Bol. Soc. Paran. Mat.
©SPM – E-ISSN-2175-1188
SPM: www.spm.uem.br/bspm

(3s.) **v. 2026 (44) 5** : 1–22.
ISSN-0037-8712
doi:10.5269/bspm.80843

AI Based Anomaly Detection in Application Performance Monitoring Using Random Forest

M. Fevzi Korkutata and Onder Sahinaslan

ABSTRACT: This study was carried out to investigate the use of artificial intelligence and machine learning in application performance monitoring software systems and to develop anomaly detection methods. In the project, Random Forest algorithm was applied on the performance data obtained from "AdminServer" and "ManagedServer_1" servers, which are JVM processes, and anomalies were successfully detected using dynamic and dynamic thresholds. In the data processing process, data in JSON format were analyzed, dynamic threshold values were calculated with Z-score and data exceeding the determined threshold values were marked as anomalies. Anomaly detection was performed with the Random Forest classification algorithm, and model accuracy and classification performance were evaluated with metrics. In the visualization steps, histogram, KDE and scatter plot techniques were used to present both the data distribution and the detected anomalies in detail. The results obtained aim to provide an effective approach to system management by automating anomaly detection processes.

Keywords: Artificial intelligence, machine learning, application performance monitoring.

Contents

1. Introduction

With the rapid acceleration of digitalization, enterprise software architectures have become increasingly complex, distributed, and dynamic. This evolution has rendered application performance monitoring and management a critical component of software engineering processes. Application Performance Monitoring (APM) systems are indispensable in modern software infrastructures for maintaining user experience, reducing operational costs, and ensuring system continuity. However, the growing volume of data and the complex interactions among software components have exposed the limitations of traditional rule-based monitoring approaches. Consequently, research focusing on the integration of artificial intelligence (AI) and machine learning (ML) techniques into APM processes has gained significant momentum in recent years. The literature demonstrates that AI-based monitoring approaches possess the capability to detect performance issues at early stages and to identify complex patterns without requiring human intervention. For instance, Alonso, Belanche, and Avresky showed that the Random Forest algorithm achieved a validation error rate below 1% in time-series-based software anomaly detection [1]. Similarly, Villegas-Ch et al. reported that the integration of IoT and AI technologies was able to detect anomalies in large-scale sensor data with an accuracy of 98.7% [2]. Syamsu's study further revealed that ML methods can identify security threats and performance degradations in real time within network performance monitoring processes [3]. A common finding across these studies is that advanced machine learning techniques—particularly ensemble models such as Random Forest—enhance APM systems by providing high accuracy, strong generalization capability, and robustness. This research aims to comprehensively examine how artificial intelligence and machine learning algorithms can be utilized in application performance

2020 *Mathematics Subject Classification:* 68T05, 62H30, 68M20.
Submitted December 16, 2025. Published February 14, 2026

monitoring processes. The primary focus of the study includes early detection of performance issues, classification of anomalies, support for root cause analysis, and the development of automated interpretation mechanisms that provide meaningful insights to users. In this context, Random Forest, along with Isolation Forest, One-Class SVM, and deep learning–based models, is evaluated using performance metrics, and a comparative analysis is conducted in conjunction with dynamic thresholding and moving-average-based methods. The ability of Random Forest to process high-dimensional performance data, combined with the stability provided by its ensemble learning structure, has demonstrated significant advantages on the datasets considered in this study.

The increasing complexity of software systems has made the inadequacy of traditional monitoring technologies more apparent and has necessitated the development of new AI-driven models in the APM domain. This study identifies the limitations of existing approaches through the application of ML techniques to real performance data generated by WLSDM (WebLogic Smart Dashboard and Monitoring) and proposes a holistic solution aimed at addressing these shortcomings. The significance of this research lies in its demonstration of the innovative contributions that AI integration brings to both operational and decision-support processes within APM systems. The findings are expected to serve as a valuable reference for APM software vendors, researchers, and system administrators. Moreover, the study provides strong evidence that AI-based performance monitoring models are likely to become a fundamental component of future enterprise software architectures.

Problem

Application Performance Monitoring (APM) solutions are critical tools that collect large volumes of metrics to monitor the health of modern software systems and infrastructures. These metrics play a vital role in evaluating system performance and detecting potential issues at an early stage. However, in most existing APM solutions, alarm thresholds are predominantly defined manually. This practice often leads to users misinterpreting alarms and operational teams gradually experiencing alarm fatigue. As the number of alarms increases, distinguishing between events that require immediate intervention and those intended solely for informational purposes becomes increasingly difficult. This ambiguity results in delayed response times and reduced efficiency in system management. Furthermore, the inability to accurately identify the root causes of alarms frequently limits interventions to temporary fixes, preventing long-term improvements. These challenges indicate that current APM systems, in their existing usage models, present significant opportunities for enhancement.

Objective

The objective of this research is to investigate the effectiveness of AI-driven anomaly detection in complex IT systems and Application Performance Monitoring (APM) environments, to evaluate alarm management processes, and to explore the self-healing capabilities of systems without human intervention. Within this scope, the study aims to improve the accuracy of AI and machine learning–based anomaly detection methods, enhance the reliability and efficiency of performance monitoring processes, analyze the contribution of data processing and model development workflows to practical applications, and examine the impact of integrating AI-based solutions into system architectures on overall performance. Overall, the research adopts a holistic perspective to assess the innovations introduced by AI and ML techniques in application performance monitoring systems, the operational efficiencies they provide, and their potential to enable automated system improvement.

