applied sciences

MDPI

Article

Anomaly Detection in Microservice-Based Systems

João Nobre [1], E. J. Solteiro Pires [2] and Arsénio Reis [2,*]

[1] Universidade Aberta, 1250-100 Lisboa, Portugal; 2001506@estudante.uab.pt
[2] Departamento de Engenharias, Universidade Trás-os-Montes e Alto Douro, 5000-801 Vila Real, Portugal; epires@utad.pt
* Correspondence: ars@utad.pt

Abstract: Currently, distributed software systems have evolved at an unprecedented pace. Modern software-quality requirements are high and require significant staff support and effort. This study investigates the use of a supervised machine learning model, a Multi-Layer Perceptron (MLP), for anomaly detection in microservices. The study covers the creation of a microservices infrastructure, the development of a fault injection module that simulates application-level and service-level anomalies, the creation of a system monitoring dataset, and the creation and validation of the MLP model to detect anomalies. The results indicate that the MLP model effectively detects anomalies in both domains with higher accuracy, precision, recovery, and F1 score on the service-level anomaly dataset. The potential for more effective distributed system monitoring and management automation is highlighted in this study by focusing on service-level metrics such as service response times. This study provides valuable information about the effectiveness of supervised machine learning models in detecting anomalies across distributed software systems.

Keywords: anomaly detection; failure detection; microservices; multi-service applications; machine learning; neural networks

Citation: Nobre, J.; Pires, E.J.S.; Reis, A. Anomaly Detection in Microservice-Based Systems. *Appl. Sci.* **2023**, *13*, 7891. https://doi.org/10.3390/app13137891

Academic Editor: Giacomo Fiumara

Received: 15 May 2023
Revised: 21 June 2023
Accepted: 2 July 2023
Published: 5 July 2023

1. Introduction

These days, web applications and platform users are becoming increasingly demanding. In the software industry, designing systems requires thinking about scalability, resiliency, and availability to satisfy users. In the past, most organizations designed their software based on traditional monolithic architectures, where the entire system combines all its functions into a single unit. However, this approach makes changes time-consuming and completely coupled, making it difficult to maintain and evolve. Therefore, it is difficult for modern organizations to update their software with best practices and new technologies.

Software engineers have been adopting microservices architecture to increase their software's development agility, scalability, availability, and reliability, thus keeping up with the demands of their users. A microservices architecture, instead of a monolithic one, is designed based on the idea that a service should represent a software function. Accordingly, each service represents a specific function isolated from the rest of the system. This type of architecture has decoupling as its principle, where each service has its life cycle isolated from the rest of the services. Communication between services is achieved through lightweight mechanisms such as Hypertext Transfer Protocol/Representational State Transfer (HTTP/REST), Remote Procedure Call (RPC), or asynchronous messaging. Intending to be independent, scalable, and replaceable, they employ containers and orchestration tools to allow fast and flexible implementation. Thus, organizations can develop, test, and deploy large and complex applications faster and in a more agile manner when compared to traditional monolithic software [1,2].

With microservices, organizations can adapt quickly to changing market demands as it enables the development of highly scalable, flexible, and available applications. It

also enables faster release cycles due to its modularity. In addition, microservices architecture provides better fault isolation, enabling better resiliency and reducing the impact of potential failures. Studies have shown that microservices architecture can improve the productivity of software engineers by reducing the time to market for new functionality [3].

The independence of the services makes their control and overall management more challenging. As services increase, manual identification and resolution of anomalies become more difficult. The more complex and extensive the software, the more difficult it is to manage and have a global view of the system. This complexity can reduce performance, such as service outages and other quality of service (QoS) issues directly affecting user satisfaction. With each system fault and the alarm raised, a system operator takes time to investigate whether the fault is genuine and, if so, where the cause is. If the fault is not verified, the system operator could have saved time identifying what happened in the event of a false alarm. Thus, setting up automated monitoring and anomaly detection systems becomes essential in microservice-based systems to maintain their health and free operators from other tasks.

Microservices architecture, while offering numerous benefits, also introduces security challenges due to the system's distributed nature. Being a collection of loosely coupled services, each with its communication channels, microservices are susceptible to potential security breaches. Distributed denial-of-service (DDoS) attacks, Advanced Persistent Threat (APT) attacks, and the introduction of malware are some of the security risks that can target microservices. These threats can lead to compromised data integrity, unauthorized access to sensitive information, and disruptions in service availability.

Therefore, organizations must implement robust security measures and monitoring systems in conjunction with automated anomaly detection to ensure the overall health, reliability, and availability of microservices-based systems. Automating monitoring and anomaly detection helps detect performance and functional anomalies and plays a vital role in identifying and mitigating security-related issues. By rapidly identifying, prioritizing, and resolving problems, automated monitoring systems reduce the risk of degraded performance, service downtime, and the impact of potential security breaches.

These automated systems enable organizations to explore the behavior of services over time, identify the root cause of anomalies (both performance-related and security-related), and report them to the operations team. By freeing up the operations team from manual problem detection and diagnosis, these systems allow them to focus on tasks that enhance software quality. Additionally, automated monitoring and anomaly detection systems provide an overall view of the system's health and performance, aiding decision-making regarding sizing, optimization, and maintenance.

While automated anomaly detection systems can improve the health of microservices-based systems, they also face several challenges:

- One of the main challenges in automatically detecting anomalies in microservice-based systems is the high volume and variety of data generated by these systems. Each service generates logs, metrics, and events that can be difficult to correlate and analyze. This fact can lead to false positives and false negatives in anomaly detection.
- The dynamic nature of microservices-based systems: Services can be added, removed, or updated anytime, leading to system behavior changes. In addition, the microservices metrics themselves can vary considerably depending on the temporal context, e.g., an online store may experience usage spikes that modify the metrics considered normal until then and can erroneously lead to false positives.
- The need for real-time detection and response: In highly dynamic systems, anomalies propagate quickly, leading to cascading failures and system downtime. Automatic anomaly detection systems must be able to detect and respond to anomalies in real-time to avoid these failures.
- Validating the efficiency of automatic anomaly detection techniques for microservices-based systems requires benchmarks of microservices systems. However, the availability of open-source microservices benchmarking system, such as Sock-shop [4], is

limited. This makes it time-consuming and impractical for researchers to design and implement their full-scale microservice systems for benchmarking purposes and to train and validate their model. The lack of open-source microservice benchmarking systems also leads to a lack of public datasets that researchers can use to develop and evaluate their innovative operating methods. This consequently hinders the effectiveness of research in this area, as researchers may need access to more data to develop accurate and robust algorithms for real industry system-based microservices.

- Finally, there is a challenge in the interpretability and explainability of automatic anomaly detection systems. Automated anomaly detection systems often use complex machine learning algorithms to detect anomalies, which can be challenging to interpret and explain to system operators. This reality can make it difficult to understand why an anomaly was detected and how to respond to it.

Several authors have worked on this topic. They mainly focused on three types of data sources from microservices architectures: (1) Log-based: based on text reproduced from the logs of the services and the underlying infrastructure [5–10]; (2) Monotoring-based: based on performance metrics data of the services, such as response time, CPU usage, memory usage [11–14]; (3) Distributed Tracing-based: based on traces in text format that the services record with information about the life cycle of the requests to the system in a global view [15–22]. Several methods for anomaly detection have been proposed, such as statistical [23–27], causal inference [28–31], traces comparison [22,32,33], Service Level Objective (SLO) checks [34], Heart Beating [35], machine learning [5–7,11–14,36–39], etc. For instance, Zhou et al. [21] used a technique, MEPFL, that applies *k*-Nearest Neighbors (*K*-NN), Random Forest, and Multi-Layer Perceptron (MLP) to data composed of distributed microservice traces in order to classify whether a trace includes an anomaly, which services are affected by such anomalies, and what types of faults they correspond using an open-source platform provided by the benchmark TrainTicket [40] which contains an experimental study with failure cases where they obtained an accuracy between 0.586–0.984 (0.891 on average) and a recall between 0.647–0.983 (0.821 on average).

Du et al. [38] used the Support Vector Machine (SVM), *k*-NN, Naive Bayes, and Random Forest algorithms to detect anomalies in a microservices system based on monitoring metrics data. They used metrics that cause a Service Level Agreement Violation (SLAV), such as CPU hog, a memory leak, or network packet loss.

Mariani et al. [39] used time series created from multi-service applications hosted on different Virtual Machines (VM) for anomaly detection. The model extracts temporal relationships and trends and then employs Granger causality tests [41] to determine whether there is a correlation between two Key Performance Indices (KPIs), achieving an accuracy of 98.797%.

Based on these concepts, this study presents the experience of developing an automatic anomaly detection system for a microservices-based system that utilizes a MLP. One of the key contributions of this work is the exploration of neural-network-based anomaly detection using service performance monitoring metrics. These metrics are widely employed in industry to monitor services and capture important performance characteristics. This paper aims to demonstrate that by leveraging these service performance metrics, it is possible to detect anomalies using neural networks effectively.

Drawing from the existing literature, a supervised machine learning approach was designed, incorporating an MLP network to accurately and efficiently detect system anomalies from metric service data collected from an open-source microservices-based system, namely Sock-Shop [4]. The proposed approach entails the installation of a microservices system, collection of metric data, preprocessing of the data, extraction of relevant features, and training of the MLP model using labeled data. By capitalizing on the power of neural networks and utilizing the service performance monitoring metrics, the results of this study showcase the effectiveness of the approach in detecting anomalies with reasonable accuracy.

