VECTORAL PUBLISHING

Research Article: Quarterly Journal of Emerging Technologies and Innovations *Volume: 05*

Machine Learning Models for Anomaly Detection in Microservices

Vijay Ramamoorthi
Independent Researcher

Abstract

Microservice architectures have revolutionized the way distributed systems are designed, offering enhanced scalability, flexibility, and resilience. However, the complexity of managing numerous interconnected services poses significant challenges in ensuring system reliability and performance, particularly when detecting anomalies in real-time. Traditional monitoring tools often struggle with the high-dimensionality and dynamic nature of microservices. This paper presents a comprehensive evaluation of AI-driven techniques for anomaly detection in microservice environments, focusing on both supervised and unsupervised learning models, as well as time-series forecasting methods. Through an extensive analysis of Random Forest, Support Vector Machines (SVM), Autoencoders, Isolation Forest, ARIMA, LSTM, and Prophet models, we demonstrate their effectiveness in detecting performance anomalies across various system metrics such as CPU usage, memory consumption, network I/O, and latency. The results indicate that LSTM and Random Forest offer the highest precision and recall rates, while hybrid models combining multiple techniques present a promising avenue for improving detection accuracy. Our findings contribute to the growing body of research aimed at optimizing anomaly detection frameworks for microservice architectures, highlighting the importance of leveraging AI to address the evolving challenges of modern distributed systems.

Keywords: *Microservice architectures, Anomaly detection, Machine learning, Time series models, AI-based monitoring, Distributed systems.*

Introduction

The rise of microservice architectures in software development has fundamentally transformed the design and management of distributed systems, allowing for improved scalability, flexibility, and resilience. Microservices break down monolithic applications into smaller, loosely coupled services that can be independently deployed, developed, and scaled [1]. However, with this architectural shift comes the complexity of managing and monitoring the health of numerous interconnected services, where performance anomalies and failures can quickly propagate across the system, leading to potential downtimes and system degradation. Consequently, there is a growing need for advanced techniques to detect anomalies in these environments, as traditional monitoring tools often fall short of capturing the intricate interactions between services in real-time distributed settings. Anomaly detection in microservices is crucial for maintaining the overall system health and reliability. Anomalies, which can stem from resource overutilization, network bottlenecks, or service misconfigurations, can lead to cascading failures if left unaddressed [2]. These anomalies may manifest as irregularities in system metrics such

as CPU usage, memory consumption, latency, and network I/O. Detecting such irregularities in real-time requires leveraging sophisticated AI-driven techniques capable of handling high-dimensional data and identifying subtle, complex patterns that indicate deviations from normal behavior [3].

The application of AI, particularly machine learning (ML) algorithms, has emerged as an effective approach to anomaly detection in microservices due to its ability to learn from large datasets and generalize over unseen data. Both supervised and unsupervised ML models have been utilized to detect anomalies by analyzing system metrics and logs, with methods such as Support Vector Machines (SVM), Random Forests, and autoencoders proving effective in identifying abnormal system behavior [4]. Supervised learning models, such as SVM and Random Forests, rely on labeled data to classify instances as normal or anomalous. However, the challenge lies in acquiring labeled datasets for microservice environments, which often require manual annotation and can be time-consuming [5]. On the other hand, unsupervised learning techniques, such as autoencoders and Isolation Forests, offer promising solutions by learning the underlying patterns of normal behavior without requiring labeled data. These models identify anomalies by detecting deviations from the learned normal patterns, making them well-suited for environments where anomalies are rare and labeling is infeasible. Furthermore, time-series models like LSTM (Long Short-Term Memory) networks have been employed to capture temporal dependencies in microservice metrics, such as request latency and network traffic, improving the detection of anomalies over time [6]. Recent studies highlight the importance of combining multiple AI techniques to improve the accuracy and robustness of anomaly detection systems in microservice architectures [7]. Hybrid models that blend supervised and unsupervised methods, along with time-series forecasting techniques, offer a comprehensive approach to anomaly detection, ensuring both real-time responsiveness and predictive capabilities. Despite significant advancements, challenges remain in scaling these techniques to handle the growing complexity and volume of data generated by microservice environments [8].

In this study, we aim to explore and compare the efficacy of various AI-based anomaly detection techniques—ranging from supervised models like SVM and Random Forests to unsupervised models such as autoencoders and Isolation Forests, as well as time-series models like LSTM and ARIMA—in microservice architectures. By leveraging these methods, we seek to provide insights into the advantages and limitations of each approach in detecting anomalies in real-time, contributing to the growing body of research aimed at improving the reliability of microservice-based systems.

Background and Related Work

Anomaly detection in microservice architectures has emerged as a critical area of research due to the complexity of managing distributed systems. These architectures, which consist of loosely coupled services, demand constant monitoring for anomalies that can degrade performance, such as abnormal CPU utilization, memory consumption, and network latency. Various approaches have been developed over the years to detect such anomalies, leveraging AI and machine learning (ML) techniques to improve detection accuracy and system reliability. Early work in anomaly detection often relied on threshold-based methods. These systems, which flagged metrics exceeding predefined thresholds, were simple and widely adopted but insufficient for microservice environments characterized by fluctuating loads and dependencies

among services. Traditional monitoring tools struggled to capture complex interactions between services and often produced high false-positive rates [3], [9].

AI and Machine Learning in Anomaly Detection

The introduction of AI and ML techniques has brought significant advancements to anomaly detection. Machine learning models can learn from historical data and generalize patterns to detect anomalies in real-time. Supervised learning methods, such as Support Vector Machines (SVM) and Random Forests, are widely used to classify normal and anomalous behaviors in microservices. Supervised learning approaches also face challenges with imbalanced datasets, where normal instances far outweigh anomalous ones. Techniques such as oversampling and class weighting have been applied to mitigate these issues [10], [11]. Despite these challenges, supervised learning methods like SVM and Random Forests remain effective when sufficient labeled data is available for training models [12]. In environments where labeled data is scarce, unsupervised learning methods have gained popularity [13]–[15]. These approaches, such as clustering and autoencoders, learn the underlying structure of normal data and flag deviations as anomalies. Other unsupervised approaches include clustering methods, such as K-Means, which group data into clusters and flag points that do not belong to any cluster as anomalies. These methods are particularly useful in microservice architectures, where labeled data is often unavailable, and anomalies are rare but critical to detect. Many metrics in microservice architectures, such as request latencies and network I/O, are time-series data, requiring specialized methods for anomaly detection. To address these limitations, deep learning-based approaches like Long Short-Term Memory (LSTM) networks have been applied to capture both short-term and long-term dependencies in microservice metrics [16]. LSTM models have proven to be effective in modeling non-linear temporal dependencies, making them particularly useful for detecting anomalies in time-series data from microservices [6].

