makes them suitable for capturing temporal dependencies. However, potentially, this can be

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

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

computationally intensive for longer sequences. From the two components, the encoder compresses the input sequence into a fixed-size latent representation while the decoder reconstructs the original sequence from this representation. In this way, the encoder filters out noise while retaining the most relevant information and learning the *normal* model. The entire *LSTMAE* is implemented in Python using *Pytorch*, a library for building and training deep neural networks. By adjusting multiple parameters in several consecutive training runs, the lowest training loss (0.0049) is reached with input dimension 512, latent dimension 4, and 0.01 as a learning rate (see Appendix B).

The *TRANAE* model, developed by Tuli et al. [27], is a deep transformer network with adjusted architecture for the task of anomaly detection in multivariate time series. It consists of two encoders and two decoders and leverages attention mechanisms for capturing complex temporal trends. In the first of the two adversarial training phases, the model aims to generate an approximate reconstruction of the input window. Deviations between this initial reconstruction and the actual input are used as a focus score in the second phase. Thus, this focus score is used to modify attention weights, giving higher importance to specific sub-sequences in the input data to extract short-term temporal trends. In this way, the *TRANAE* model is able to amplify deviations and capture anomalies more effectively. A more detailed description of the model is presented in the original paper [27]. The model was trained using the same environment, with the same programming language and libraries as *LSTMAE* model. We used the original architecture of the *TRANAE* model and varied only the number of epochs and learning rate during several training runs. We reached the convergence and the training loss of 0.0073 with the 10 epochs while keeping the learning rate at 0.0001.

The choice between the LSTM autoencoders and transformer networks often depends on the training data distribution and the complexity of the anomaly detection task. In this study, we evaluate both models for detecting anomalies in multivariate metrics representing monitoring data of the cloud-based distributed system. Since the case company could not provide the ground truth data, the performance of models was only partially evaluated prior to deployment. For in-depth evaluation, the operations teams needed to review each reported alert and respond to the questionnaire regarding the overall functionality of the autonomous monitors. The results are elaborated in the following section.

5.1.3 *Detecting anomalies.*

We calculated the anomaly likelihood for *LSTMAE* model as suggested by Islam et al. [6] and Ahmad et al. [1]. Thus, we maintain a record of reconstruction errors over short-term (W') and long-term (W) intervals and monitor how their mean and standard deviation change over time. Consequently, we calculate the likelihood of an anomaly occurring at time t using the formula $A_t = 1 - Q(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}), A_t \in (0, 1)$ [1]. In this equation, Q represents a Gaussian tail probability [9], $\tilde{\mu}_t$ represents the mean of the short-term interval, while μ_t and σ_t denote the mean and standard deviation of the long-term interval, respectively. If A_t is greater than or equal to $1 - \epsilon$, then the observation at time t is classified as an anomaly. Given that the approximate contamination in the training data is 5%, we set ϵ to be 0.05. The long-term interval (W) is set to 30, corresponding to 450 minutes ($30 * 15$), while the short-term interval (W') is set to 3, corresponding to 45 minutes ($3 * 15$). The choice of interval lengths is based on the duration of glitches and significant performance deviations in operations. We aim to investigate the effect of shorter window lengths (15 minutes) as a part of our future work, as well as using a smaller time granularity (5 minutes) for data samples.

For the *TRANAE* model, anomalies are detected using the POT statistical method, used in extreme value analysis to assess and model rare events that exceed a certain threshold. Thus, only data points exceeding a predetermined threshold value are analyzed and used to estimate the distribution of extreme values. More specifically and as proposed by Siffer et al. [22], the POT method fits Generalized Pareto Distribution (GDP) to the excesses, the difference between the observed value and threshold. In the initialization step of this method, we assumed that the excesses can be identified in the 5% of the data, which corresponds to the initial assumption about the contamination of the training data set. Thus, the *initial threshold* equals 95% of the maximum value observed within each feature. The output of this step is the first threshold z_q, computed based on the fitted distribution of excesses and fixed risk q. The objective here is to calculate the threshold value z_q such that the probability of the observed value exceeding it is smaller than q. For this step, we used the reconstruction losses from the training dataset. Further, we used the losses from the test data set to update the distribution parameters and the current threshold z_q if new values (test losses) exceed the initial threshold t. The value for the fixed risk q is determined experimentally after executing multiple runs of the POT method and trying to match the number of detected anomalies in the test data set to predetermined contamination (5%). By using this method, we managed to compute thresholds for reconstruction losses obtained for each of the 127 features, as well as the threshold for the mean value of losses across the columns (features).