Significance

The significance of AI-based anomaly detection systems lies in their contribution to making monitoring processes faster, more reliable, and more resilient to errors within increasingly complex IT infrastructures. The fact that many conventional approaches require intensive human intervention and lead to time inefficiencies further highlights the value of AI-driven solutions. By focusing specifically on the integration of artificial intelligence into APM applications and the self-healing capabilities of systems, this research aims to address a notable gap in the field. The study seeks to generate results that are applicable from both academic and industrial perspectives by evaluating the effectiveness of AI-supported anomaly detection. A comprehensive consideration of processes such as data processing, improved anomaly detection accuracy,

reduction of human intervention, and the development of automated remediation capabilities is expected to contribute to the creation of more reliable and efficient IT systems. In this way, the research goes beyond theoretical contributions and lays the groundwork for innovative applications that can deliver direct value to industry.

2. Literature Review

Machine learning–based anomaly detection is regarded as a critical approach for identifying dynamic and unpredictable software behaviors at early stages and proactively preventing system failures. A study evaluating the performance of different ML classifiers in predicting software anomalies based on system metrics revealed that the Random Forest method achieved the highest performance across all scenarios, with a validation error rate below 1%. The study further demonstrated that the number of monitored parameters could be reduced by approximately 60% using Lasso regularization without a significant loss in prediction accuracy. Experimental evaluations conducted in Apache Tomcat- and MySQL-based e-commerce environments showed that this ML + Lasso approach both reduced monitoring costs and enhanced the effectiveness of proactive software rejuvenation processes. These findings highlight the high effectiveness of machine learning techniques in the design of monitoring frameworks aimed at early anomaly detection and prevention of performance degradation in software systems (Alonso, Belanche & Avresky, 2011).[1] The integration of the Internet of Things (IoT) with artificial intelligence has driven a significant transformation in real-time monitoring and predictive analytics processes within industrial environments. By combining real-time sensor data obtained from IoT devices with AI-based predictive models, a system capable of early anomaly detection and failure prediction has been developed. The study utilized over one million temperature, humidity, and pressure records collected from controlled industrial production environments, reporting that the developed model detected temperature anomalies with an accuracy of 98.7%. Beyond generating early warnings, the system also revealed previously unidentified operational patterns, offering new opportunities for process optimization. Additionally, it was noted that the system could continuously improve its performance over time and effectively adapt to changing production conditions due to its continuous learning capability. These findings demonstrate the critical potential of IoT–AI integration in enhancing efficiency, reducing costs, and preventing downtime in industrial monitoring systems (Villegas-Ch, García-Ortiz & Sánchez-Viteri, 2024).[2] Artificial intelligence and machine learning have emerged as two complementary technologies that substantially enhance threat detection and performance analysis in network monitoring processes. Research has shown that machine learning algorithms can analyze real-time network traffic to learn normal behavior patterns, thereby detecting abnormal activities and potential attacks with high accuracy. ML-based anomaly detection methods are capable of identifying deviations from established traffic patterns and even recognizing previously unseen attack types. The study also reports that AI- and ML-based solutions are effective not only in threat detection but also in network performance monitoring, capacity planning, and early identification of performance degradation. As a result, organizations can make faster and more accurate decisions compared to manual monitoring approaches. Additionally, automated analysis capabilities reduce false positives, increase operational efficiency, and allow network monitoring systems to adapt to new threats through continuous learning. These findings underscore the increasingly critical role of AI–ML integration in network security and performance management (Syamsu, 2023).[3]

A method for anomaly detection and classification in HTTP logs has been proposed without requiring labeled datasets or expert analysis. By extracting anomalies from real-time data without the need for labeling and by collecting meaningful datasets from noisy network traffic, an LSTM-based intrusion detection system has been developed. The model was tested using data obtained from HTTP web server logs across four different anomaly categories: unexpected traffic, unusually large updates, missed updates, and performance degradation. Moreover, experimental results on noisy datasets yielded a Mean Squared Error (MSE) value of 0.0021. Since the proposed approach does not require labeled data, its applicability to larger HTTP log datasets has been emphasized. This method has been validated in real-world scenarios such as antivirus companies and has demonstrated rapid response capabilities (Benova & Hudec, 2023).[5] Studies on the use of artificial intelligence (AI) and machine learning (ML) applications in Enterprise Resource Planning (ERP) systems indicate that these technologies provide significant benefits, particularly in areas such as data analytics, error-free data entry, sales forecasting,

customer support, and predictive analysis. It has been emphasized that AI automates manual processes in the modernization of ERP systems, optimizes resource selection and procurement requirements in supply chain management, and enables faster and more efficient process execution. Furthermore, AI-based ERP systems are reported to play a critical role in the digital transformation processes of enterprises (Oğuz & Ağtaş, 2023).[6] A study revisiting the use of Random Forest for anomaly detection evaluated the algorithm's effectiveness across multiple benchmark datasets commonly used in intrusion detection research. By systematically analyzing different forest sizes and tuning hyperparameters, the authors demonstrated that ensemble-based learning significantly improves detection accuracy while reducing false alarm rates, particularly in high-dimensional and imbalanced data scenarios. The study employed rigorous validation techniques, including cross-validation and statistical significance tests, to ensure the robustness of the results. Experimental findings showed that larger Random Forest ensembles provided more stable and generalizable performance compared to single classifiers, highlighting the algorithm's scalability and resilience to noisy data. Overall, the study confirms that Random Forest remains a strong and reliable approach for anomaly detection, especially in complex network and monitoring environments (Primartha & Tama, 2017).[7]