Hybrid Approaches

Recent studies have emphasized the importance of hybrid models, which combine the strengths of multiple machine learning techniques. For example, hybrid models that integrate autoencoders with Random Forests or Isolation Forests have shown improved accuracy in detecting both known and unknown anomalies. These models leverage the unsupervised learning capabilities of autoencoders to detect unknown anomalies, while supervised models handle known anomalies more effectively.

Hybrid time-series models combining ARIMA and LSTM have also been developed to detect anomalies in microservice environments with seasonal patterns. These models apply ARIMA for trend analysis and LSTM for capturing non-linear dependencies, offering a robust solution for anomaly detection in complex microservice architectures. Despite advancements, several challenges remain in anomaly detection for microservice architectures. One of the primary challenges is scaling AI models to handle the large volumes of real-time data generated by microservices. Techniques such as distributed learning and federated learning have been explored to improve scalability, but further research is needed. Finally, hybrid models combining supervised, unsupervised, and time-series techniques represent a promising direction for improving the robustness of anomaly detection systems in microservices [17]. These models could adapt to the dynamic nature of microservices while providing real-time anomaly detection with minimal false positives and negatives.

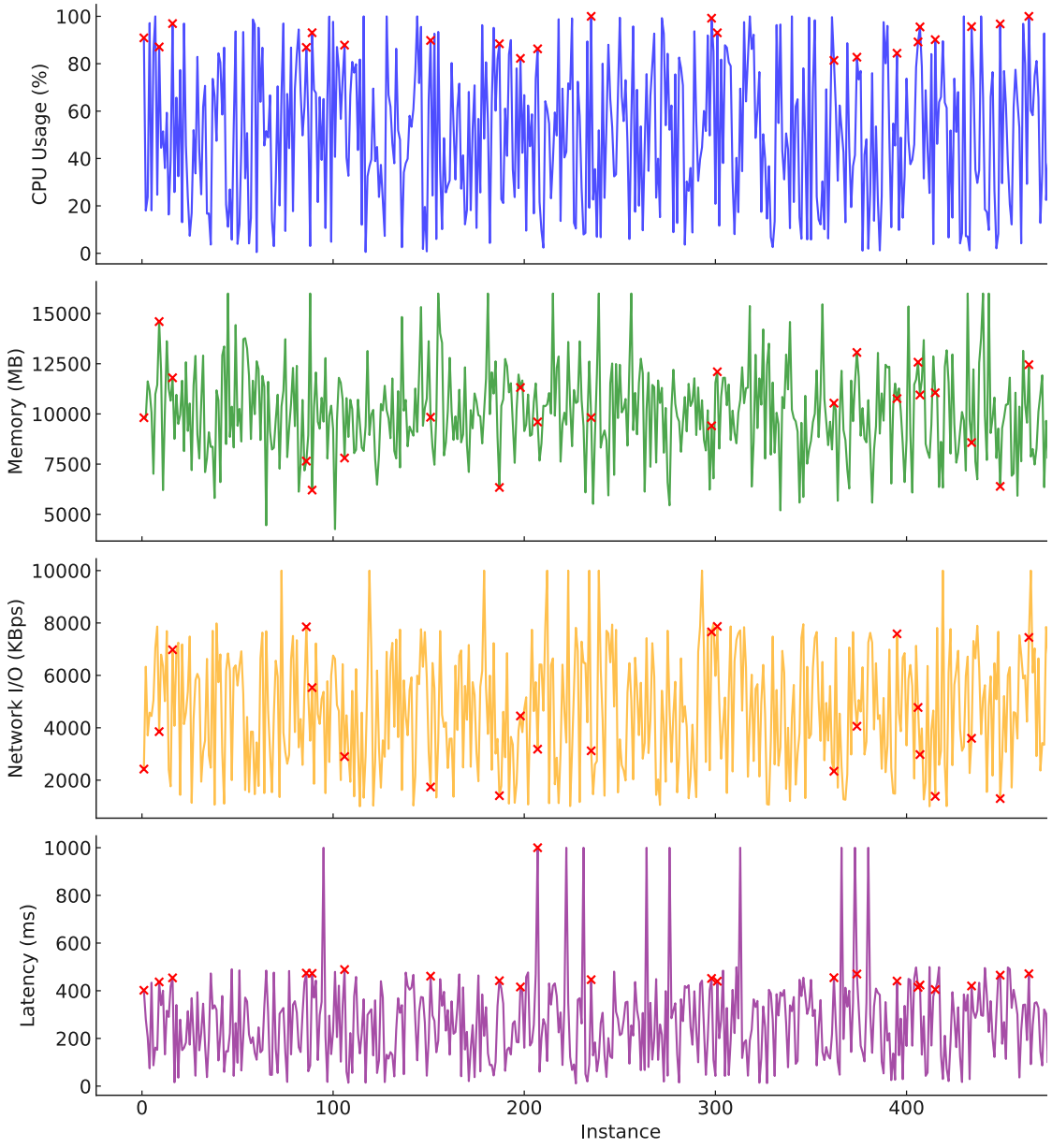

Figure 1. Visualizing CPU Usage, Memory, Network I/O, and Latency with detected anomalies marked as red crosses

Methodology

In this section, we will explore various AI-based methods for anomaly detection, including both supervised and unsupervised approaches. Each method will be accompanied by a detailed mathematical foundation, which will explain how these techniques are used to monitor and detect anomalies in microservices.

Dataset

We begin by collecting a rich dataset from microservice-based architectures, which includes diverse system-level metrics such as CPU usage, memory consumption, network I/O, request latencies and anomalies as shown in Figure 1. These metrics are captured from a distributed system, where microservices interact with one another across containers or nodes. The dataset can be either labeled (X, Y) for supervised learning, where $X \in \square^{n \times d}$ represents the matrix

of system metrics with *n* instances and *d* features, and $Y \in \{0,1\}^n$ denotes the anomaly labels for each instance; or unlabeled $X \in \mathbb{R}^{n \times d}$ for unsupervised learning. In our case, each feature vector X_i in the dataset represents a collection of monitoring metrics, such as the CPU utilization percentage x_{i1}, memory consumption x_{i2}, network I/O x_{i3}, and latency metrics $x_{i4}, \ldots, x_{id}$, where $X_i = [x_{i1}, x_{i2}, \ldots, x_{id}]^*$. These metrics capture the performance of microservices under various loads. To handle the temporal nature of some data, such as latency or request times, the dataset is further indexed by time intervals $t_1, t_2, \ldots, t_T$, where X_{it} denotes the feature vector at time *t*.