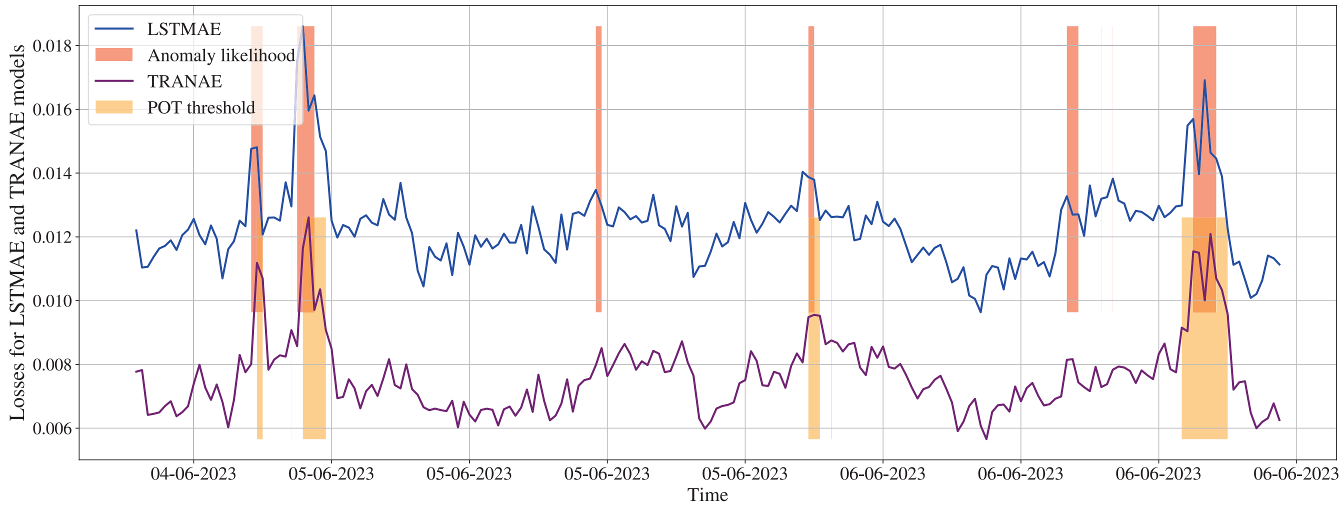


Figure 3: Anomaly detection with models *LSTMAE* and *TRANAE*: Regions in red show anomalies detected with *LSTMAE* and anomaly likelihood, while regions in orange show anomalies detected with *TRANAE* and the POT method

5.2 Evaluation

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

5.2.1 Pre-deployment evaluation. A shortage of ground truth data prevented the evaluation of both models before deployment. Nevertheless, we compared the model outputs and corresponding thresholding methods on the subset of the dataset used for testing, which included 864 (10%) samples. Reconstruction losses calculated on the test data for each model are shown in Figure 3, where red and orange shading corresponds to data points showing anomalies. Anomalies detected with *LSTMAE* model and by calculating anomaly likelihood are highlighted in red, whereas those detected using the *TRANAE* model and POT method are highlighted in orange. The two reconstruction curves shown in Figure 3 appear very comparable, following similar patterns in certain regions with the MSE (mean square error) of $2.25 \cdot 10^{-5}$. However, due to different models' architecture, weights, and biases, the *LSTMAE* loss has a slightly higher amplitude than the *TRANAE* corresponding values. Even though reconstruction losses look alike, we could not deduce whether each of them has better reconstruction accuracy and thus better captures anomalous samples. Moreover, we used different methods for thresholding and detecting anomalies, which resulted in anomalous regions that differ to some degree. As shown in Figure 3, there is an overlap between four shaded anomalous segments detected by both approaches. Additionally, the *LSTMAE* and anomaly likelihood detected two isolated anomalous events, which were not detected with the fixed threshold obtained with the POT method. The anomaly likelihood method better adapts to changes in the data distribution over time, making it potentially more robust to varying anomalies. However, depending on the window sizes, this approach may introduce some delay in detecting anomalies compared to fixed thresholding. The POT method could eventually update the threshold based on newly detected anomalies in the time series. This would require computing the threshold in real-time, and due to its complexity, we decided to follow the original implementation [27]. By setting the threshold parameters to detect approximately 5% anomalies in streaming time series (metrics), we approached the evaluation phase, where the operations team helped with the evaluation process.