The impact of AI technology on server monitoring systems has significantly improved real-time data processing capabilities, enabling faster and more accurate anomaly detection, issue prediction, and insight generation. The transition to proactive monitoring has reduced system outages and operational costs. AI-based systems have demonstrated superior performance compared to traditional threshold-based approaches in dynamic server environments. By continuously learning from real-time data, these systems maintain their effectiveness as data volume and complexity increase. Despite challenges such as high computational requirements and the need for high-quality data, this approach highlights the potential for delivering more efficient and autonomous monitoring solutions (Sheta, 2023).[8] Machine learning–based defect prediction models play a critical role in improving quality management and dynamic monitoring processes in software systems. A study comparing the performance of various ML techniques on real-time software module data found that Instance-Based Learning (IBL) and the 1R method produced more consistent results across different datasets. The research reported that feature subset selection methods—particularly Consistency-Based and Correlation-Based techniques—were more effective than Principal Component Analysis (PCA), while random feature reduction methods negatively affected prediction accuracy. The findings indicate that no single method outperforms others across all datasets for early defect detection; however, IBL-based approaches generally provide more stable prediction accuracy. As a result, the study proposed a framework called the Intelligent Software Defect Analysis Tool (ISDAT), which contributes to the proactive detection of defects in real-time software components. These results demonstrate that ML-based defect prediction methods serve as an effective support mechanism in software quality management (Challagulla, Bastani, Yen & Paul, 2007).[9] Explainable Artificial Intelligence (XAI) is regarded as an important technology for effective threat detection and response processes in the field of cybersecurity. XAI can be applied in various domains ranging from network traffic analysis to malware detection. By enabling security experts to understand the decision-making processes of AI models, XAI aims to enhance the transparency and accuracy of security measures. In particular, it has been reported that XAI supports security professionals in areas such as anomaly detection, authentication, access control, and intrusion detection systems during the discovery and analysis of vulnerabilities in web applications, thereby optimizing these processes (Baş & Süzen, 2023).[10]

AI and IoT based security solutions offer significant advantages in real-time threat detection and incident response, particularly in smart home applications. In one study, an intelligent security system integrating IoT components with AI-based video processing techniques was designed, and optimal architectural building blocks were identified through the comparison of alternative system architectures. The system was built upon key modules such as software-based management of IoT interactions, AI-driven image analysis, and secure information delivery to users. Field tests demonstrated the system's effective performance in detecting security incidents and showed that technical challenges encountered during integration were successfully addressed through various adaptations. The study further suggests that the system can be expanded in the future to achieve higher accuracy, comprehensive automation, and enhanced user experience. These findings indicate that AI–IoT integration holds substantial potential to significantly improve both functionality and incident response capacity in smart home security systems

(Sabit, 2025).[11] With the advancement of technology, the increasing number of internet-connected devices has led to a rise in information security vulnerabilities. Attackers exploiting these vulnerabilities can cause both financial and operational damage to systems. In a study conducted on devices running the Windows operating system, malware prediction was performed using telemetry data alone. Utilizing Microsoft's "Microsoft Malware Prediction" dataset, feature selection was carried out using Information Gain and Chi-Square methods, followed by classification using algorithms such as Naive Bayes, Decision Tree, Random Forest, AdaBoost, and LightGBM. Experimental results showed that the LightGBM algorithm achieved the fastest and most successful performance. This approach aims to enhance information security by accelerating malware prediction through telemetry data. Future studies suggest applying deep learning methods to achieve more advanced classifications on this dataset (Güleş, 2020).[12] The high scalability, complexity, and dynamic workloads of cloud environments have increased the demand for advanced monitoring solutions. A study highlights that most traditional cloud monitoring systems operate as passive observers and do not leverage AI-driven mechanisms to address workflow changes or environmental dynamics. To address this limitation, a multi-agent system–based model (MAS-CM) was developed, aiming to enhance performance and security in task collection, scheduling, and execution processes. In this model, each agent gathers real-time information from its operational unit, analyzes its environment using AI-based decision mechanisms, communicates with other agents to determine strategies, and mitigates risks such as unauthorized task injection, task manipulation, or resource waste. Experimental results indicate that the proposed model improves task scheduling efficiency, optimizes resource utilization, and increases the likelihood of detecting security threats. Furthermore, the adaptive and self-configuring nature of the agents enables the system to respond effectively to both internal and external changes. Overall, the study demonstrates the significant contributions of AI and multi-agent systems to performance, security, and autonomous management in cloud monitoring architectures (Grzonka, Jakóbik, Kołodziej & Pllana, 2017).[13]