One of the key reasons for selecting the MLP in this study is its capability to learn complex patterns and relationships within the microservices-based system. MLPs are widely

recognized and commonly employed in anomaly detection tasks [42–49] due to their ability to capture non-linear dependencies present in the data [50]. By leveraging the multi-layer structure of MLPs, the system can effectively model the intricate behaviors and interactions of the microservices, thus enhancing the detection of anomalies. Additionally, MLPs are well-suited for handling high-dimensional data, making them suitable for capturing the numerous performance metrics associated with microservices-based systems.

The paper is organized as follows: The Section 2 describes the methodology used to create the datasets, the preprocessing, and the techniques adopted to perform the classification. The Section 3 presents the implemented model and the results. The Section 4 presents the discussion of this work. Finally, the Section 5 presents the conclusions and recommendations for future work.

2. Data and Materials

This section describes the different steps and techniques used to create the dataset. Additionally, the MLP model and metrics used are presented.

2.1. Installation and Setup

Figure 1 illustrates the general design used to create the dataset.

The dataset used for this study was built from the open-source benchmark of a microservices system, Sock-shop [4]. The installation was performed on two VMs. Each VM has 7 CPUs, a 300 Gigabyte (GB) disk, and 14 GB memory. The Sock-shop system is a microservices-structured application that is the user-facing part of an online store that sells socks. It is a polyglot system, characteristic of microservices systems, using technologies, for example, Spring Boot (https://spring.io/(accessed on 1 May 2023)), Golang (https://go.dev/doc/(accessed on 1 May 2023)), and Node.js (https://nodejs.org/(accessed on 1 May 2023)), and is packaged in Docker containers. It comprises seven services: Front-end, Order, Cart, Payment, User, Catalogue, and Shipping. All services communicate using REST over HTTP. Besides REST, the Sock-shop system also includes a queue for asynchronous communication in the Shipping service.

The system was deployed in a Kubernetes (https://kubernetes.io/(accessed on 1 May 2023)) environment. Kubernetes is an open-source system that helps automate, scale, and manage containerized applications, such as Sock-shop. In the case of this study, a Kubernetes cluster was set up with two servers, the two available VMs, described above.

Observability in microservices-based systems is understanding and diagnosing system behavior and performance by collecting, analyzing, and visualizing relevant system data. It typically involves monitoring the interactions between different services, the system infrastructure, and user behavior to extract valuable information for any issues that may arise [51]. In the case of this study, technologies were used for this purpose, in this case, Grafana and Prometheus, which collect metric data regarding the behaviors of the different services.

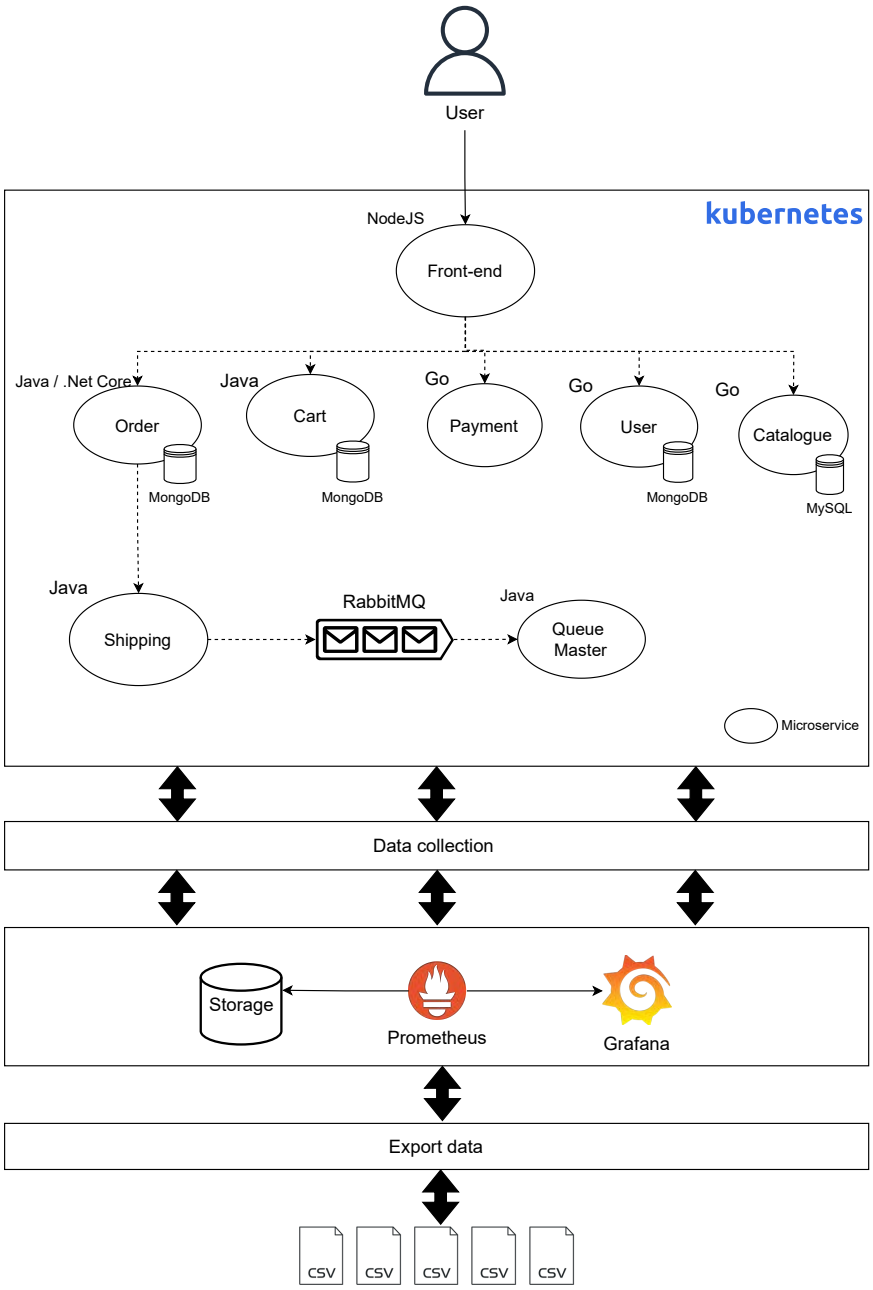

Figure 1. General architecture of the microservices system used with the integration of observability components (Prometheus, Grafana) used to create the dataset for training and model validation.

2.2. Load Testing and Anomaly Creation

After the system was installed, it was simulated to mimic a real-world system. For this, a load-testing framework, Locust (https://locust.io/(accessed on 1 May 2023)), was used. The load-testing scripts were written in Python and simulated normal user usage of an e-commerce system, such as creating users, authenticating users, searching for products, browsing the product catalog, purchasing products. With this, these scripts will simulate the expected behavior of users of this system.

This study creates an anomaly generation module, generating anomalies at both the application and service levels.

Application-level anomalies refer to anomalous behavior that occurs at the level of the entire application. In order to generate this type of anomaly, stress was simulated on the CPU, disk, and memory using *Stress-ng* (https://wiki.ubuntu.com/Kernel/Reference/stress-ng(accessed on 1 May 2023)) software. Increased latency and packet loss on the

network were also generated with *tc* (https://man7.org/linux/man-pages/man8/tc.8.html(accessed on 1 May 2023)) software. As the application is contained in a Kubernetes environment, the anomalies were injected at the server, *Pod* (https://kubernetes.io/docs/concepts/workloads/pods/(accessed on 1 May 2023)), and *DaemonSet* (https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/(accessed on 1 May 2023)) levels.

The service-level anomalies were generated by changing the source code of Orders and Carts to change the normal behavior of the respective services. For this, the *Hystrix* (https://github.com/Netflix/Hystrix(accessed on 1 May 2023)) framework was used, which helps to control the interaction between services to inject anomalies. In order to simulate real-world data, performance and functional anomalies are injected at the service level. Performance anomalies refer to deviations from expected performance metrics, such as request response time. On the other hand, functional anomalies refer to deviations from the expected behavior of the service in terms of its intended functionality or purpose. As illustrated in Algorithm 1, to inject artificial performance anomalies, a procedure was created that injects unspecified waiting time into the processing of each request by the respective services. Consequently, the response time of the respective service will increase, thus creating an artificial performance anomaly. To inject functional anomalies, as seen in the procedure *GenerateFunctionalAnomaly*, an additional procedure that throws an unexpected artificial exception was implemented, so the request is not processed and returns an error, thus creating a functional anomaly. The two types of anomalies were executed in the Orders and Carts services of the created application and, thus, being able to identify in the dataset which services are anomalous.

Algorithm 1 Generate Anomalies in Orders Service.

```
1:  procedure GeneratePerformanceAnomaly
2:      Add latency to simulate performance anomaly
3:      Pause the execution of the program for 5 s
4:  end procedure
5:  procedure GenerateFunctionalAnomaly
6:      Throw exception to simulate functional anomaly
7:      Throw a runtime exception with the message "Error retrieving Orders"
8:  end procedure
9:  procedure GenerateAnomalies
10:     Step 1: Generate Performance Anomaly
11:     Call GeneratePerformanceAnomaly()
12:     Step 2: Generate Functional Anomaly
13:     Call GenerateFunctionalAnomaly()
14: end procedure
```

After performing the anomaly injection procedures, a dataset was created from the service-level performance metrics monitoring metrics with the anomaly types described above.

2.3. Metrics Collection and Visualization

Service monitoring metrics provide a more efficient and scalable way to monitor the performance and behavior of microservices-based systems when compared to logs and traces that can, in turn, quickly become unmanageable and require significant effort to store, analyze, and visualize the information. Metrics are typically more straightforward and focused, allowing for more efficient storage, faster analysis, and easier visualization using tools like Grafana. Moreover, metrics can be collected in real-time, providing more immediate insights into system behavior.