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

The main criterion for reviewing reported alerts was the applicability of GPT-3 predictions to underlying issues based on log inputs. Even though both the *LSTMAE* and *TRANAE* models would identify a deviation in metrics, this perceived applicability of GPT-3 predictions ultimately determined whether the detected event was an anomaly or just a temporary glitch. The total number of detected anomalies is 26, where 17(65.4%) were detected by *LSTMAE* and 20(76.9%) of them by the *TRANAE* model, with an overlap of 11 alerts. Temporary glitches are identified whenever there was a vague or recurring prediction by GPT-3. As shown in Figure 4, these false positive predictions appear in most cases before or after actual anomalies, even though there were some single events. This indicates that the thresholds need to be adjusted upward to meet the requirements of the case company. However, a positive observation is that an anomalous event was always detected by one of the deployed models, where the *TRANAE* model demonstrated

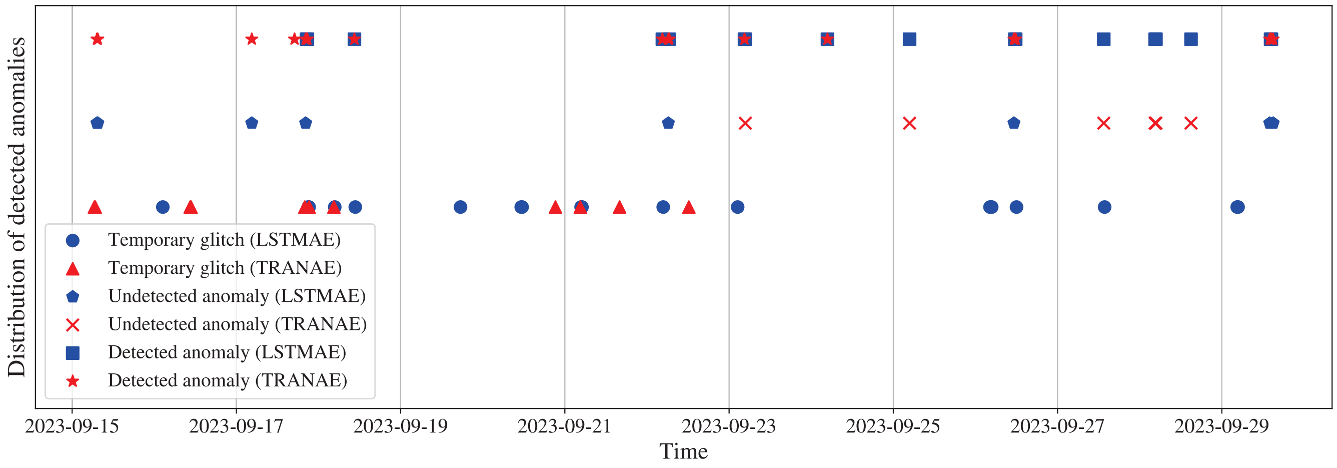


Figure 4: Overview of results based on the feedback from the operations team

slightly higher precision by 76.9 − 65.4 = 11.5%. Given that none of the anomalies are missed, the F1 score equals 0.64, representing the balanced measure of the performance of *smart alerts*. This result is still sufficiently good considering the customized definition of the anomaly construct, elaborated in Section 6.2.