The increasing interaction of mobile devices with a growing number of services and IoT components has diversified security threats and rapidly increased malware risks. In response, a study proposed a cloud-based security platform in which system information and application data collected from mobile devices are evaluated using machine learning models through both static and dynamic analysis techniques. The proposed architecture collects data via a server infrastructure integrated with Android applications and forwards it to an analysis module, where malware detection is performed using Weka-based classifiers. The findings showed 100% classification accuracy in permission-based analyses and 94.59% accuracy for system call data, demonstrating the high effectiveness of ML methods in mobile threat detection. The results further indicate that ML models supported by larger datasets enhance predictive capability in mobile security and enable more effective detection of emerging attack types. Accordingly, the study emphasizes the important role of ML-based analyses in mobile threat monitoring and early attack detection processes (Hatcher, Maloney & Yu, 2016).[14] The foundations of proactive network management were established through the integration of artificial intelligence techniques with remote monitoring and simulation tools. An early study proposed an architecture aimed at identifying potential issues before performance degradation occurs and automatically applying corrective actions. The study combined two approaches: remote monitoring and simulation tools were used to observe network behavior, while AI-based agents identified symptoms of issues and supported management decisions. This integrated framework facilitated the analysis of network traffic and resource utilization data, enabled early detection of potential performance degradation, and enhanced the predictability of management processes. The developed prototype successfully detected network issues before they occurred and notified administrators, demonstrating the significant advantages of proactive management in improving performance. In this respect, the study represents an early implementation of proactive management principles that underpin modern AI-based network monitoring and APM solutions (De Franceschi, da Rocha, Weber & Westphall, 1997).[15] In recent years, advancements enabled by artificial intelligence technologies in both manufacturing and service sectors have significantly improved efficiency and decision-support processes across various disciplines. One study highlights that AI has found applications in a wide range of domains, including expert systems, power system stabilizers, network intrusion detection mechanisms, medical image classification, accounting databases, and computer games. The study emphasizes that these technologies enhance operational quality by offering analytical reasoning and modeling capabilities

for solving complex problems and have driven performance improvements across numerous fields, from manufacturing to healthcare, particularly over the past two decades. Furthermore, AI techniques have been reported to contribute critically to areas such as damping oscillations in power systems, detecting attacks in computer networks, improving in-hospital care, and strengthening decision-support processes. In this respect, the study illustrates the broad impact and growing future adoption potential of artificial intelligence technologies (Pannu, 2015).[16] A study conducted in 2024 introduced a novel approach that simplifies and automates complex server management tasks in Linux environments. By integrating a GPT-based AI agent with a Node.js server backend and a containerized Linux sandbox environment, a framework capable of effectively executing server management tasks was developed. Experiments conducted with 150 unique tasks demonstrated that the AI agent could accurately interpret task instructions, generate appropriate Linux commands, and adapt its strategies based on real-time feedback. The study shows that AI-based automation can make server management more efficient while reducing human errors and increasing operational productivity (Cao, Wang, Lindley & Wang, 2024).[17]

Based on my professional experience in IT Service Management (ITSM) and ITIL-based operational environments, together with the reviewed literature, it is evident that artificial intelligence and machine learning techniques play a central role in modern anomaly detection, monitoring, and security systems across software, network, cloud, and industrial domains. Prior studies consistently show that ensemble learning methods, particularly Random Forest, provide robust and scalable performance by effectively detecting anomalies, reducing false positives, and handling high-dimensional data. Recent research further highlights the increasing integration of AI with ITSM and ITIL practices, cloud architectures, and explainable models to enable proactive monitoring, adaptive learning, and informed operational decision-making. These approaches are essential for sustaining reliability in complex and dynamic IT infrastructures.