This study uses service-level performance monitoring metrics for anomaly detection. The metrics are stored by Prometheus and visualization and extraction by Grafana. Grafana dashboards are widely used in the industry for monitoring and visualizing metrics from

various data sources because they provide real-time insights into system performance and behavior [52]. The metrics used were the percentiles of response time and the number of requests for each service. These metrics provide valuable insight into the performance and behavior of each service in the system. Percentile metrics allow an understanding of response time distribution for a given service, and quantitative metrics provide information about the volume of requests or queries made to the service. Both metrics can help identify potential bottlenecks, capacity issues, and other system performance problems.

Based on the above ideas, this study creates two datasets to distinguish between detecting application-level and service-level anomalies. The goal is to study the performance of the automatic anomaly detection system in detecting both types of anomalies. Application anomalies, such as high CPU usage, high memory usage, packet loss, and increased network latency, are one focus. The other focus is on service anomalies, such as changing service behavior.

In the realm of microservices architecture, where numerous independent services work together to accomplish a larger goal, automated anomaly detection plays a crucial role in maintaining system stability and performance. To effectively identify anomalies and potential issues, it is essential to choose appropriate metrics that provide valuable insights into the behavior of individual services and the system as a whole. The metrics chosen-—the 90th percentile, 50th percentile, average response time, number of requests returning HTTP 2XX, 5XX, and 4XX response codes—encompass a comprehensive set of measurements that offer a holistic view of the system's performance.

The 90th percentile of the response time for each service is a valuable metric as it captures the behavior of the service under high-load conditions. By considering the response time at the 90th percentile, one is essentially focusing on the upper bound of response times, which helps identify instances where the service may be experiencing significant delays or bottlenecks. Anomalies in this metric can indicate performance degradation, potential failures, or issues that require attention.

Similarly, the 50th percentile (also known as the median) provides insights into the typical service response time. This metric helps detect anomalies in the baseline behavior, highlighting variations in the response time that may be indicative of issues such as increased load, changes in service dependencies, or suboptimal performance.

The average response time is a fundamental metric that offers an overall measure of the service's performance. While the 50th percentile focuses on typical behavior, the average response time provides a broader perspective by considering all response times. Anomalies in this metric can help detect significant shifts in performance trends or identify service-level issues that affect the average response time across requests.

The number of requests returning HTTP 2XX (refers to the range of successful response codes, such as 200 OK), 5XX (represents server error codes, such as 500 Internal Server Error), and 4XX (corresponds to client error codes, such as 400 Bad Request) response codes are crucial metrics for monitoring the health and reliability of the microservices. A high number of HTTP 5XX or 4XX response codes suggests server errors or client-related issues, respectively, which may require immediate attention. Conversely, a consistently high number of HTTP 2XX response codes indicates successful requests, providing insights into the system's stability and functionality.

These metrics collectively offer valuable information about the system's performance, enabling the detection of anomalies, performance degradation, and potential issues. While other metrics may also be relevant, the chosen set provides a solid foundation for monitoring and maintaining the stability and reliability of microservices.

Tables 1 and 2 represent an excerpt of the datasets created for application and service anomaly detection. Both datasets have the same structure, while, unlike service anomalies, application anomalies affect the entire system; thus, all services will have their metrics changed. The samples that make up the dataset were collected at 30-minute time intervals. The normal data were collected without the fault injection, and the user simulation was achieved from the load tests. The "Time" column represents the timestamp at which the

log was collected. The "IsError" column represents the label generated for the supervised machine learning algorithm to determine whether a record is anomalous. These data were extracted to a CSV file using the metrics described above.

Table 1. An excerpt from the dataset created for training and validation of the system for the application-level anomalies.

Time	99th Percentile	50th Percentile	Mean	2xx [1]	4xx/5xx [2]	IsError
2023-02-27 18:46:00	0.01390	0.00257	0.00234	164	0	True
2023-02-27 18:46:30	0.00906	0.00264	0.00257	167	0	True
2023-02-27 18:47:00	0.05550	0.00253	0.00318	173	0	True
2023-02-23 18:37:30	0.00953	0.00280	0.00304	187	0	False
2023-02-23 18:38:00	0.01610	0.00276	0.00309	262	0	False
2023-02-23 18:38:30	0.00989	0.00273	0.00303	264	0	False

[1] Represents the number of responses with the HTTP response code in the 200s of the services. [2] Represents the number of HTTP responses with the response code in the 400s or 500s of the services.

Table 2. An excerpt from the dataset created for training and validation of the system for the service-level anomalies.

Time	99th Percentile	50th Percentile	Mean	2xx [1]	4xx/5xx [2]	IsError
2023-02-22 21:19:00	4.920	2.030	2.070	24.4	0	True
2023-02-23 15:44:00	0.498	0.375	0.388	24.4	0	True
2023-02-22 21:19:30	2.470	1.200	1.050	20.0	0	True
2023-03-21 14:08:00	0.00495	0.00250	0.000578	604.0	0	False
2023-03-21 13:54:30	4.740	0.00324	0.317	898.0	6.67	False
2023-02-27 19:08:00	7.380	2.200	2.610	31.1	0	False

[1] Represents the number of responses with the HTTP response code in the 200s of the services. [2] Represents the number of HTTP responses with the response code in the 400s or 500s of the services.

2.4. Dataset Labeling

The pre-processing phase starts by labeling the records as anomalous and normal, adding in the metrics CSV the IsError column, represented by a boolean (true is anomalous, false is normal). After labeling, all types of application anomalies were added in a single CSV representing application anomalies. In the case of application anomalies, a column with the identifier of the respective record was also added.

The application-level anomaly dataset contains 3430 records, of which 1715 represent normal and 1715 represent abnormal records. Figure 2 plots the data, two features simultaneously. The diagonal plots show the univariate distribution data in each column. The service-level anomaly dataset contains 2074 records, of which 1043 represent normal and 1031 represent abnormal records. Figure 3 displays the feature plots for the service-level anomaly dataset.

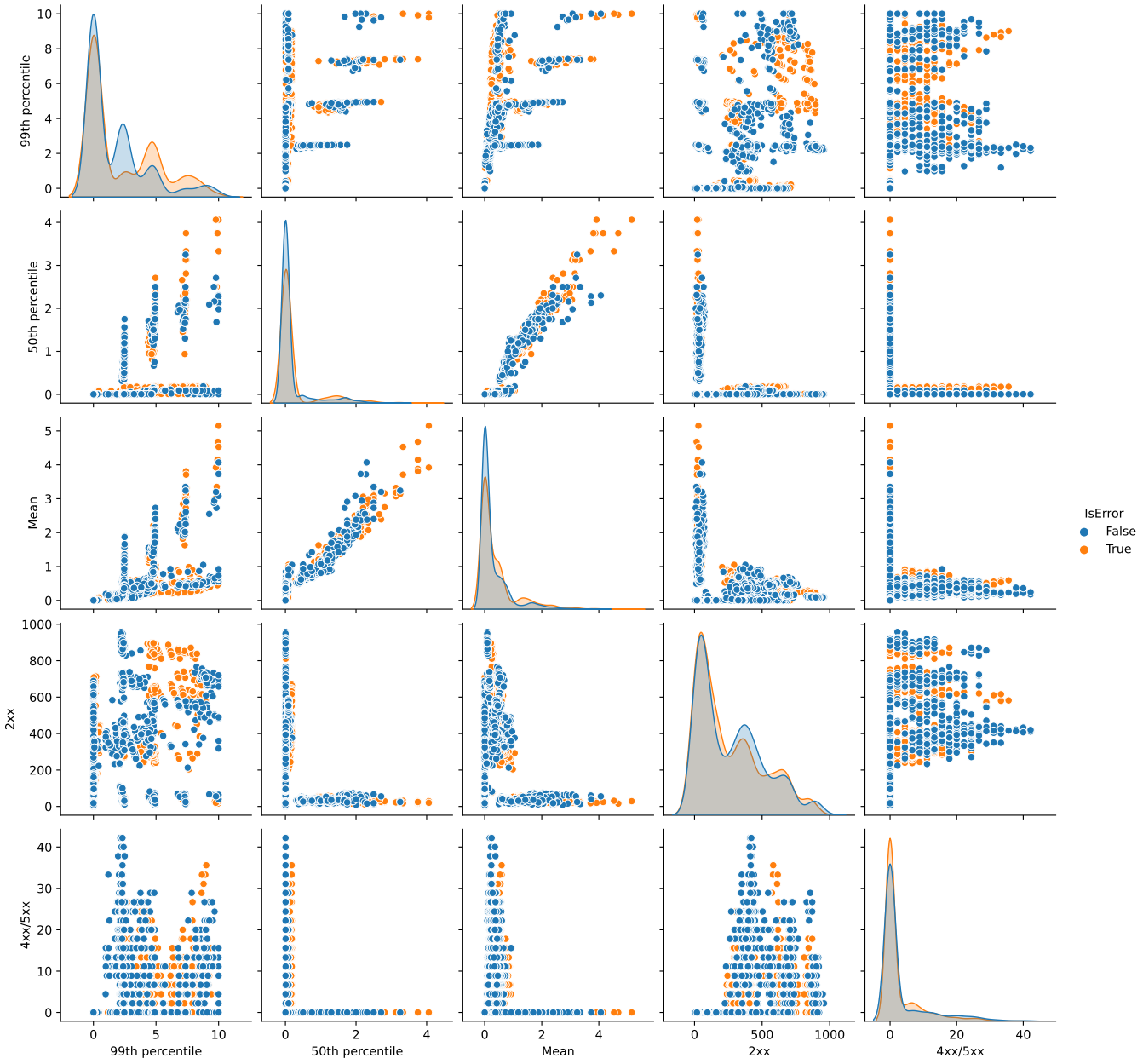

Figure 2. Representation of the pairwise relationships of the application-level anomaly dataset.