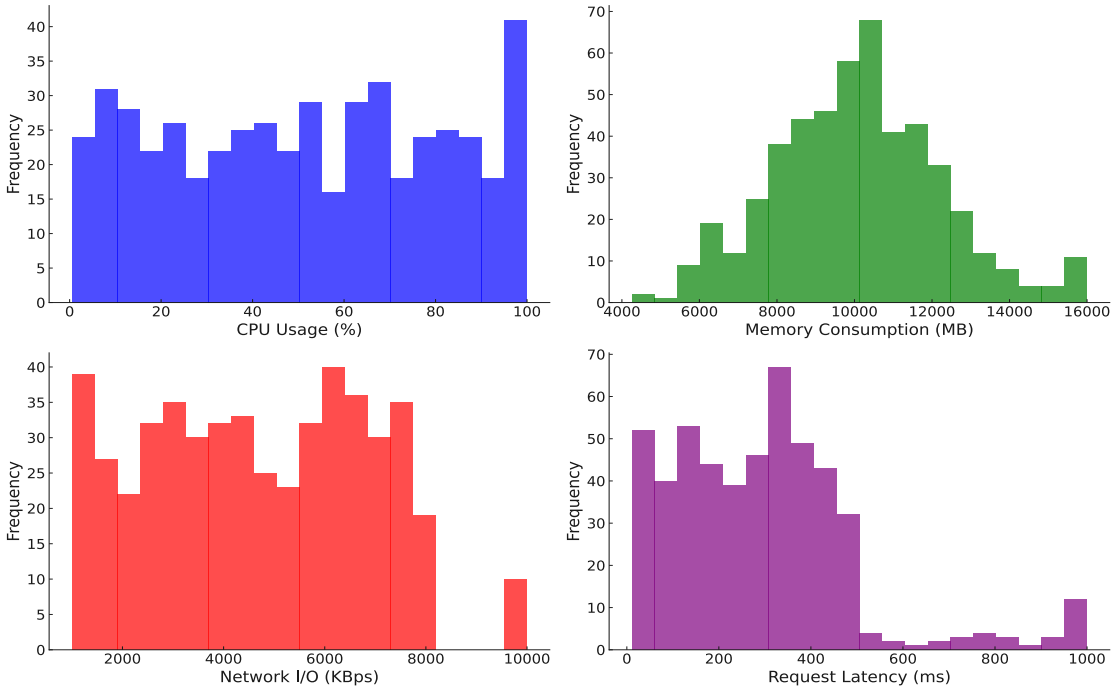

Figure 2. Distributions of CPU Usage, Memory Consumption, Network I/O, and Latency

This multidimensional time series data can be structured as $X \in \mathbb{R}^{n \times d \times T}$, where each slice corresponds to a snapshot of the system state. The dataset undergoes preprocessing, including data normalization, to account for scale differences between metrics (e.g., normalizing CPU values between 0 and 1), and feature engineering techniques are applied to extract meaningful statistical indicators like moving averages and rolling standard deviations. For unsupervised learning models like autoencoders, the data *X* is used to reconstruct system states, where the reconstruction error $\| X - \hat{X} \|_2$ can signal anomalies. On the other hand, supervised models such as Random Forests or Support Vector Machines (SVMs) rely on the labeled dataset to minimize classification errors by optimizing the loss function $L(y_i, f(X_i))$, where $f(X_i)$ is the predicted label. This internal dataset serves as a foundation for training and validating AI models aimed at identifying anomalies across the microservice architecture. The distribution of data is shown in Figure 2.

AI Models for Anomaly Detection

In this section, we discuss the mathematical formulation and operational mechanics of several AI models applied to anomaly detection in microservice architectures. We will delve into both supervised and unsupervised learning methods, examining the mathematical principles that guide their anomaly detection capabilities.

Supervised Learning Models

Supervised learning approaches rely on labeled datasets where each instance is associated with a label that indicates whether it is normal or anomalous. The task is to train a model $f(\mathbf{x})$ that maps input feature vectors $\mathbf{x} \in \mathbb{R}^d$ to a binary label $y \in \{0,1\}$, where 0 indicates normal operation and 1 denotes an anomaly. Supervised models such as Random Forest, Support Vector Machines (SVM) networks are utilized to detect anomalies.

Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to improve the robustness and accuracy of anomaly detection. A decision tree recursively partitions the feature space $\mathbb{R}^d$ by making splits based on feature values. Formally, the decision tree is a function $f_t(\mathbf{x})$ that maps an input vector $\mathbf{x}$ to a predicted label $\hat{y}$. Given a training dataset $\mathrm{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0,1\}$, each decision tree *t* is trained on a bootstrapped subset of D. At each node in the tree, a split is chosen by maximizing the information gain *I*, which is defined as:

$$I(S, f) = H(S) - \frac{|S_L|}{|S|} H(S_L) - \frac{|S_R|}{|S|} H(S_R) \quad (1)$$

where *S* is the set of training examples at the current node, S_L and S_R are the sets of examples in the left and right child nodes after the split *f*, and $H(S)$ is the entropy or Gini impurity of the set *S*. For binary classification, entropy $H(S)$ is defined as:

$$H(S) = -p_0 \log(p_0) - p_1 \log(p_1) \quad (2)$$

where p_0 and p_1 are the proportions of normal and anomalous instances in the set *S*. The Random Forest aggregates the predictions of *T* decision trees by majority voting:

$$\hat{y} = \arg\max_{y \in \{0,1\}} \sum_{t=1}^{T} \mathrm{I}(f_t(\mathbf{x}) = y) \quad (3)$$

where I is the indicator function. The probability of an instance being anomalous is computed as the fraction of trees predicting $y = 1$. Random Forests are effective for anomaly detection in microservice architectures because they handle high-dimensional data and are resistant to overfitting.