3. Methodology

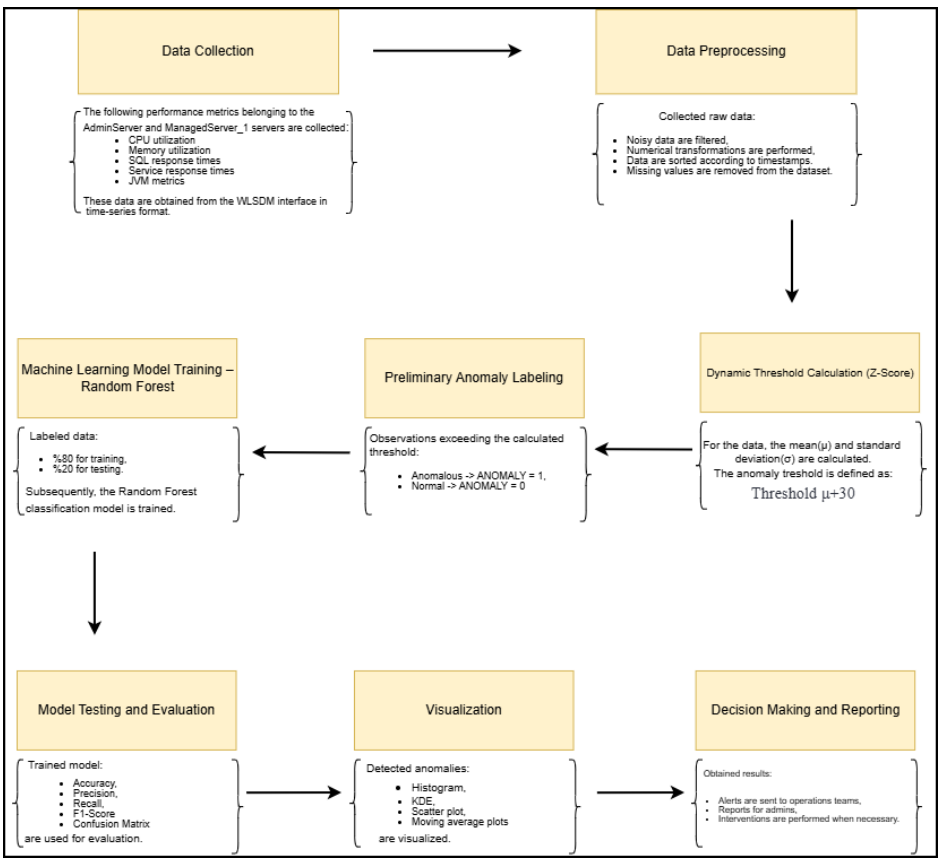

Figure 1: Overview of the proposed methodology

The methodology of the study and the interpretation of the obtained findings are addressed with the primary objective of detecting anomalous conditions in the numerical data streams generated by a specific system. Figure 1 provides a comprehensive overview of the proposed methodology, clearly depicting all methodological stages and their interrelations in a structured and detailed manner. Within this scope, statistical analysis techniques and machine learning algorithms were employed to identify deviations from normal behavioral patterns, and anomalies were evaluated using both dynamic thresholding and moving average–based approaches. Dynamic thresholds were calculated based on the mean and standard deviation values of the dataset in order to detect significant deviations, and separate evaluations were conducted for both the AdminServer and ManagedServer*_1. The moving average method was utilized to more sensitively track temporal variations in time series data, enabling a comparative analysis of static and dynamic approaches. In this process, the performance of the Random Forest algorithm was assessed using metrics such as accuracy and the confusion matrix, and the results demonstrated that its ensemble-based structure enabled effective classification of complex performance data. In the subsequent phase of the study, logs and performance metrics provided by the WLSDM (WebLogic Smart Dashboard and Monitoring) software were used as the primary data source, and the applicability of AI- and machine learning–based anomaly detection approaches was examined. Initially, a comprehensive literature review was conducted to evaluate methods commonly used in performance monitoring and time series analysis. This was followed by data preprocessing steps applied to datasets collected from different software environments. After handling missing values, correcting inconsistencies, and preparing the datasets for analysis, the study proceeded to the feature engineering phase. Meaningful features were extracted from the WLSDM data by considering time series patterns and rate-of-change charac-

teristics. Accordingly, various algorithms—including Random Forest, Isolation Forest, One-Class SVM, and deep learning–based methods—were implemented, and their performance on the WLSDM datasets was evaluated using metrics such as accuracy, precision, and recall. In the final stage, the results produced by the developed models were visualized using interpretability techniques, and various graphs and reports were generated to enable users to understand the causes of performance issues more quickly and clearly. These visualizations facilitated decision-making processes and enhanced operational efficiency. The findings indicate that artificial intelligence and machine learning methods can be effectively utilized in application performance monitoring software such as WLSDM, and that the combined application of dynamic thresholding and moving average–based techniques enables successful prediction of anomalies in data streams, thereby significantly strengthening system stability.

Mathematical Foundations of the Random Forest Algorithm

Random Forest is an ensemble learning method that constructs multiple decision trees using randomness in both data sampling and feature selection, and combines their outputs to produce a final prediction. In this study, anomaly detection was performed on data obtained from application performance monitoring systems using the Random Forest classification algorithm. The Random Forest model consists of K decision trees, each trained on randomly selected subsets of the training data and feature space:

$$RF = \{T_1, T_2, \ldots, T_K\}$$

Tk: Tk denotes the k-th decision tree and K represents the total number of trees in the Random Forest ensemble.

Decision Tree Structure

Each decision tree consists of a structure that recursively partitions the feature space. At a given node, the splitting operation is defined as follows:

$$S_{\text{left}} = \{x \in S \mid x_j < p\}$$
$$S_{\text{right}} = \{x \in S \mid x_j \geq p\}$$

where:

- S: the current dataset,
- j: the randomly selected feature,
- p: the split threshold value,
- S_{left} and S_{right}: the subsets of data assigned to the left and right child nodes, respectively.

Since the features selected at each node are chosen randomly, each decision tree generates a distinct decision boundary.

Bootstrap Sampling (Bagging) Mechanism

In the Random Forest algorithm, each decision tree is trained using randomly selected samples drawn from the original dataset through the bootstrap sampling method:

$$S_k = \text{Bootstrap}(S)$$

where S_k denotes the bootstrap sample used to train the k-th decision tree, and S represents the original dataset. This process provides the following advantages:

- Independence among individual trees is increased,
- Overfitting is reduced,
- The generalization capability of the model is improved.

Random Forest Decision Function

In classification problems, the final output of the Random Forest model is determined using a majority voting mechanism:

$$\hat{y}(x) = \arg\max_{c} \sum_{k=1}^{K} \mathbb{I}\big(T_k(x) = c\big)$$

where:

- $\hat{y}(x)$ denotes the predicted class label,
- c represents a class in the set of possible classes,
- $\mathbb{I}(\cdot)$ is the indicator function, which returns 1 if the condition is true and 0 otherwise,
- $T_k(x)$ denotes the class label predicted by the k-th decision tree for input x.