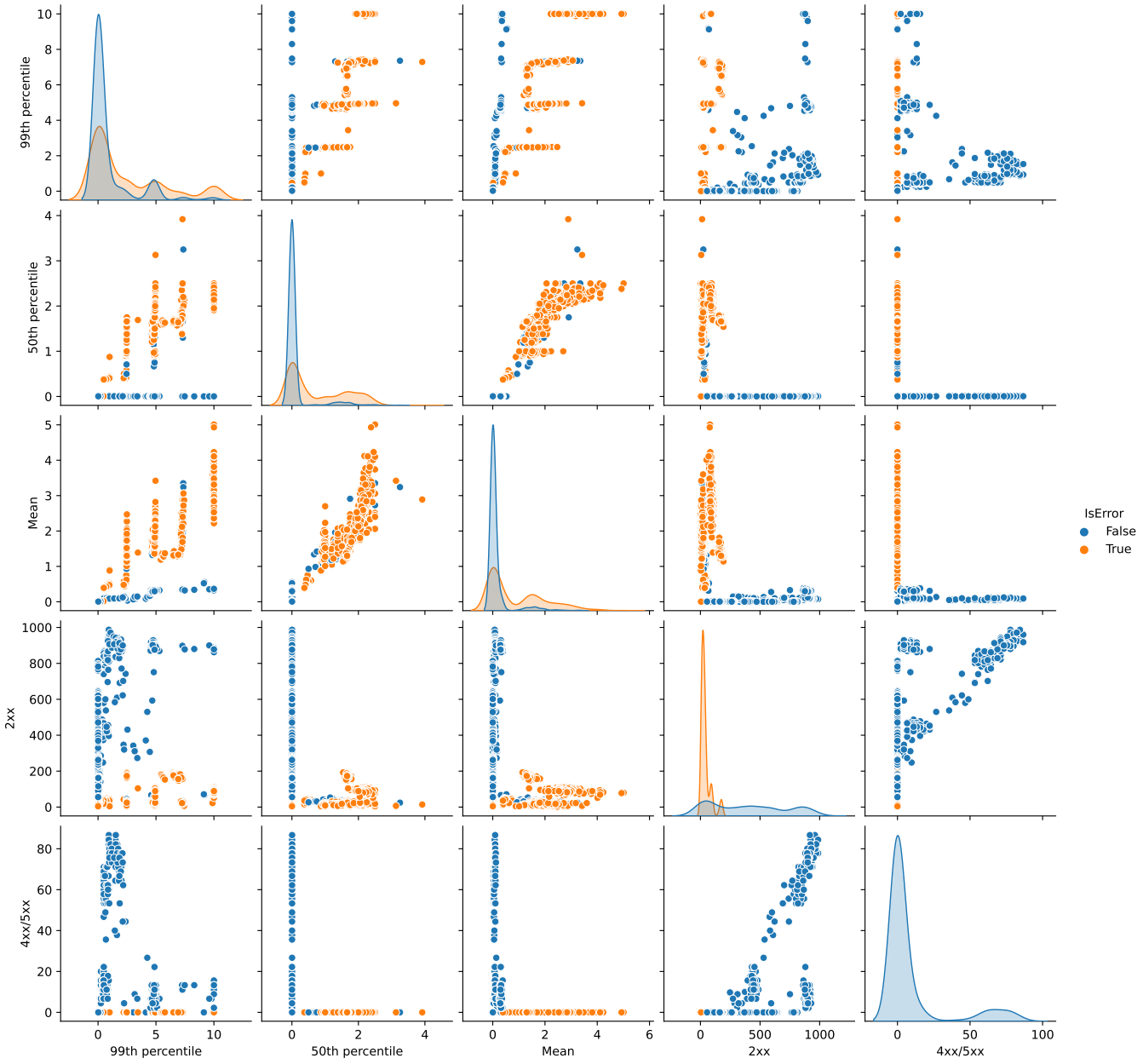

Figure 3. Representation of the pairwise relationships of the service-level anomaly dataset.

2.5. Neural Networks

Multi-layer Perceptron (MLP) neural networks are widely employed in anomaly detection tasks, mainly when dealing with small datasets that exhibit non-linear relationships [53]. In the context of monitoring data in a microservices system, where complex relationships may exist between different metrics, MLPs offer a suitable modeling approach. MLPs are known for their flexibility and ability to capture intricate patterns in data [54], making them well-suited for handling the diverse and interconnected nature of monitoring data in microservice architectures.

The structure of an MLP consists of an input layer, one or more hidden layers, and an output layer [55]. Each layer comprises multiple artificial neurons or nodes that process incoming data and pass it through a series of weighted connections. Non-linear activation functions, such as the sigmoid or rectified linear unit (ReLU), are typically applied to introduce non-linearity and enable the network to capture complex relationships. During

training, the network learns to adjust the weights of the connections through backpropagation, iteratively refining its ability to map input data to the desired output.

Monitoring data in a microservices system often suffers from incompleteness or errors due to various factors. MLPs can handle such noisy data gracefully, thanks to their inherent ability to generalize and extract meaningful patterns from imperfect information [56,57]. By learning from the available data, MLPs can identify abnormal patterns and flag them as potential anomalies, even in the presence of noise.

In the context of anomaly detection tasks, MLP neural networks have been extensively employed to detect deviations from normal behavior in various domains. Researchers and practitioners have successfully applied MLPs to monitor system logs [58], network traffic [59,60], sensor data [61,62], and other sources of information to identify anomalies. By training MLPs on historical data that encompasses normal behavior, the networks learn to recognize patterns and predict expected outcomes. During the testing phase, when presented with unseen data, MLPs can compare the observed behavior against what they have learned and detect anomalies that deviate significantly from the expected patterns. This ability to learn from historical data and generalize to new instances makes MLPs valuable tools for anomaly detection in microservices monitoring systems.

MLP neural networks offer a powerful approach to anomaly detection in microservices monitoring. Their flexible architecture, capable of capturing non-linear relationships, makes them suitable for handling complex interdependencies within monitoring data. Additionally, their robustness to noisy data enables them to operate effectively in real-world scenarios where data quality may be compromised. By leveraging MLPs, anomaly detection systems can uncover hidden anomalies and contribute to microservices architectures' overall stability and reliability.

2.6. Classifier Performance

This work used four metrics to evaluate the classifier's performance: Accuracy, Recall, Precision, and F1-score. These statistical metrics can be calculated using the following equations:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{5}$$

where True Positives (TP) represent the number of anomalies that the model detects as anomalous, True Negatives (TN) represent the number of anomalies that the model detects as normal, False Positives (FP) denote the number of normal records that the model detects as normal, and False Negatives (FN) are the number of abnormal records that the model detects as normal.

This study uses 80% of the data for training, and 20% to test the model.

3. Model and Results

Figure 4 illustrates the step-by-step process of the model MLP created for anomaly detection in microservices systems. The first step involves loading and pre-processing the data. Then the features are normalized using MinMaxScaler. Hyperparameter tuning is performed using GridSearchCV to optimize the performance of the model. After the hyperparameters are tuned, the final MLP classifier is trained using the training set. Finally,

the performance of the classifier is evaluated on both the training and validation sets. The approach and its evaluation is detailed in the following subsections.

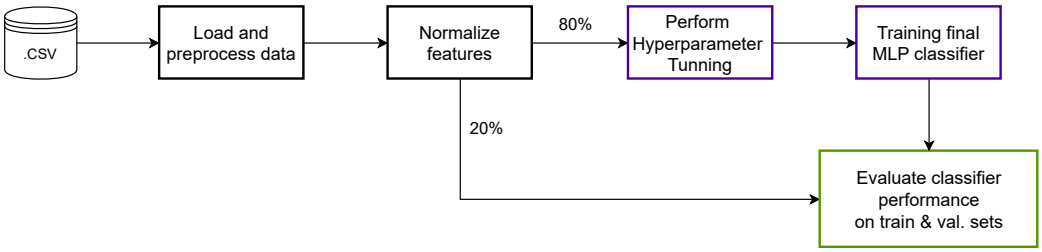

Figure 4. Illustration of the steps involved in the MLP Classification model procedure.

3.1. Data Preprocessing

Data preprocessing plays a crucial role in preparing the dataset for anomaly detection. Data preprocessing aims to transform the raw transaction data into a format suitable for training a machine learning model, ensuring improved accuracy and performance in anomaly detection.

This study's first data pre-processing step involves loading the data from the CSV files generated during the collection and visualization phase (Section 2.3). The dataset includes the features described in Tables 1 and 2. The "Time" column is removed from the dataset as part of the preprocessing steps. This column indicates the time at which each transaction occurred. Although valuable for certain analyses, it is considered irrelevant for identifying anomalous transactions based on their characteristics. Removing this column simplifies the data and reduces computational complexity without compromising the classifier's accuracy.

Subsequently, the dataset is divided into features and target variables. The features represent the transaction characteristics, while the target variable indicates whether a transaction is normal or anomalous. In the case of this dataset, the characteristics are 99th percentile, 50th percentile, Mean, 2xx, and 4xx/5xx. At the same time, the target variable is "IsError". This step facilitates a clear distinction between the input data and the expected output, enabling the machine learning model to learn patterns associated with normal transactions.

To address the challenge of varying scales and ranges in the dataset, particularly in features such as 99th percentile, 50th percentile, Mean, 2xx, and 4xx/5xx, MinMaxScaler [63] is applied. These features exhibit different numerical values ranging from small decimals to larger numbers.