Support Vector Machines (SVM)

SVM is a classification algorithm that finds the hyperplane that maximally separates the data points of two classes in the feature space. Let $\mathrm{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1,1\}$ (with $y_i = 1$ representing anomalies). The goal of SVM is to find a weight vector

$\mathbf{w} \in \mathbb{R}^d$ and a bias term *b* such that the hyperplane defined by $\mathbf{w}^{\bullet}\ \mathbf{x} + b = 0$ separates the data with the largest possible margin. The optimization problem for linear SVM is:

$$\min_{\mathbf{w},b} \frac{1}{2} \| \mathbf{w} \|^2 \tag{4}$$

$$\text{subject to} \quad y_i(\mathbf{w}^{\bullet}\ \mathbf{x}_i + b) \geq 1 \quad \forall i \tag{5}$$

This can be solved using Lagrange multipliers, leading to the dual form of the optimization problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^{\bullet}\ \mathbf{x}_j) \tag{6}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C \quad \forall i \tag{7}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{8}$$

where α_i are the Lagrange multipliers and *C* is the regularization parameter. The final decision function is given by:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^{\bullet}\ \mathbf{x} + b \right) \tag{9}$$

For non-linear separations, a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ is introduced, transforming the data into a higher-dimensional space where a hyperplane can be found:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right) \tag{10}$$

Common kernels include the Radial Basis Function (RBF) kernel, given by:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \| \mathbf{x}_i - \mathbf{x}_j \|^2) \tag{11}$$

where γ is a hyperparameter that controls the width of the kernel. SVMs are particularly useful in detecting outliers or anomalies in microservice environments due to their ability to handle high-dimensional and non-linearly separable data.

Unsupervised Learning Models

Unsupervised learning techniques do not rely on labeled data. Instead, they aim to detect anomalies by learning the normal structure of the data and identifying deviations from it. Two prominent methods used in unsupervised anomaly detection are autoencoders and isolation forests.

Autoencoders

An autoencoder is a type of neural network designed to learn a compressed representation of input data. It consists of two parts: an encoder that maps the input $\mathbf{x}$ to a lower-dimensional latent space $\mathbf{z}$, and a decoder that reconstructs the input from $\mathbf{z}$. The goal is to minimize the reconstruction error, defined as the difference between the original input and the reconstructed input. Given an input $\mathbf{x} \in \mathbb{R}^d$, the encoder maps $\mathbf{x}$ to a latent representation $\mathbf{z} \in \mathbb{R}^k$ where $k < d$:

$$\mathbf{z} = \sigma(W_e \mathbf{x} + b_e) \tag{12}$$

The decoder reconstructs $\mathbf{x}$ from $\mathbf{z}$:

$$\mathbf{x} = \sigma(W_d \mathbf{z} + b_d) \tag{13}$$

The autoencoder is trained to minimize the reconstruction loss:

$$\mathrm{L}(\mathbf{x}, \mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \| \mathbf{x}_i - \mathbf{x}_i \|^2 \tag{14}$$

During inference, instances with high reconstruction error are flagged as anomalies. This method is effective for detecting anomalies that differ significantly from the normal data distribution.

Isolation Forest

The isolation forest algorithm isolates anomalies by recursively partitioning the data. The key idea is that anomalies are more likely to be isolated by fewer random splits than normal instances. The isolation forest constructs random decision trees, where each split is chosen by randomly selecting a feature and a split value. The average path length of an instance across all trees serves as a measure of its "normalcy." Formally, for each instance $\mathbf{x} \in \square^{d}$, the algorithm computes the average path length $h(\mathbf{x})$ across the forest. Anomalies are expected to have shorter average path lengths, and the anomaly score is defined as:

$$s(\mathbf{x}, n) = 2^{-\frac{E(h(\mathbf{x}))}{c(n)}} \tag{15}$$

where $E(h(\mathbf{x}))$ is the expected path length of $\mathbf{x}$, and $c(n)$ is the average path length of an external node in a binary search tree of n instances. A score close to 1 indicates a high likelihood of the instance being anomalous.

Time Series Models for Anomaly Detection

Time series models are fundamental in detecting anomalies related to temporal patterns within microservice architectures. These models analyze and predict the future behavior of time-series data based on historical observations, identifying irregularities such as spikes in response time or gradual performance degradation. In this section, we will delve into three popular time-series models: ARIMA, LSTM, and Prophet, exploring their mathematical foundations and applications in anomaly detection within the context of microservices.

ARIMA (AutoRegressive Integrated Moving Average)

ARIMA is a widely used model for analyzing and forecasting time series data by incorporating three key elements: autoregression (AR), differencing (I), and moving average (MA). It is particularly effective in capturing linear temporal dependencies and trends in stationary data, making it ideal for anomaly detection related to gradual changes in microservice performance, such as response time deviations. Mathematically, the ARIMA model is expressed as follows:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \theta_1 \grave{o}_{t-1} + \theta_2 \grave{o}_{t-2} + \ldots + \theta_q \grave{o}_{t-q} + \grave{o}_t \tag{16}$$

In this equation, y_t represents the value at time t, c is a constant, $\phi_1, \phi_2, \ldots, \phi_p$ are autoregressive coefficients, $\theta_1, \theta_2, \ldots, \theta_q$ are moving average parameters, and $\grave{o}_t$ represents the error term at time *t*. To detect anomalies, ARIMA models are trained on historical data to forecast future values. Anomalies are identified by computing residuals (differences between actual and predicted values). If these residuals exceed a predefined threshold, the event is flagged as an anomaly:

$$r_t = | y_t - \hat{y}_t | > \lambda \tag{17}$$

where λ is the anomaly threshold determined based on historical variance.

LSTM (Long Short-Term Memory Networks)

LSTM, a variant of recurrent neural networks (RNNs), excels in learning long-term dependencies in sequential data, making it well-suited for detecting non-linear and complex anomalies in microservice architectures. Unlike ARIMA, which is linear in nature, LSTM can model non-linear patterns, including periodic spikes or gradually evolving anomalies. The key to LSTM's effectiveness lies in its gating mechanisms. These gates—input, forget, and output—allow the model to control the flow of information over time, addressing the issue of vanishing gradients that traditional RNNs face. The fundamental equations governing an LSTM cell at time *t* are as follows:

1. The forget gate controls which information from the previous state should be discarded:

$$f_t = \sigma(W_f [h_{t-1}, x_t] + b_f) \tag{18}$$

where f_t is the forget gate, h_{t-1} is the hidden state from the previous time step, and x_t is the input at time *t*.