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

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

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

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

Usage. The majority of respondents pointed out that they would use *smart alerts* for initial investigation before temporary anomalies roll out to severe issues that might affect customers. Additionally, one of the respondents brought up an interesting remark that the new alert channel would be helpful in the QA environment for preventing failures during releases. However, they also shared their concern regarding the GTP-3 predictions, as they could overwhelm the operations team with the same or similar predictions. Even though they are related to recurring errors that are not immediately resolved, they could lower the overall usage of *smart alerts*.

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

Implementing and evaluating machine learning models in an industrial setting remains a challenging task. The difficulty increases even more when ground truth data is unavailable, as in our case. A priori assumptions and tuned parameters might not hold after deployment in operations and thus will require adaptations and fine-tuning. As suggested in our recent work [5], several iterations might be needed to reach an optimal version of the solution that meets the demands of the industrial context.

6 DISCUSSION

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

6.1 Findings

As mentioned in Section 3, the goal of this study was to extend the validity of the technological rule implemented and evaluated in the context of the microservice system for ticket and payment management in public transportation (TPM system) [5]. The new context introduced a different cloud-based system in terms of functionality but similar in regards to the cloud architecture and environment. Having the same cloud provider of services for deployment and monitoring, Microsoft Azure, enabled an easy transition between the contexts. However, the monitoring data characterizing the health of both systems partially differed, as some parts of the systems were implemented and deployed using different cloud services. Additionally, the distinct functionalities of the systems contributed to prioritizing different data types, such as database-related metrics for the PIM system, since managing customer data was one of the main functionalities. Regarding the source of the monitoring data, the TPM system mainly relied on performance metrics accessible in Applications Insights, while obtaining critical observations of the PIM system required executing customized queries in Applications Insigts and Log Analytics. These custom queries targeted specific parts of the system and environments and were defined by operations teams. This was a slightly undesired circumstance as fetching the data in near-real-time needed to be implemented with four Azure Functions. On the contrary, for the TPM system, the specific performance counters were accessed by running simple HTTP requests against Azure Metrics APIs.

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

Besides case-based generalization, which included implementing and evaluating the autonomous monitors in another case context, we aimed to improve the proposed solution design in terms of functionality. We were mainly inspired by the observation from the previous case study [5], where we identified that operations teams were not sufficiently motivated to examine reported alerts, as they could not conclude the severity level based on the provided information. They required additional information upon which they could take action. Accordingly, we decided to include logs in the analysis and propose a novel approach for reporting interpretable alerts (contribution **C1**). Based on the qualitative feedback, the *smart alerts* containing actionable guidelines for root cause analysis seemed very appealing to the operations engineers. However, minor adjustments might be needed regarding the visual aspect to enable smooth analysis and investigation of reported alert notifications.

Even though our initial solution [5] included the widely adopted method for detecting anomalies in multivariate time series (*LSTMAE* model), we decided to enrich the original design of the autonomous monitors considering the latest advancements in the ML field. Thus, the solution by Tuli et al. [27] seemed like a perfect fit as it showed very promising results on the publicly available datasets and only needed to be instantiated and evaluated in an industrial setting. We refer to this validity extension as contribution **C2**. Industrial contexts without ground truth data are always challenging, as testing selected ML models before deployment is not feasible. Furthermore, the distribution of data used for training might affect the learning process and bring even more uncertainty to the evaluation phase. In our case, the *TRANAE* model showed solid performance and higher precision rate by 11.5% compared to the *LSTMAE* model. It can be used to detect both short- and long-term anomalies, as shown in Figure 3. Due to its high potential, it will be used for similar solutions in our future work.

6.2 Threats to validity

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

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

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

Internal validity. Selection and preprocessing of the features can significantly impact the performance of ML models and overall results. The 127 features described in Section 4 were carefully selected based on the very thorough observations of monitoring data visualized in Grafana and inputs we received from the operations team. We mainly focused on data types (features) that were usually monitored and examined when some severe failures were detected. However, considering the system dimensionality and the volume of available monitoring data, the selected set of features could be adjusted in the future by adding new or removing some of the existing features. This will, however, require training new ML models, but revising the existing and exploring new features is necessary for building trustworthy and reliable machine learning solutions.