The decision to use MinMaxScaler over other alternatives, such as Z-score [64] and RobustScaler [65], is motivated by several factors that make it well-suited for this study's anomaly detection task. First, MinMaxScaler offers a straightforward implementation, making it easy to incorporate into the preprocessing pipeline. Its simplicity reduces the complexity of the overall process. Another reason for selecting MinMaxScaler is its computational efficiency. It is known to work fast, enabling efficient scaling of feature values, which is particularly advantageous when dealing with large datasets or when repeated preprocessing is necessary.

Likewise, MinMaxScaler effectively addresses the varying scales and ranges of the features in the dataset. Transforming the feature values to a common range of [0, 1] ensures that each feature contributes equally during training. This fact is crucial, especially when employing distance-based algorithms, as it prevents certain features from dominating others due to disparities in their original scales. Additionally, MinMaxScaler preserves the shape of the original distribution while normalizing the data. This characteristic is essential for maintaining the integrity of the dataset and enabling accurate anomaly detection based on the underlying patterns.

After normalizing the data, a stratified sampling technique is employed to split the dataset into training datasets (80%) and validation datasets (20%). This split ensures

that each class of the target variable, representing normal and anomalous transactions, is proportionally represented in both the training and validation sets. Training the classifier on a balanced dataset aims to improve its performance and generalizability.

3.2. Hyperparameter Tuning

Hyperparameters are essential configuration settings determined before training a machine learning model and not learned during the training process. This study employs a dynamic method of selecting hyperparameter values, utilizing GridSearchCV [66]. GridSearchCV is a popular hyperparameter tuning technique that involves searching over a specified grid of hyperparameters and evaluating the model's performance for each combination. The approach was influenced by the work of Gonzalez-Cuautle et al. [67], which proposed a resampling method called SMOTE combined with a grid-search algorithm optimization procedure for optimizing classification tasks in botnet and intrusion-detection-system datasets.

To identify the optimal hyperparameters for the MLP model, GridSearchCV is compared with the Bayes algorithm [68]. Bayes algorithm is a probabilistic model that leverages Bayes' theorem to estimate class probabilities based on input features. While Bayes is known for its robustness, it may require a larger dataset to achieve optimal performance and could be less effective on smaller datasets like that used in this study. Consequently, GridSearchCV was chosen as the preferred method due to its comprehensive exploration of the hyperparameter space, ensuring effective model performance on the validation set and mitigating the risk of overfitting.

In order to perform hyperparameter tuning using GridSearchCV, an MLPClassifier object was instantiated with 1000 iterations and a random seed. A GridSearchCV object was created with the MLPClassifier and a predefined hyperparameter grid. The hyperparameter grid encompasses various aspects of the MLP model configuration, including the activation function, hidden layer sizes, solver algorithm, learning rate initialization, and L2 penalty parameter.

The range of values explored for each hyperparameter in the grid search were as follows:

- Activation function: The activation function determines the output of a neuron and plays a crucial role in modeling complex non-linear relationships. Two activation functions were considered: "tanh" (hyperbolic tangent) and "relu" (rectified linear unit). "tanh" is known for mapping inputs to the range $(-1, 1)$, while "relu" provides a range of [0, inf). By exploring both options, the grid search evaluated the impact of different activation functions on the model's performance.
- Hidden layer sizes: The hidden layer sizes define the number of neurons in each hidden layer of the MLP. The grid search examined three configurations: (100), (50, 50), and (50, 100, 50). These configurations represent the number of neurons in one, two, and three hidden layers. By varying the hidden layer sizes, the grid search assessed the influence of different network architectures on the model's ability to capture complex patterns.
- Solver algorithm: The solver algorithm determines the optimization strategy for weight optimization. Two solvers were included in the grid search: "sgd" (stochastic gradient descent) and "adam" (adaptive moment estimation). "sgd" updates the weights using a subset of training samples at each iteration. At the same time, "adam" adapts the learning rates based on previous gradients. By evaluating both solvers, the grid search investigated the impact of different optimization algorithms on the model's convergence and performance.
- Learning rate initialization: The learning rate determines the step size taken during weight updates. Three learning rate initialization values were considered: 0.01, 0.001, and 0.0001. Higher learning rates enable faster convergence but may risk overshooting the optimal weights, while lower learning rates may converge slowly. The grid search examined the trade-off between convergence speed and accuracy by exploring multiple learning rates.

- L2 penalty parameter (alpha): The L2 penalty parameter controls the regularization strength, preventing overfitting by adding a penalty term to the loss function. Three alpha values were explored: 0.1, 0.01, and 0.001. Higher alpha values impose more robust regularization, reducing the risk of overfitting but potentially sacrificing model performance on the training set. By varying the alpha values, the grid search assessed the model's sensitivity to regularization and aimed to strike a balance between fitting the training data and generalizing to unseen data.

By searching through this range of values, GridSearchCV thoroughly explores different configurations of the MLP model and identifies the combination of hyperparameters that yields the highest validation accuracy. This set of optimal hyperparameters is then used to create the final MLP model, which is expected to improve performance on unseen data.

3.3. Training the Final Classifier

The MLP neural network is created with the hyperparameters calculated in the previous step, the number of iterations, in this case 1000 iterations, and a random seed is also set. The size of the MLP is one of the hyperparameters calculated by GridSearchCV. In the training phase, the neuronal network uses the tuning method that updates the model weights using an optimization algorithm, such as stochastic grand descent. More specifically, this model uses backpropagation [54] to update the weights during training. Backpropagation computes the gradient of the loss function concerning the model weights and then updates the weights in the opposite direction of the gradient to minimize the loss. This process is repeated for several epochs (i.e., it runs over the training data) until convergence. This study uses cross-validation by running the model 10 times with different random seeds to get an estimate of performance variation due to randomness in the data. A more robust estimate of the algorithm's performance can be obtained by averaging the results over multiple runs.

3.4. Evaluating Classifirer Performance

Application-level anomaly detection: In the application-level anomalies dataset, there are 3430 records, among which 1715 are control records, and 1715 are abnormal records. The validation results are shown in Table 3.

Table 3. Validation results of the application-level anomaly datasets.

Run	Train. [5] Acc. [1]	Val. [6] Acc.	Train. Prec. [2]	Val. Prec.	Train. Rec. [3]	Val. Rec.	Train. F1 [4]	Val. F1	Val. FPR
1	0.72	0.70	0.66	0.66	0.87	0.88	0.75	0.75	0.37
2	0.75	0.73	0.78	0.77	0.70	0.64	0.74	0.70	0.31
3	0.69	0.67	0.65	0.62	0.86	0.85	0.74	0.71	0.44
4	0.72	0.70	0.66	0.65	0.89	0.89	0.76	0.75	0.36
5	0.74	0.73	0.66	0.65	0.98	0.96	0.79	0.78	0.27
6	0.70	0.67	0.71	0.69	0.68	0.66	0.69	0.67	0.41
7	0.72	0.69	0.66	0.64	0.90	0.87	0.76	0.74	0.41
8	0.75	0.76	0.77	0.74	0.73	0.75	0.75	0.75	0.28
9	0.72	0.72	0.73	0.75	0.69	0.68	0.71	0.71	0.41
10	0.73	0.72	0.76	0.77	0.67	0.62	0.72	0.69	0.32
Max	0.75	0.76	0.78	0.77	0.98	0.96	0.79	0.78	0.44
Min	0.69	0.67	0.65	0.62	0.67	0.62	0.69	0.67	0.27
Mean	0.72	0.71	0.70	0.69	0.80	0.78	0.74	0.73	0.36
Std.dev.	0.02	0.03	0.05	0.06	0.11	0.12	0.03	0.03	0.06

[1] Accuracy [2] Precision [3] Recall [4] F1-score [5] Training [6] Validation.

Regarding the validation results in Table 3, the mean values of accuracy, precision, recall, F1-score, and false positive ratio (FPR) from 10 validation runs were 71%, 69%, 78%, 73%, and 0.06, respectively. Of these metrics, recall stands out as the highest, with a mean value of 78%, indicating that the model could correctly identify a high percentage of

anomalies. On the other hand, FPR had the lowest mean value of 0.06%, indicating that the model had a low rate of false positives.

The variability from the mean of the metric results across the ten validation runs ranged from 0.02 to 0.11, showing that the model's performance was consistent across multiple runs. In addition, the model achieved a maximum F1 score of 98% in the validation phase, indicating high accuracy and recall. The minimum F1 score obtained was 96%, demonstrating that the model consistently performed well on this metric.

Taking as an example the run that obtained the highest F1-score metric value in the validation phase, the MLP hyperparameters chosen by the GridSearch algorithm (Section 3.2) chose the ReLU function as the network activation function. Set the L2 penalty parameter to a value of 0.001. It defined three hidden layers, each with 100, 25, and 25 neurons. Furthermore, it defined the Adam as the optimization algorithm used by the neural network.

Figure 5a displays the confusion matrix for the average results of the ten validation runs for the application-level anomaly dataset, shown in Table 3. The matrix indicates that, on average, there were 266.6 TP, 219.8 TN, 124.3 FP, and 75.3 FN cases. In comparison, the false negative rate was 11%, indicating that 11% of cases were incorrectly identified as negative.