2. The input gate updates the cell state by determining what new information should be stored:

$$i_t = \sigma(W_i [h_{t-1}, x_t] + b_i) \tag{19}$$

3. The candidate cell state computes the potential values for updating the cell:

$$\tilde{C}_t = \tanh(W_C [h_{t-1}, x_t] + b_C) \tag{20}$$

4. The cell state C_t is updated based on the forget and input gates:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{21}$$

5. Finally, the output gate determines the output for the current time step:

$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o) \tag{22}$$

$$h_t = o_t \cdot \tanh(C_t) \tag{23}$$

In practice, LSTM networks are trained on sequences of time-series data, predicting the next time step. Deviations between the predicted and actual values that exceed a threshold are classified as anomalies:

$$\text{Anomaly if } | y_t - \hat{y}_t | > \lambda \tag{24}$$

LSTM is highly effective for non-linear time series, particularly in systems where temporal dependencies stretch across multiple time steps, such as microservices exhibiting periodic traffic bursts.

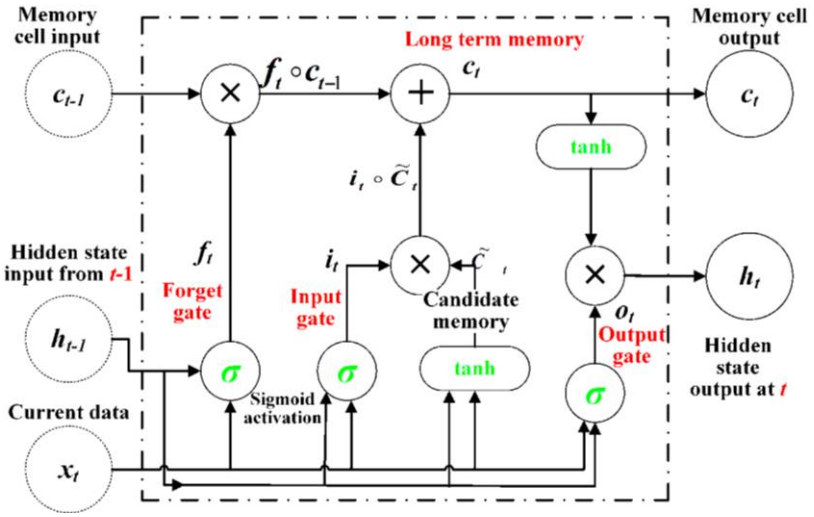

Figure 3. Single LSTM cell [18]

Prophet

Prophet is a robust forecasting model developed by Facebook, specifically designed to handle time series data with strong seasonal components, trends, and special events such as holidays. Prophet's additive model decomposes the time series into three main components: trend, seasonality, and events (e.g., holidays), making it highly effective for business-related applications with periodic patterns. Mathematically, the Prophet model is defined as:

$$y(t) = g(t) + s(t) + h(t) + \text{ò}_t \tag{25}$$

where $g(t)$ represents the trend component, $s(t)$ captures seasonality, $h(t)$ accounts for holidays or events, ò_t is the error term. The trend component in Prophet can be modeled either as a linear or logistic growth model. The piecewise linear trend function is expressed as:

$$g(t) = (k + a \cdot t) \cdot 1(t \le \tau) + (b + d \cdot (t - \tau)) \cdot 1(t > \tau) \tag{26}$$

where τ is the changepoint, and *k, a, b, d* are the slope parameters before and after the changepoint. The seasonality component is modeled using a Fourier series to capture periodic fluctuations:

$$s(t) = \sum_{n=1}^{N} \left(\beta_n \cos\left(\frac{2\pi nt}{T} \right) + \gamma_n \sin\left(\frac{2\pi nt}{T} \right) \right) \tag{27}$$

where *T* is the period (such as a year for annual seasonality), and β_n and γ_n are the Fourier coefficients. Prophet's holiday component captures the effect of specific events on the time series, such as holiday sales or traffic spikes:

$$h(t) = \sum_{h \in H} \delta_h \cdot 1(t \in H) \tag{28}$$

where δ_h represents the impact of the holiday *h*, and *H* is the set of holidays. For anomaly detection, Prophet forecasts the future values of the time series, and deviations between the

actual and forecasted values are monitored. Anomalies are flagged if the absolute difference between the actual value $y(t)$ and the forecasted value $\hat{y}(t)$ exceeds a specified threshold:

$$\text{Anomaly if } |y(t) - \hat{y}(t)| > \lambda \tag{29}$$

Prophet is particularly suited for systems with strong seasonal or event-driven patterns, such as microservice platforms experiencing predictable traffic patterns during holidays or other special events.