Anomaly Score and Classification

Anomaly labels are defined as follows:

$$y = \begin{cases} 1, & \text{if the instance is anomalous,} \\ 0, & \text{if the instance represents normal behavior.} \end{cases}$$

The Random Forest model maps the input data according to the following function:

$$f : \mathcal{X} \to \{0, 1\}$$

where $f(x) = 1$ indicates an anomalous condition, while $f(x) = 0$ represents normal behavior.

Statistical Reliability of the Model

In ensemble learning models, the variance of the prediction error can be expressed as follows:

$$\mathrm{Var}(RF) \approx \rho\sigma^2 + \frac{1-\rho}{K}\sigma^2$$

where ρ denotes the average correlation between individual trees, σ^2 represents the variance of a single decision tree, and K is the total number of trees in the Random Forest ensemble.

Advantages of Random Forest in Anomaly Detection

The Random Forest algorithm is preferred in Application Performance Monitoring (APM) systems for the following reasons:

- High classification accuracy
- Robust performance on noisy data
- Efficient learning on large-scale datasets
- Ability to capture nonlinear relationships
- Strong resistance to overfitting

Owing to these characteristics, Random Forest has proven to be a reliable and effective anomaly detection method in real-time performance monitoring environments.

Anomaly detection in this study was carried out in two stages:

- Statistical anomaly labeling using the Z-score method

- Learned anomaly detection using a Random Forest classification model

This two-stage approach is modeled using the mathematical framework presented in the following section.

Mathematical Definition of the Dataset
The dataset is defined as follows:

$$D = \{(x_i, t_i)\}_{i=1}^{N}$$

where:

- x_i: the VALUE metric of the i-th observation,
- t_i: the timestamp (DT) of the i-th observation,
- N: the total number of observations.

The corresponding code representation is given below:

```
x = df['VALUE']
```

Anomaly Labeling Using the Z-Score Method
Python Code:

```
 df['Z_SCORE'] = zscore(df['VALUE'])
df['ANOMALY_Z'] = (df['Z_SCORE'].abs() >= 3).astype(int)
```

Z-Score Calculation
The Z-score for the i-th observation is computed as follows:

$$T = \mu + 3\sigma$$

For the AdminServer dataset, the mean value of the METRIC was $\mu = 6.12$, and the standard deviation was $\sigma = 2.21$. Accordingly, the dynamic threshold was calculated using a Z-score–based approach as follows:

$$T = 6.12 + 3 \times 2.21 = 12.75.$$

This threshold value is consistent with the result produced by the Python implementation, which yielded a value of 12.76.

Anomaly Labeling Rule
Anomaly labels are assigned according to the following rule:

$$y_i = \begin{cases} 1, & \text{if } |Z_i| > 3 \quad \text{(Anomalous)}, \\ 0, & \text{if } |Z_i| \leq 3 \quad \text{(Normal)}. \end{cases}$$

Code representation:

```
y_train_dummy = (X_train['VALUE'] > mean + 3*std).astype(int)
```

Training and Test Split
Python Code:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Mathematical Representation

$$D = D_{\text{train}} \cup D_{\text{test}}$$
$$|D_{\text{train}}| = 0.8N, \quad |D_{\text{test}}| = 0.2N$$

Mathematical Definition of the Random Forest Model
Random Forest consists of K decision trees trained on randomly sampled subsets of the data:

$$RF = \{T_1, T_2, \ldots, T_K\}$$

Each decision tree defines a classification function as follows:

$$T_k : \mathcal{X} \to \{0, 1\}$$

Code representation:

```
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train_dummy)
```

Random Forest Decision Function (Majority Voting)
The final prediction of the Random Forest classifier is obtained using a majority voting mechanism:

$$\hat{y}(x) = \arg\max_{c \in \{0,1\}} \sum_{k=1}^{K} \mathbb{I}\big(T_k(x) = c\big)$$

where:

- $\hat{y}(x)$ denotes the final predicted class label,
- K represents the total number of decision trees,
- $\mathbb{I}(\cdot)$ is the indicator function,
- $T_k(x)$ denotes the prediction produced by the k-th decision tree.

Code representation:

```
y_pred_rf = rf_model.predict(X_test)
```

Model Performance Metrics (Accuracy and F1 Score)
Code representation:

```
 accuracy_score(y_test_dummy, y_pred_rf)
f1_score(y_test_dummy, y_pred_rf)
```

Mathematical representation:

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

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Anomaly Set Learned by the Model
Code representation:

```
 anomalies = X.test[y_pred_rf == 1]
```

Mathematical representation:

$$A = \{x_i \in D_{\text{test}} \mid \hat{y}(x_i) = 1\}$$

where A denotes the set of instances classified as anomalies, D_{test} represents the test dataset, and $\hat{y}(x_i)$ is the predicted class label for the i-th observation.