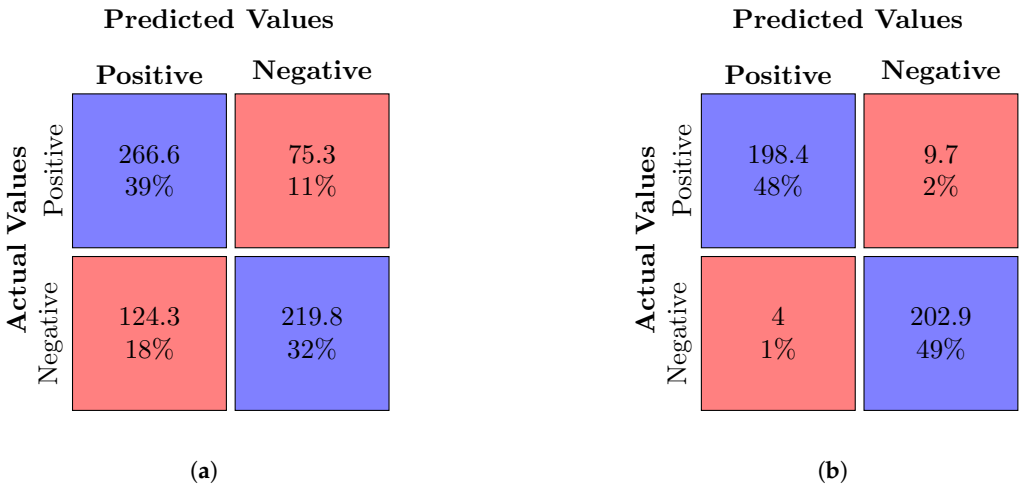

Figure 5. Comparison of confusion matrices of the model with the dataset with application-level and service-level anomalies. (**a**) Confusion matrix for the application-level anomaly dataset. (**b**) Confusion matrix for the service-level anomaly dataset.

Service-level anomaly detection: The service-level anomalies dataset consists of 2074 records, of which 1031 are anomalous and 1043 are control.

Table 4 presents the validation results for the service-level anomalies dataset, where the mean values of accuracy, precision, recall, F1-score, and false positive ratio (FPR) from ten runs were 97%, 95%, 98%, 97%, and 0.01, respectively. Among these metrics, recall stood out with the highest average of 98%, indicating that the model could accurately identify a high percentage of anomalies. In contrast, FPR had the lowest average value of 0.01%, indicating that the model had a low false-positive rate. The validation results suggest that the model performed well in detecting anomalies. The high recall value indicates that the model effectively identified most anomalies, while the low FPR value indicates that the model produced few false positives. These results support the use of the model in anomaly detection applications.

Table 4. Validation results of the service-level anomaly datasets

Run	Train. [5] Acc. [1]	Val. [6] Acc.	Train. Prec. [2]	Val. Prec.	Train. Rec. [3]	Val. Rec.	Train. F1 [4]	Val. F1	Val. FPR
1	0.98	0.97	0.96	0.95	0.99	0.99	0.98	0.97	0.00
2	0.97	0.97	0.97	0.97	0.98	0.98	0.97	0.97	0.00
3	0.97	0.96	0.95	0.94	0.99	0.99	0.97	0.96	0.00
4	0.96	0.96	0.94	0.93	0.98	0.98	0.96	0.96	0.01
5	0.96	0.97	0.94	0.94	0.98	0.99	0.96	0.97	0.02
6	0.96	0.96	0.95	0.94	0.98	0.98	0.96	0.96	0.01
7	0.97	0.97	0.97	0.96	0.98	0.98	0.97	0.97	0.00
8	0.98	0.97	0.97	0.96	0.98	0.98	0.98	0.97	0.00
9	0.98	0.96	0.97	0.97	0.98	0.96	0.98	0.96	0.00
10	0.97	0.98	0.96	0.98	0.98	0.98	0.97	0.98	0.02
Max	0.98	0.98	0.97	0.98	0.99	0.99	0.98	0.98	0.02
Min	0.96	0.96	0.94	0.93	0.98	0.96	0.96	0.96	0.00
Mean	0.97	0.97	0.96	0.95	0.98	0.98	0.97	0.97	0.01
Std.dev.	0.01	0.01	0.01	0.02	0.00	0.01	0.01	0.01	0.01

[1] Accuracy [2] Precision [3] Recall [4] F1-score [5] Training [6] Validation.

Figure 5b displays the confusion matrix for the average of the 10 dataset runs with service-level anomalies. The values of TP, TN, FP, and FN were 198.4, 202.9, 4, and 9.7, respectively. These values correspond to 48% TP, 49% TN, 1% FP, and 9.7% FN. These results suggest that the model was able to classify the majority of the data points effectively. The high percentages of TP and TN indicate that the model had good overall performance in identifying both positive and negative cases. The low percentages of FP and FN indicate that the model had low rates of misclassifying data points. These results support the use of the model in detecting application-level anomalies.

The standard deviation of the ten runs concerning the mean was between 0.01 and 0.02 for all the evaluated metrics. Precision had the largest variability with a value of 0.02, while recall had the smallest one with 0. These results suggest that the model's performance was relatively consistent across the ten runs for all the metrics, except for precision which showed greater dispersion. The low standard deviation in recall suggests that the model was able to consistently identify a high percentage of TP in each run. However, the higher standard deviation in precision indicates that the model's ability to avoid false positives may have been less consistent. These insights into the model's standard deviation can help understand its strengths and weaknesses and can inform further improvements and refinements to the model.

Among the 10 runs evaluated, the highest F1-score value was obtained in run ten. The hyperparameters used in this run were: the activation function tanh, an L2 penalty parameter of 0.001, a network architecture composed of two layers with 100 and 50 neurons, respectively, an initial learning rate of 0.001, and the Adam for weight optimization. These results suggest that the combination of hyperparameters used in run ten was particularly effective in achieving a high F1-score. Further analysis could be conducted to understand the impact of each individual hyperparameter on the model's performance. These findings highlight the importance of tuning hyperparameters in machine learning models to achieve optimal performance.

4. Discussion

The presented results demonstrate the performance of supervised machine learning models, in this case, a Multi-Layer Perceptron (MLP), in detecting anomalies at two levels: application and service. The models were trained on datasets with a balanced number of normal and abnormal records, and their accuracy, precision, recall, F1 score, and false positive rate (FPR) validation metrics were reported.

Among various techniques used for anomaly detection [69], including Random Forest [70], and SVM [71], MLP emerges as a strong candidate for anomaly detection in

microservices-based systems. The complexity of relationships in microservice architectures requires a model that can effectively capture intricate patterns and dependencies. MLPs are known for their ability to handle complex nonlinear relationships [42], making them suitable for these scenarios. Additionally, MLPs excel at feature extraction and representation learning, automatically discovering meaningful, multidimensional representations observed in microservice architectures. Furthermore, the adaptability of MLPs to dynamic environments aligns well with the nature of microservice architectures, which often undergo changes in service behavior and interactions over time. The positive results of MLPs in intrusion detection systems [72] also suggest their potential effectiveness in anomaly detection across various domains.

In the context of the presented study, the application-level anomaly detection model achieved an overall average accuracy of 71%, with an average F1 score of 73%. The recall value, which measures the proportion of actual anomalies correctly identified by the model, was the highest among the reported metrics, indicating that the model can detect most anomalies. The proportion of FP was low, indicating that the model produced very few FP. These results suggest that the model can detect anomalies based on anomalies injected at the application level, which is critical to ensuring the safety and reliability of these system operators.

On the other hand, the anomaly detection model with the service-level anomalies performed even better than the application level, achieving an average accuracy of 97% and an average F1 score of 97%. The recall value was the highest among the reported metrics, indicating that the model can detect most application anomalies. The proportion of FP was deficient, indicating that the model produced few FP. These results suggest that the model has great potential for detecting anomalies in microservices, which ensures greater reliability and specificity for operators running such systems.

When comparing the performance of MLP on the service-level anomaly dataset with the application-level anomaly, it is clear that the model performs significantly better on the former than on the latter. This difference is evident from the higher accuracy, recovery, and F1 score values obtained on the service-level dataset compared to the application-level dataset. Furthermore, the FPR value obtained from the service-level dataset was significantly lower than the FPR value obtained from the application-level dataset, indicating that the model is better at distinguishing between normal and anomalous data at the service level.

One possible reason for this performance difference can be attributed to the different nature of the anomalies in the two datasets. The service-level dataset injects performance and functional anomalies by changing the code directly in the services. In contrast, the application-level dataset injects anomalies by stressing the application's infrastructure (CPU, disk, memory, and network). As such, in application anomalies, the entire application goes into anomalous mode by changing the metrics of all services at the same time, and in service-level anomalies, the behavior is only changed in only a specific quantity of services, which can make it easier for the model to detect anomalies as it specifically looks for changes in performance monitoring metrics of each service. In addition, the application-level dataset injects anomalies into the Kubernetes environment, which requires the model's greater understanding of the impact of stress on service metrics.

The observed difference in performance between the application-level and service-level anomaly detection models may also be supported by the visual analysis of pairwise relationships between variables in each dataset. As shown in Figures 2 and 3, the distribution of anomalies in the service-level dataset appears more distinct and separable from normal cases than the distribution of anomalies in the application-level dataset. Specifically, while the separation between positive and negative cases is not entirely clear in either dataset, it is comparatively more straightforward in the service-level dataset than in the application-level dataset. This fact suggests that the nature of the anomalies in the two datasets may contribute to the observed difference in performance between the two models.

In practice, this study can be applied to monitor and manage distributed systems effectively. The ability to detect anomalies in distributed systems is critical to maintaining system performance, reliability, and security. This research suggests that developing anomaly detection systems focusing on application-level metrics can provide better results, enabling more effective monitoring and management of distributed systems.

A limitation of this study is the absence of benchmarks for applications built on microservices. The datasets used in the study were created based on a small application, which may not fully capture the complexity and diversity of real-world data. In addition, the data used to build the datasets were synthetic, which may further reduce the variety and complexity of the cases represented in the datasets.