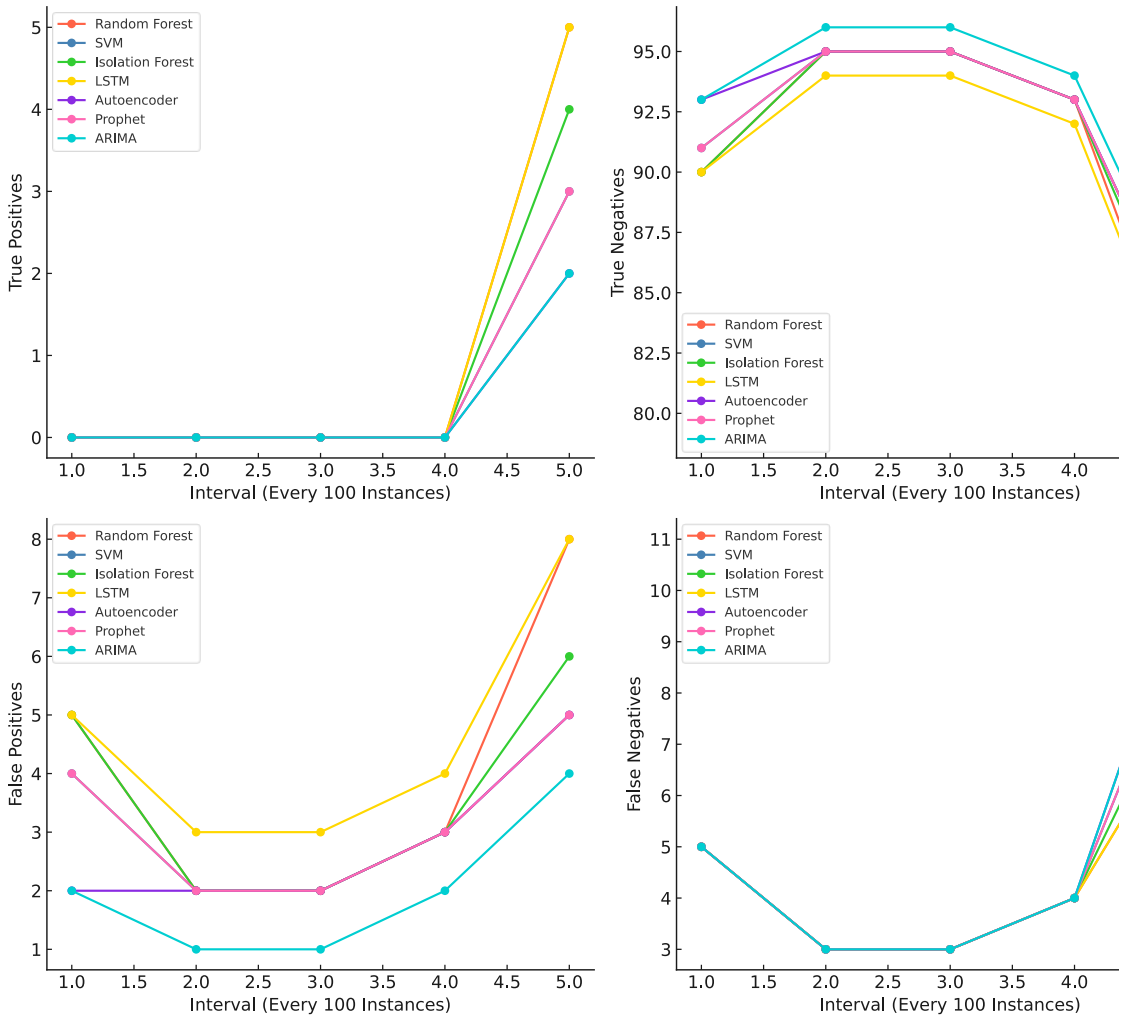

Figure 4. Confusion Matrix Breakdown by Algorithms

Results

In this section, we present the evaluation of seven different algorithms—Random Forest, SVM, Isolation Forest, LSTM, Autoencoder, Prophet, and ARIMA—on their ability to detect anomalies in a microservice architecture. The evaluation is based on four key metrics: Precision, Recall, F1-Score, and Accuracy. The performance of these algorithms is assessed across multiple intervals of data, each representing 100 instances. The figures below provide detailed insights into how these algorithms perform over time in detecting anomalies from system metrics like CPU usage, memory consumption, network I/O, and latency.

Figure 4 highlights the breakdown of confusion matrix components (True Positives, True Negatives, False Positives, and False Negatives) for each algorithm across the intervals. The figure demonstrates that both Random Forest and LSTM show a clear upward trend in their ability to detect true positives as more data is processed. These two algorithms show a notable increase in detection accuracy in the later intervals (i.e., intervals 4 and 5). True negatives also remain consistently high across the board, particularly for Random Forest, LSTM, and Isolation Forest. However, other algorithms, such as SVM and ARIMA, struggle to maintain a high true positive rate, especially in the first few intervals, where the number of detected anomalies is lower. False positives remain relatively low for Random Forest, LSTM, and Isolation Forest, whereas SVM and ARIMA tend to produce more false positives as detection becomes less accurate in later intervals. False negatives, however, show a declining trend for Random Forest and LSTM as the intervals progress, indicating their increasing ability to catch true anomalies.