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

7 CONCLUSION

Using ML-inspired solutions for improving software engineering environments and different quality aspects of cloud-based systems has become a standard nowadays. The possibilities are endless and yet to be explored. In this paper, we report one of the applied solutions that leverage state-of-the-art ML models and cloud-based services for the timely detection of anomalies and reporting of interpretable alerts. To remain at the forefront of recent research, we included ML models (*TRANAE* [27] and *GPT-3* [17]) based on deep transformer networks in the design of the solution, the autonomous monitors. Thus, we extended the validity scope for the *TRANAE* model as it has not been evaluated previously in the industrial context of the cloud-based system. Additionally, the validity was extended for the overall solution for improving alert management, expressed in the form of the technological rule in our recent work [5]. The solution design for reporting *smart alerts* proved to be efficient in proactive monitoring and early investigation of raised issues before they escalate and cause severe consequences.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262 (2017), 134–147. https://doi.org/10.1016/j.neucom.2017.04.070

[2] Juncal Alonso, Leire Orue-Echevarria, and Maider Huarte. 2022. CloudOps: Towards the Operationalization of the Cloud Continuum: Concepts, Challenges and a Reference Framework. *Applied Sciences* 12, 9 (2022). https://doi.org/10.3390/app12094347

[3] Nathan Bosch and Jan Bosch. 2020. Software Logs for Machine Learning in a DevOps Environment. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 29–33. https://doi.org/10.1109/SEAA51224.2020.00016

[4] Domenico Cotroneo, Luigi De Simone, Pietro Liguori, and Roberto Natella. 2021. Enhancing the Analysis of Software Failures in Cloud Computing Systems with Deep Learning. *Journal of Systems and Software* 181 (2021), 111043. https://doi.org/10.1016/j.jss.2021.111043

[5] Adha Hrusto, Emelie Engström, and Per Runeson. 2023. Towards optimization of anomaly detection in DevOps. *Information and Software Technology* 160 (2023), 107241. https://doi.org/10.1016/j.infsof.2023.107241

[6] Mohammad S. Islam, William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. 2021. Anomaly Detection in a Large-Scale Cloud Platform. In *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice* (Virtual Event, Spain) *(ICSE-SEIP '21)*. IEEE Press, 150–159. https://doi.org/10.1109/ICSE-SEIP52600.2021.00024

[7] Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski. 2020. Detecting Performance Anomalies in Cloud Platform Applications. *IEEE Transactions on Cloud Computing* 8, 3 (July 2020), 764–777. https://doi.org/10.1109/TCC.2018.2808289

[8] Javier Jose Diaz Rivera, Talha Ahmed Khan, Waleed Akbar, Muhammad Afaq, and Wang-Cheol Song. 2021. An ML Based Anomaly Detection System in Real-Time Data Streams. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, Las Vegas, NV, USA, 1329–1334. https://doi.org/10.1109/CSCI54926.2021.00270

[9] George K. Karagiannidis and Athanasios S. Lioumpas. 2007. An Improved Approximation for the Gaussian Q-Function. *IEEE Communications Letters* 11, 8 (2007), 644–646. https://doi.org/10.1109/LCOMM.2007.070470

[10] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Yongqiang Yang, and Michael R. Lyu. 2023. Heterogeneous Anomaly Detection for Software Systems via Semi-supervised Cross-modal Attention. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1724–1736. https://doi.org/10.1109/ICSE48619.2023.00148

[11] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Yongqiang Yang, and Michael R. Lyu. 2023. Heterogeneous Anomaly Detection for Software Systems via Semi-supervised Cross-modal Attention. arXiv:2302.06914 [cs]

[12] Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. 2019. MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks. In *Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series*, Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis (Eds.). Springer, Cham, 703–716.