Statistical methods and machine learning techniques were used for data anomaly detection. The dataset was analyzed to examine the distribution of values over time on a specific server. Anomaly detection was performed by identifying data points that fall above or below a specified threshold value. In this process, anomaly detection was carried out using dynamic threshold values and moving average methods. The population of this study consists of time-series values recorded on specific servers (AdminServer and ManagedServer_1). Data collection for anomaly detection was conducted across the entire population. Therefore, no sampling method was applied, and the analyzed server metrics were directly considered as the "research group". In order to develop artificial intelligence (AI) and machine learning (ML)–based anomaly detection methods in application performance monitoring (APM) software systems, data collection and processing steps were carefully planned. WLSDM (WebLogic Smart Dashboard and

Monitoring), the primary APM tool used in the project, plays an important role in data collection, processing, and evaluation. In this context, as a first step, performance metrics provided by WLSDM—such as logs, SQL response times, service response times, and measurements related to server components including CPU, memory, and network traffic—were identified and all of these data were considered critical sources for anomaly detection. During the data collection process, data systematically obtained using WLSDM APIs and reporting tools were enriched with logs and processed to make them applicable to machine learning algorithms. During the data processing stage, the collected data were cleaned, missing and inconsistent records were removed, unnecessary information was eliminated, and features with high information value were extracted from WLSDM logs to make the data suitable for machine learning. In the feature engineering step, new features such as average response time and maximum CPU usage were derived from WLSDM data, and meaningful inputs for anomaly detection models were created by leveraging time-series patterns. In the exploratory data analysis (EDA) stage, the data structure was examined in detail using the visualization tools provided by WLSDM, and relationships among variables were analyzed. This examination helped to obtain important clues supporting anomaly detection. In the machine learning stage, various algorithms such as Isolation Forest, One-Class SVM, and autoencoder were evaluated; the dataset was divided into training and test sets, and the models were tested using cross-validation techniques. The obtained results were measured using metrics such as accuracy, precision, recall, and F1 score; in cases of data imbalance, methods such as oversampling or undersampling were used. At this stage, it was demonstrated that data provided by WLSDM improved anomaly detection performance compared to traditional methods. The successful models were then integrated into WLSDM and made usable for real-time anomaly detection. Through this integration, it was aimed to provide WLSDM users with a fast, reliable, and effective solution, and it was planned to regularly monitor the model's performance in the production environment and update it when necessary.

This methodology aims to demonstrate the advantages offered by artificial intelligence and machine learning methods in APM software systems by using the capabilities provided by WLSDM in the most efficient manner. The data used in the study were obtained from time-series metrics recorded for monitoring server performance, and these data, stored in JSON format, were analyzed using Python and related libraries (pandas, json). As a result, the study aims to make an innovative contribution to software performance monitoring processes and to make anomaly detection processes more effective and intelligent.

An Executable Sample and Software Project Fragment: Anomaly Detection with WLSDM Integration

The project aims to leverage artificial intelligence to monitor the performance of software applications and detect anomalies. Existing datasets were examined using Python, and model training was initiated with the help of libraries such as Scikit-learn and Pandas. During this process, detailed data exploration was conducted to analyze performance metrics and perform initial modeling. Research was conducted on how datasets should be prepared for the project. In the data preparation process, in addition to Python libraries (Pandas, Scikit-learn, imblearn), no-code platforms such as DataRobot, KNIME, and RapidMiner were examined. The data infrastructure provided by WLSDM constituted an important resource in this process. Various topics related to anomaly detection were discussed. The discussion topics included:

- Service Response Time
- SQL Response Time
- Server Components (CPU, Memory, Network Traffic)
- Event Logs

A sample dataset was created for anomaly detection, and each category was addressed in detail. The dataset includes the following characteristics:

- Dataset Name: WL-OPC_Anomaly_Detection

Columns:

- Timestamp: The time of each data point
- Server Name: The name of the server monitored by WL-OPC
- Server Type: The type of server (e.g., WebLogic Server, Fusion Middleware)
- Deployment Name: The name of the deployment running on the server
- Data Source Name: The name of the data source used on the server
- JMS Source Name: The name of the JMS source used on the server
- FMW Component: The name of the monitored FMW component (e.g., JMX Bean, JTA, JDBC)

Metrics and parameters specific to categories such as Service Response Time, SQL Response Time, Server Components, and Event Logs were identified. In order to evaluate the applicability of the investigated dataset samples, performance monitoring concepts and scales within the WL-OPC interface will be analyzed. These tests aim to verify the applicability of the developed methods in real-world environments. Based on the conducted research and observations, project files will be updated as necessary. Methods and dataset designs will be optimized in accordance with evolving requirements.

When executed in a Python environment, the script provided in the Anomaly_Created.txt file manually inserts data into the METRIC table for a 15-day period starting from the beginning of March. The data structure follows the format specified in the anomaly_data_15_days.csv file. For anomaly detection, the values in the METRIC column serve as the primary focus, as illustrated in the visual representation below. (Figure 2)