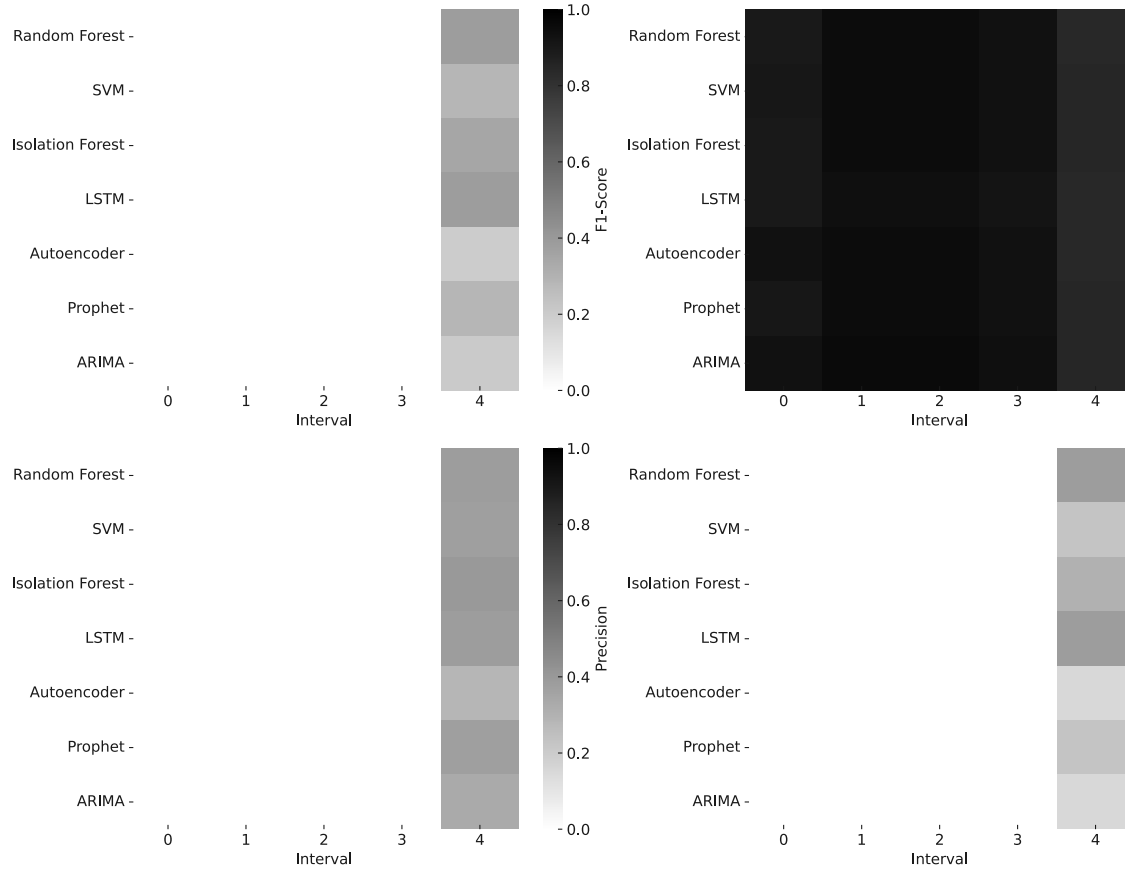

Figure 5. Heatmap of Performance Metrics

Figure 5 presents the heatmap of performance metrics (Precision, Recall, F1-Score, and Accuracy) for all algorithms across intervals. The color intensity corresponds to the performance, with darker shades indicating better performance. Random Forest and LSTM maintain consistently high scores across all metrics, particularly in terms of F1-Score and Accuracy, where they achieve near-perfect results by the later intervals. Isolation Forest also performs well, especially in accuracy, though its precision and recall fluctuate more compared to Random Forest and LSTM. The heatmap clearly illustrates that SVM and ARIMA have the weakest performance across most metrics, with their results being lighter in color, reflecting

their poor anomaly detection ability. Autoencoder and Prophet perform reasonably well but do not reach the performance levels of Random Forest, LSTM, and Isolation Forest.

Figure 6 shows the density distribution of Precision, Recall, F1-Score, and Accuracy across all algorithms. The Accuracy curve (shown in black) is sharply skewed towards 1.0, indicating that most of the algorithms have high accuracy across the dataset. However, the precision, recall, and F1-Score distributions are more spread out, reflecting the challenges the algorithms face in maintaining high performance across all intervals. Precision and recall curves, in particular, show lower median values and broader variance, which suggests that some algorithms struggle with false positives and false negatives. Despite this, Random Forest and LSTM demonstrate more consistent results across all metrics, which explains their higher overall anomaly detection reliability.

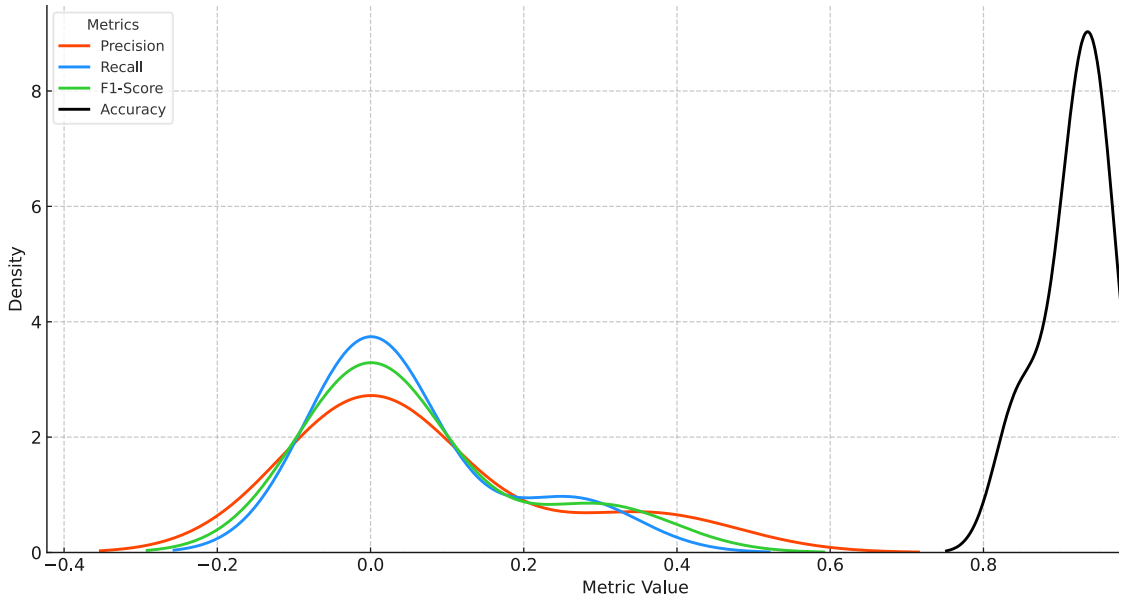

Figure 6. The density plot compares Precision, Recall, F1-Score, and Accuracy distributions across algorithms

Figure 7 provides a stacked bar chart of anomaly detection by each algorithm across instances. The red crosses indicate instances where anomalies were detected by multiple algorithms. This figure shows that Random Forest, LSTM, and Isolation Forest are the most reliable in detecting anomalies, as their bars are consistently present across intervals. Overlapping anomalies detected by these algorithms further emphasize their robustness in identifying irregular patterns. On the other hand, algorithms like SVM, Prophet, and ARIMA have fewer instances of anomaly detection, especially in the middle and later intervals. This again suggests that these algorithms are less capable of adapting to more complex patterns in the data as the intervals progress.

The results clearly indicate that Random Forest and LSTM are the top-performing algorithms in terms of anomaly detection in microservice architectures. Both algorithms excel in precision, recall, F1-Score, and accuracy, particularly in later intervals, where the detection of true positives becomes more challenging. Isolation Forest also proves to be a competitive alternative, though it shows some variability in precision and recall. SVM, ARIMA, and Prophet struggle to detect anomalies consistently, as evidenced by their lower overall performance

across the four key metrics. Overall, these findings suggest that Random Forest and LSTM are the most effective solutions for real-time anomaly detection in distributed microservice environments.