5. Conclusions and Future Work

In this study, we investigated the performance of a supervised machine learning model, specifically an MLP, in detecting microservices anomalies at the application and service levels. The results demonstrate the model's effectiveness in detecting anomalies in both domains, with higher accuracy, precision, recovery, and F1 score obtained in the service-level dataset. These findings have practical implications for automating the monitoring and management of distributed systems, particularly microservices, by leveraging service-level metrics such as percentiles of service response times and the number of service responses.

By focusing on service-level metrics, the model has the potential to enable more effective automation of anomaly detection, providing system operators with actionable insights. Identifying anomalous services can facilitate decision-making and prompt appropriate remedial actions. For instance, in the case of service-level anomalies, the model can detect the presence of an anomaly and pinpoint the specific service responsible for the anomalous behavior. This capability empowers system operators to quickly identify and address issues, thereby enhancing the reliability and efficiency of distributed systems.

In addition to these practical implications, future research directions can explore the model's suitability for incremental or online learning scenarios. An incremental learning approach would enable the model to continuously update its knowledge and adapt to evolving performance data, ensuring its effectiveness in dynamic microservices environments. This capability would allow the model to handle gradual changes, new patterns, or emerging anomalies, thereby contributing to real-time anomaly detection and system management. Furthermore, it would be valuable to compare the performance of MLP with other supervised machine learning techniques, such as *K*-Nearest Neighbors (*K*-NN) and Random Forest, in the context of microservices anomaly detection. Comparative studies involving multiple algorithms would provide insights into their strengths, weaknesses, and trade-offs, considering detection accuracy, computational efficiency, and interpretability. Such a comparative analysis would contribute to a better understanding of the most suitable algorithm for anomaly detection in microservice architectures and guide the selection and deployment of effective models in real-world scenarios.

Furthermore, there is a need to develop robust benchmarks that closely resemble real-world microservices applications. By creating more publicly available and diverse examples, future studies can enhance the evaluation and comparison of anomaly detection models, fostering advancements in the field. Additionally, efforts can be directed toward extending the model's capabilities to identify anomalies and provide automated decision support to system operators. This would involve incorporating intelligent decision-making mechanisms that leverage the model's insights to guide operators in making informed choices.

Overall, this study provides valuable insights into the effectiveness of supervised machine learning models for anomaly detection in distributed systems, specifically microservices. The findings highlight the potential for practical implementation, emphasizing the automation of monitoring and management processes. As future research progresses, the model's adaptability for incremental learning, the development of robust benchmarks, and the integration of decision support mechanisms can further enhance its utility and contribute to anomaly detection in distributed systems.

Author Contributions: Conceptualization, J.N.; methodology, J.N.; software, J.N.; validation, J.N., E.J.S.P. and A.R.; formal analysis, J.N.; investigation, J.N.; data curation, J.N.; writing—original draft preparation, J.N.; writing—review and editing, J.N., E.J.S.P. and A.R.; supervision, E.J.S.P. and A.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: The data supporting the reported results can be found at https://doi.org/10.6084/m9.figshare.22726298.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

FP	False Positive
FN	False Negative
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
HTTP/REST	Hypertext Transfer Protocol/Representational State Transfer
k-NN	*k*-Nearest Neighbors
KPI	Key Performance Indices
MLP	Multi-Layer Perceptron
QoS	Quality of Service
RPC	Remote Procedure Call
SLAV	Service Level Agreement Violation
SLO	Service Level Objective
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
VM	Virtual Machine

References

1. Lewis, J.; Fowler, M. Microservices: A Definition of This New Architectural Term. 2014. Available online: https://martinfowler.com/articles/microservices.html. (accessed on 4 May 2023).
2. Newman, S. *Building Microservices*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2021.
3. Mazzara, M.; Bucchiarone, A.; Dragoni, N.; Rivera, V. Size matters: Microservices research and applications. *Microservices: Science and Engineering*; Springer: Cham, Switzerland, 2020; pp. 29–42.
4. Weaveworks. Sock Shop: A Microservice Demo Application. 2016. Available online: https://microservices-demo.github.io/ (accessed on 4 May 2023).
5. Yagoub, I.; Khan, M.A.; Jiyun, L. IT equipment monitoring and analyzing system for forecasting and detecting anomalies in log files utilizing machine learning techniques. In Proceedings of the 2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD), Durban, South Africa, 6–7 August 2018; pp. 1–6.
6. Brown, A.; Tuor, A.; Hutchinson, B.; Nichols, N. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In Proceedings of the First Workshop on Machine Learning for Computing Systems, Tempe, AZ, USA, 12 June 2018; pp. 1–8.
7. Nandi, A.; Mandal, A.; Atreja, S.; Dasgupta, G.B.; Bhattacharya, S. Anomaly detection using program control flow graph mining from execution logs. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 215–224.
8. Jia, T.; Yang, L.; Chen, P.; Li, Y.; Meng, F.; Xu, J. Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs. In Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, HI, USA, 25–30 June 2017; pp. 447–455.
9. Fu, Q.; Lou, J.G.; Wang, Y.; Li, J. Execution anomaly detection in distributed systems through unstructured log analysis. In Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, Miami Beach, FL, USA, 6–9 December 2009; pp. 149–158.

10. Du, M.; Li, F.; Zheng, G.; Srikumar, V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1285–1298.
11. Sharma, B.; Jayachandran, P.; Verma, A.; Das, C.R. CloudPD: Problem determination and diagnosis in shared dynamic clouds. In Proceedings of the 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, 24–27 June 2013; pp. 1–12.
12. Zhang, X.; Meng, F.; Chen, P.; Xu, J. Taskinsight: A fine-grained performance anomaly detection and problem locating system. In Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 27 June–2 July 2016; pp. 917–920.
13. Xu, H.; Chen, W.; Zhao, N.; Li, Z.; Bu, J.; Li, Z.; Liu, Y.; Zhao, Y.; Pei, D.; Feng, Y.; et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In Proceedings of the 2018 World Wide Web Conference, Lyon, France, 23–27 April 2018; pp. 187–196.
14. Gulenko, A.; Schmidt, F.; Acker, A.; Wallschläger, M.; Kao, O.; Liu, F. Detecting anomalous behavior of black-box services modeled with distance-based online clustering. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 912–915.
15. Liu, P.; Xu, H.; Ouyang, Q.; Jiao, R.; Chen, Z.; Zhang, S.; Yang, J.; Mo, L.; Zeng, J.; Xue, W.; et al. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, Portugal, 12–15 October 2020; pp. 48–58.
16. Pahl, M.O.; Aubet, F.X. All eyes on you: Distributed Multi-Dimensional IoT microservice anomaly detection. In Proceedings of the 2018 14th International Conference on Network and Service Management (CNSM), Rome, Italy, 5–9 November 2018; pp. 72–80.
17. Jin, M.; Lv, A.; Zhu, Y.; Wen, Z.; Zhong, Y.; Zhao, Z.; Wu, J.; Li, H.; He, H.; Chen, F. An anomaly detection algorithm for microservice architecture based on robust principal component analysis. *IEEE Access* **2020**, *8*, 226397–226408. [CrossRef]
18. Bogatinovski, J.; Nedelkoski, S.; Cardoso, J.; Kao, O. Self-supervised anomaly detection from distributed traces. In Proceedings of the 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), Leicester, UK, 7–10 December 2020; pp. 342–347.
19. Nedelkoski, S.; Cardoso, J.; Kao, O. Anomaly detection and classification using distributed tracing and deep learning. In Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Larnaca, Cyprus, 14–17 May 2019; pp. 241–250.
20. Gan, Y.; Zhang, Y.; Hu, K.; Cheng, D.; He, Y.; Pancholi, M.; Delimitrou, C. Leveraging deep learning to improve performance predictability in cloud microservices with seer. *ACM SIGOPS Oper. Syst. Rev.* **2019**, *53*, 34–39. [CrossRef]
21. Zhou, X.; Peng, X.; Xie, T.; Sun, J.; Ji, C.; Liu, D.; Xiang, Q.; He, C. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, 26–30 August 2019; pp. 683–694.
22. Wang, T.; Zhang, W.; Xu, J.; Gu, Z. Workflow-aware automatic fault diagnosis for microservice-based applications with statistics. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 2350–2363. [CrossRef]
23. Salfner, F.; Malek, M. Using hidden semi-Markov models for effective online failure prediction. In Proceedings of the 2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007), Beijing, China, 10–12 October 2007; pp. 161–174.
24. Beschastnikh, I.; Brun, Y.; Ernst, M.D.; Krishnamurthy, A. Inferring models of concurrent systems from logs of their behavior with CSight. In Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 31 May–7 June 2014; pp. 468–479.
25. Magalhaes, J.P.; Silva, L.M. Detection of performance anomalies in web-based applications. In Proceedings of the 2010 Ninth IEEE International Symposium on Network Computing and Applications, Cambridge, MA, USA, 15–17 July 2010; pp. 60–67.
26. Peiris, M.; Hill, J.H.; Thelin, J.; Bykov, S.; Kliot, G.; Konig, C. Pad: Performance anomaly detection in multi-server distributed systems. In Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA, 27 June–2 July 2014; pp. 769–776.
27. Abdelrahman, G.M.; Nasr, M.M. Detection of Performance Anomalies in Cloud Services: A Correlation Analysis Approach. *Int. J. Mech. Eng. Inf. Technol.* **2016**, *4*, 1773–1781. [CrossRef]
28. Wu, L.; Tordsson, J.; Elmroth, E.; Kao, O. Causal Inference Techniques for Microservice Performance Diagnosis: Evaluation and Guiding Recommendations. In Proceedings of the 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), Washington, DC, USA, 27 September–1 October 2021; pp. 21–30.
29. Chen, P.; Qi, Y.; Zheng, P.; Hou, D. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In Proceedings of the IEEE INFOCOM 2014-IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; pp. 1887–1895.
30. Chen, P.; Qi, Y.; Hou, D. Causeinfer: automated end-to-end performance diagnosis with hierarchical causality graph in cloud environment. *IEEE Trans. Serv. Comput.* **2016**, *12*, 214–230. [CrossRef]