[13] Jinyang Liu, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Cong Feng, Zengyin Yang, and Michael R. Lyu. 2023. Practical Anomaly Detection over Multivariate Monitoring Metrics for Online Services. arXiv:2308.09937 [cs]

[14] Sepehr Maleki, Sasan Maleki, and Nicholas R. Jennings. 2021. Unsupervised anomaly detection with LSTM autoencoders using statistical data-filtering. *Applied Soft Computing* 108 (2021), 107443. https://doi.org/10.1016/j.asoc.2021.107443

[15] Leonardo Mariani, Mauro Pezzè, Oliviero Riganelli, and Rui Xin. 2020. Predicting Failures in Multi-Tier Distributed Systems. *Journal of Systems and Software* 161 (March 2020), 110464. https://doi.org/10.1016/j.jss.2019.110464

[16] Mohsin Munir, Shoaib Ahmed Siddiqui, Andreas Dengel, and Sheraz Ahmed. 2019. DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. *IEEE Access* 7 (2019), 1991–2005. https://doi.org/10.1109/ACCESS.2018.2886457

[17] OpenAI. 2020. *GPT-3: Language Models are Few-Shot Learners*. https://openai.com/research/language-models-are-few-shot-learners

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[19] Rui Ren, Yang Wang, Fengrui Liu, Zhenyu Li, and Gaogang Xie. 2023. Triple:The Interpretable Deep Learning Anomaly Detection Framework based on Trace-Metric-Log of Microservice. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS57198.2023.10188773

[20] Per Runeson, Emelie Engström, and Margaret-Anne Storey. 2020. The Design Science Paradigm as a Frame for Empirical Software Engineering. In *Contemporary Empirical Methods in Software Engineering*, Michael Felderer and Guilherme Horta Travassos (Eds.). Springer, 127–147. https://doi.org/10.1007/978-3-030-32489-6_5

[21] Carla Sauvanaud, Mohamed Kaâniche, Karama Kanoun, Kahina Lazri, and Guthemberg Da Silva Silvestre. 2018. Anomaly Detection and Diagnosis for Cloud Services: Practical Experiments and Lessons Learned. *Journal of Systems and Software* 139 (2018), 84–106. https://doi.org/10.1016/j.jss.2018.01.039

[22] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. 2017. Anomaly Detection in Streams with Extreme Value Theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Halifax NS Canada, 1067–1075. https://doi.org/10.1145/3097983.3098144

[23] Jacopo Soldani and Antonio Brogi. 2023. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *Comput. Surveys* 55, 3 (2023), 1–39. https://doi.org/10.1145/3501297

[24] Daniel Sun, Min Fu, Liming Zhu, Guoqiang Li, and Qinghua Lu. 2016. Non-Intrusive Anomaly Detection With Streaming Performance Metrics and Logs for DevOps in Public Clouds: A Case Study in AWS. *IEEE Transactions on Emerging Topics in Computing* 4, 2 (2016), 278–289. https://doi.org/10.1109/TETC.2016.2520883

[25] Damian A. Tamburri, Marco Miglierina, and Elisabetta Di Nitto. 2020. Cloud Applications Monitoring: An Industrial Study. *Information and Software Technology* 127 (2020), 106376. https://doi.org/10.1016/j.infsof.2020.106376

[26] Laszlo Toka, Gergely Dobreff, David Haja, and Mark Szalay. 2021. Predicting Cloud-Native Application Failures Based on Monitoring Data of Cloud Infrastructure. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 842–847.

[27] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. 2022. TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data. arXiv:2201.07284 [cs]

[28] Yiannis Verginadis. 2023. A Review of Monitoring Probes for Cloud Computing Continuum. In *Advanced Information Networking and Applications*, Leonard Barolli (Ed.). Springer, Cham, 631–643.

A QUESTIONNAIRE FORM

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

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

B ONLINE RESOURCES

All Python scripts and Jupyter notebooks used in the study are available upon request at https://github.com/adha7/inAlerts-icse due to privacy concerns of the case company. Implementation of the *TRANAE* model by Tuli et al. [27] can be found at the https://github.com/imperial-qore/TranAD.

Received 6 October 2023