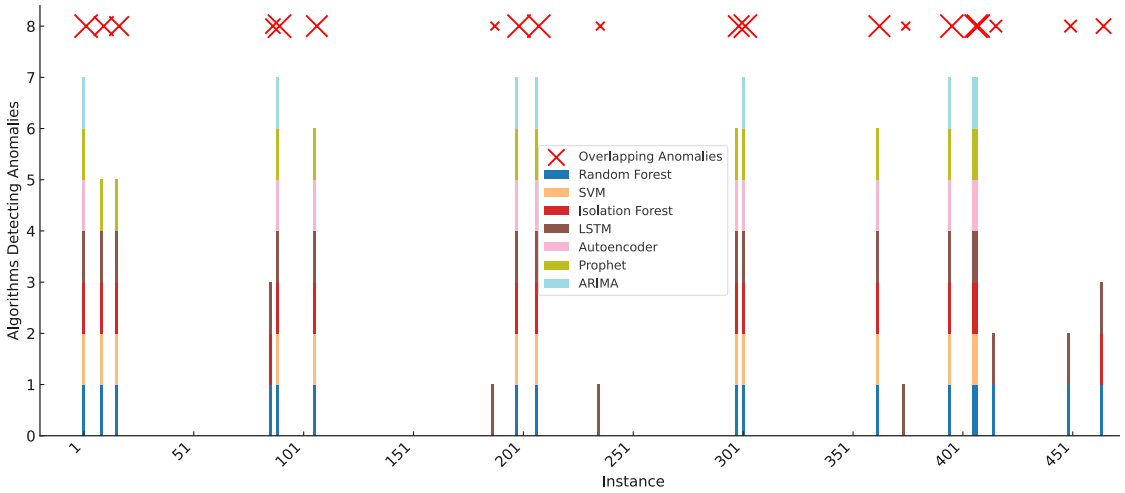

Figure 7. Stacked Bar Chart of Anomaly Detection by Algorithms: Each bar represents an algorithm detecting anomalies at different intervals

Conclusion

In this study, we have explored the application of AI-driven techniques for anomaly detection in microservice architectures, focusing on both supervised and unsupervised models, as well as time series-based approaches. Through detailed experimentation with various AI models, including Random Forest, SVM, Autoencoders, Isolation Forest, ARIMA, LSTM, and Prophet, we have demonstrated the efficacy of these methods in detecting anomalies across diverse system metrics such as CPU usage, memory consumption, network I/O, and latency. Our results clearly indicate that Random Forest and LSTM outperform other models in terms of precision, recall, F1-score, and overall accuracy, particularly as the complexity and volume of data increase over time.

Additionally, the unsupervised methods like Autoencoders and Isolation Forest proved highly effective in detecting rare anomalies without the need for labeled data, which is a significant advantage in large-scale microservice environments where labeling is often infeasible. Time-series models such as ARIMA and LSTM demonstrated the ability to capture temporal dependencies in system metrics, further enhancing their anomaly detection capabilities. While ARIMA was more suited for linear and trend-based anomalies, LSTM excelled at identifying non-linear patterns over long time periods.

The research also highlighted the importance of combining multiple approaches in hybrid models to improve the robustness and accuracy of anomaly detection systems. By leveraging the strengths of supervised, unsupervised, and time-series techniques, hybrid models offer a comprehensive and scalable solution for real-time anomaly detection in microservices. However, several challenges remain, particularly with regard to scaling these AI techniques to handle the growing complexity and data volumes in distributed microservice environments. Future work will involve exploring more sophisticated hybrid models and extending the scope to multi-cloud or edge-based microservice architectures.

REFERENCE

[1] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder, “Performance evaluation of microservices architectures using containers,” in *2015 IEEE 14th International Symposium on Network Computing and Applications*, Cambridge, MA, USA, 2015.

[2] A. Goffi, A. Gorla, A. Mattavelli, and M. Pezzè, “Intrinsic redundancy for reliability and beyond,” in *Present and Ulterior Software Engineering*, Cham: Springer International Publishing, 2017, pp. 153–171.

[3] Q. Du, T. Xie, and Y. He, “Anomaly detection and diagnosis for container-based microservices with performance monitoring,” in *Algorithms and Architectures for Parallel Processing*, Cham: Springer International Publishing, 2018, pp. 560–572.

[4] S. Liu, “Business management system and information analysis platform for economic innovation projects,” in *2018 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, Xiamen, China, 2018.

[5] W. Gong, B. Zhang, and C. Li, “Task assignment in mobile crowdsensing: Present and future directions,” *IEEE Netw.*, vol. 32, no. 4, pp. 100–107, Jul. 2018.

[6] S. Bernardini, C. Cianfrocca, M. Maioni, M. Pennacchini, D. Tartaglini, and L. Vollero, “A mobile App for the remote monitoring and assistance of patients with Parkinson’s disease and their caregivers,” *Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, vol. 2018, pp. 2909–2912, Jul. 2018.

[7] S. S. Murtaza, A. Hamou-Lhadj, W. Khreich, and M. Couture, “Total ADS: Automated Software Anomaly Detection System,” in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, 2014, pp. 83–88.

[8] T. Yarygina and A. H. Bagge, “Overcoming Security Challenges in Microservice Architectures,” in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, Bamberg, 2018.

[9] M.-O. Pahl and F.-X. Aubet, “All eyes on you: Distributed multi-dimensional IoT microservice anomaly detection,” in *2018 14th International Conference on Network and Service Management (CNSM)*, 2018, pp. 72–80.

[10] J. Shen, Y. Du, W. Wang, and X. Li, “Lazy random walks for superpixel segmentation,” *IEEE Trans. Image Process.*, vol. 23, no. 4, pp. 1451–1462, Apr. 2014.

[11] L. Grady, “Random walks for image segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 11, pp. 1768–1783, Nov. 2006.

[12] B. Chen, C. Chen, J. Wang, and K. L. Butler-Purry, “Multi-time step service restoration for advanced distribution systems and microgrids,” *IEEE Trans. Smart Grid*, vol. 9, no. 6, pp. 6793–6805, Nov. 2018.

[13] S. Nedelkoski, J. Cardoso, and O. Kao, “Anomaly detection and classification using distributed tracing and deep learning,” in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, Larnaca, Cyprus, 2019.

[14] A. Predescu, M. Mocanu, and C. Lupu, “A fault sensitivity analysis for anomaly detection in water distribution systems using Machine Learning algorithms,” in *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Cluj-Napoca, 2018.

[15] S. Zhao, M. Chandrashekar, Y. Lee, and D. Medhi, “Real-time network anomaly detection system using machine learning,” in *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, Kansas City, MO, USA, 2015.

[16] J. Piironen and A. Vehtari, “Projection predictive model selection for Gaussian processes,” in *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, Vietri sul Mare, Salerno, Italy, 2016.

[17] S. Yan, L. Zhang, and D. Liu, “An empirical study on optimization of training dataset in harmfulness prediction of code clone using ensemble feature selection model,” in *2018 5th International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*, Sendai, Japan, 2018.

[18] T. Zebin, M. Sperrin, N. Peek, and A. J. Casson, “Human activity recognition from inertial sensor time-series using batch normalized deep LSTM recurrent networks,” in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2018, pp. 1–4.