31. Lin, J.; Chen, P.; Zheng, Z. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In Proceedings of the International Conference on Service-Oriented Computing, Hangzhou, China, 12–15 November 2018; Springer: Berlin/Heidelberg, Germany, pp. 3–20.
32. Chen, H.; Chen, P.; Yu, G. A framework of virtual war room and matrix sketch-based streaming anomaly detection for microservice systems. *IEEE Access* **2020**, *8*, 43413–43426. [CrossRef]
33. Meng, L.; Ji, F.; Sun, Y.; Wang, T. Detecting anomalies in microservices with execution trace comparison. *Future Gener. Comput. Syst.* **2021**, *116*, 291–301. [CrossRef]
34. Shan, H.; Chen, Y.; Liu, H.; Zhang, Y.; Xiao, X.; He, X.; Li, M.; Ding, W. ?-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In Proceedings of the World Wide Web Conference, San Francisco, CA, USA, 13–17 May 2019; pp. 3215–3222.
35. Zang, X.; Chen, W.; Zou, J.; Zhou, S.; Lisong, H.; Ruigang, L. A fault diagnosis method for microservices based on multi-factor self-adaptive heartbeat detection algorithm. In Proceedings of the 2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2), Beijing, China, 20–22 October 2018, pp. 1–6.
36. Sauvanaud, C.; Kaâniche, M.; Kanoun, K.; Lazri, K.; Silvestre, G.D.S. Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. *J. Syst. Softw.* **2018**, *139*, 84–106. [CrossRef]
37. Liu, D.; Zhao, Y.; Xu, H.; Sun, Y.; Pei, D.; Luo, J.; Jing, X.; Feng, M. Opprentice: Towards practical and automatic anomaly detection through machine learning. In Proceedings of the 2015 Internet Measurement Conference, Tokyo, Japan, 28–30 October 2015; pp. 211–224.
38. Du, Q.; Xie, T.; He, Y. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing, Copenhagen, Denmark, 10–12 October 2018; pp. 560–572.
39. Mariani, L.; Pezzè, M.; Riganelli, O.; Xin, R. Predicting failures in multi-tier distributed systems. *J. Syst. Softw.* **2020**, *161*, 110464. [CrossRef]
40. FudanSELab. TrainTicket: A Microservices-Based Online Ticket Booking System. 2019. Available online: https://github.com/FudanSELab/train-ticket/ (accessed on 4 May 2023).
41. Arnold, A.; Liu, Y.; Abe, N. Temporal causal modeling with graphical granger methods. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, CA, USA, 12–15 August 2007; pp. 66–75.
42. Akkaya, B.; Çolakoğlu, N. Comparison of Multi-Class Classification Algorithms on Early Diagnosis of Heart Diseases. In Proceedings of the ISBIS Young Business and Industrial Statisticians Workshop on Recent Advances in Data Science and Business Analytics, Istanbul, Turkey, 25–28 September 2019.
43. Omar, S.; Ngadi, A.; Jebur, H.H. Machine learning techniques for anomaly detection: An overview. *Int. J. Comput. Appl.* **2013**, *79*, 33–41. [CrossRef]
44. Moghanian, S.; Saravi, F.B.; Javidi, G.; Sheybani, E.O. GOAMLP: Network intrusion detection with multilayer perceptron and grasshopper optimization algorithm. *IEEE Access* **2020**, *8*, 215202–215213. [CrossRef]
45. Rosay, A.; Riou, K.; Carlier, F.; Leroux, P. Multi-layer perceptron for network intrusion detection: From a study on two recent data sets to deployment on automotive processor. *Ann. Telecommun.* **2022**, *77*, 371–394. [CrossRef]
46. Mubarek, A.M.; Adalı, E. Multilayer perceptron neural network technique for fraud detection. In Proceedings of the 2017 International Conference on Computer Science and Engineering (UBMK), Antalya, Turkey, 5–8 October 2017; pp. 383–387.
47. Mishra, M.K.; Dash, R. A comparative study of chebyshev functional link artificial neural network, multi-layer perceptron and decision tree for credit card fraud detection. In Proceedings of the 2014 International Conference on Information Technology, Bhubaneswar, India, 22–24 December 2014; pp. 228–233.
48. Mohapatra, S.K.; Swain, J.K.; Mohanty, M.N. Detection of diabetes using multilayer perceptron. In Proceedings of the International Conference on Intelligent Computing and Applications: Proceedings of ICICA, Sydney, Australia, 8–10 January 2018; pp. 109–116.
49. Serpen, G.; Gao, Z. Complexity analysis of multilayer perceptron neural network embedded into a wireless sensor network. *Procedia Comput. Sci.* **2014**, *36*, 192–197. [CrossRef]
50. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
51. Sridharan, C. *Distributed Systems Observability*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2018.
52. Labs, G. Grafana Observability Survey 2023. Available online: https://grafana.com/observability-survey-2023/ (accessed on 4 May 2023).
53. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
54. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef]
55. Bishop, C.M. *Neural Networks for Pattern Recognition*; Oxford University Press: Oxford, UK, 1995.
56. Teoh, T.; Chiew, G.; Franco, E.J.; Ng, P.; Benjamin, M.; Goh, Y. Anomaly detection in cyber security attacks on networks using MLP deep learning. In Proceedings of the 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE), Selangor, Malaysia, 11–12 July 2018; pp. 1–5.
57. Adnan, J.; Daud, N.G.N.; Ishak, M.T.; Rizman, Z.I.; Rahman, M.I.A. Tansig activation function (of MLP network) for cardiac abnormality detection. In *AIP Conference Proceedings*; AIP Publishing LLC: Melville, NY, USA, 2018; Volume 1930, p. 020006.

58. Lu, S.; Wei, X.; Li, Y.; Wang, L. Detecting anomaly in big data system logs using convolutional neural network. In Proceedings of the 2018 IEEE 16th International Conference on Dependable, Autonomic and Secure Computing, 16th International Conference on Pervasive Intelligence and Computing, 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), Athens, Greece, 12–15 August 2018; pp. 151–158.
59. Nikravesh, A.Y.; Ajila, S.A.; Lung, C.H.; Ding, W. Mobile network traffic prediction using MLP, MLPWD, and SVM. In Proceedings of the 2016 IEEE International Congress on Big Data (BigData Congress), San Francisco, CA, USA, 27 June–2 July 2016; pp. 402–409.
60. Oliveira, T.P.; Barbar, J.S.; Soares, A.S. Computer network traffic prediction: a comparison between traditional and deep learning neural networks. *Int. J. Big Data Intell.* **2016**, *3*, 28–37. [CrossRef]
61. Zhai, X.; Ali, A.A.S.; Amira, A.; Bensaali, F. MLP neural network based gas classification system on Zynq SoC. *IEEE Access* **2016**, *4*, 8138–8146. [CrossRef]
62. Orrù, P.F.; Zoccheddu, A.; Sassu, L.; Mattia, C.; Cozza, R.; Arena, S. Machine learning approach using MLP and SVM algorithms for the fault prediction of a centrifugal pump in the oil and gas industry. *Sustainability* **2020**, *12*, 4776. [CrossRef]
63. Scikit-Learn. MinMaxScaler. 2023. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (accessed on 4 May 2023).
64. Fei, N.; Gao, Y.; Lu, Z.; Xiang, T. Z-score normalization, hubness, and few-shot learning. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada, 10–17 October 2021; pp. 142–151.
65. Xu, S.; Liu, H.; Duan, L.; Wu, W. An improved LOF outlier detection algorithm. In Proceedings of the 2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 28–30 June 2021; pp. 113–117.
66. Brownlee, J. How to Grid Search Hyperparameters for Deep Learning Models in Python with Keras. *Machine Learning Mastery*. Available online: https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/ (accessed on 1 July 2023).
67. Gonzalez-Cuautle, D.; Hernandez-Suarez, A.; Sanchez-Perez, G.; Toscano-Medina, L.K.; Portillo-Portillo, J.; Olivares-Mercado, J.; Perez-Meana, H.M.; Sandoval-Orozco, A.L. Synthetic minority oversampling technique for optimizing classification tasks in botnet and intrusion-detection-system datasets. *Appl. Sci.* **2020**, *10*, 794. [CrossRef]
68. Brochu, E.; Cora, V.M.; De Freitas, N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv* **2010**, arXiv:1012.2599.
69. Agrawal, S.; Agrawal, J. Survey on anomaly detection using data mining techniques. *Procedia Comput. Sci.* **2015**, *60*, 708–713. [CrossRef]
70. Primartha, R.; Tama, B.A. Anomaly detection using random forest: A performance revisited. In Proceedings of the 2017 International Conference on Data and Software Engineering (ICoDSE), Palembang, Indonesia, 1–2 November 2017; pp. 1–6.
71. Fronza, I.; Sillitti, A.; Succi, G.; Terho, M.; Vlasenko, J. Failure prediction based on log files using random indexing and support vector machines. *J. Syst. Softw.* **2013**, *86*, 2–11. [CrossRef]
72. Eltanbouly, S.; Bashendy, M.; AlNaimi, N.; Chkirbene, Z.; Erbad, A. Machine learning techniques for network anomaly detection: A survey. In Proceedings of the 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), Doha, Qatar, 2–5 February 2020; pp. 156–162.