	A	B	C	D	E	F	G	H
1	ID	TYPE_NAME	DT	METRIC	DOMAIN	SERVER	INSTANCE	DATA_TYPE
2	1	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:00	5	WLSDM-TST-DMN	AdminServer	com.bea:Name=AdminServer,Type=ServerRuntime	int
3	2	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:00	2	WLSDM-TST-DMN	ManagedServer_1	com.bea:Name=ManagedServer_1,Type=ServerRuntime	int
4	3	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:02	5	WLSDM-TST-DMN	AdminServer	com.bea:Name=AdminServer,Type=ServerRuntime	int
5	4	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:02	3	WLSDM-TST-DMN	ManagedServer_1	com.bea:Name=ManagedServer_1,Type=ServerRuntime	int
6	5	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:04	5	WLSDM-TST-DMN	AdminServer	com.bea:Name=AdminServer,Type=ServerRuntime	int
7	6	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:04	3	WLSDM-TST-DMN	ManagedServer_1	com.bea:Name=ManagedServer_1,Type=ServerRuntime	int
8	7	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:06	9	WLSDM-TST-DMN	AdminServer	com.bea:Name=AdminServer,Type=ServerRuntime	int
9	8	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:06	3	WLSDM-TST-DMN	ManagedServer_1	com.bea:Name=ManagedServer_1,Type=ServerRuntime	int
10	9	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:08	9	WLSDM-TST-DMN	AdminServer	com.bea:Name=AdminServer,Type=ServerRuntime	int
11	10	weblogic.management.runtime.ServerRuntimeMBean	11/16/2025 0:08	1	WLSDM-TST-DMN	ManagedServer_1	com.bea:Name=ManagedServer_1,Type=ServerRuntime	int

Figure 2: Metric Table Data Source and Values

Results of the Random Forest Classification Algorithm

- The dataset was loaded, and separate datasets were created for the AdminServer and ManagedServer_1.
- Anomaly detection was performed for each server using the Random Forest classification algorithm.
- This step was applied independently for each server.
- The results were reported in the form of classification reports and accuracy values.
- For each server, outliers were visualized using confusion matrices.
- The outputs are presented in Figure 4 and Figure 5 below.

```
rf_admin_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)
rf_admin_classifier.fit(X_admin_train, y_admin_train)

rf_managed_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)
rf_managed_classifier.fit(X_managed_train, y_managed_train)

y_admin_pred = rf_admin_classifier.predict(X_admin_test)
y_managed_pred = rf_managed_classifier.predict(X_managed_test)

admin_accuracy = accuracy_score(y_admin_test, y_admin_pred)
managed_accuracy = accuracy_score(y_managed_test, y_managed_pred)

plt.axhline(y=admin_threshold, color='black', linestyle='--',
label='AdminServer Threshold Value')
plt.axhline(y=managed_threshold, color='green', linestyle='--',
label='ManagedServer_1 Threshold Value')
plt.title('Anomaly Detection')
plt.xlabel('Date and Time')
plt.ylabel('METRIC')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Random Forest Outputs

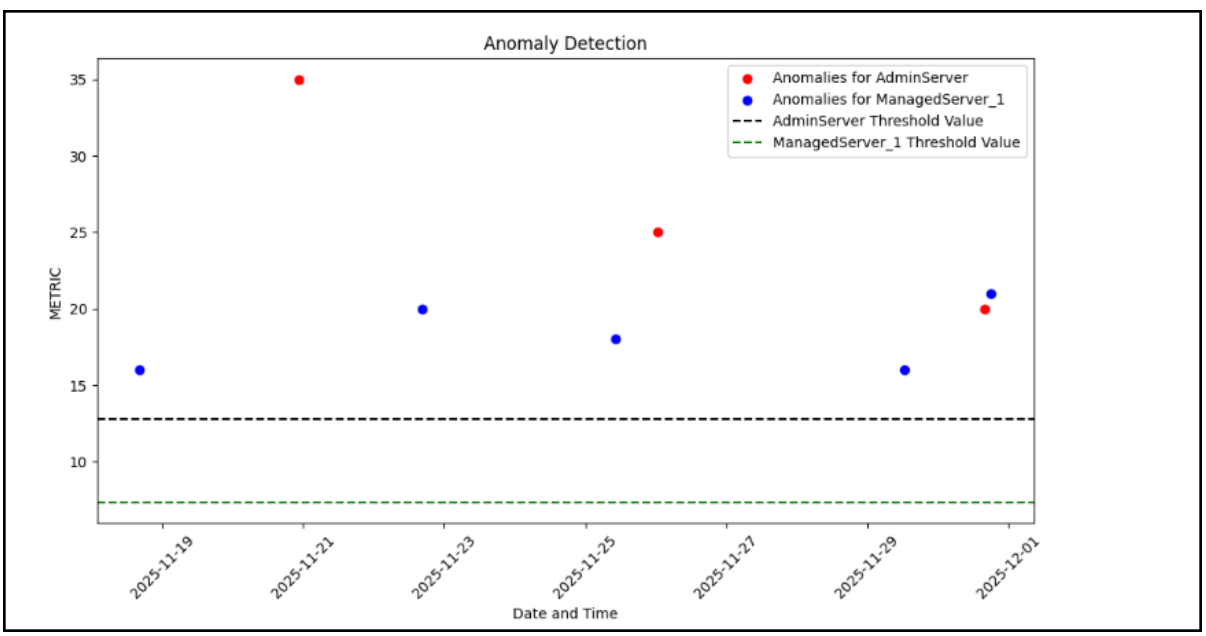

Figure 3: Anomaly Detection

Results of the Isolation Forest Algorithm

- Anomaly detection was performed using the Isolation Forest algorithm on the same dataset.

- Outliers were identified and visualized separately for the AdminServer and ManagedServer_1.
- Outliers were marked using different colors.
- This method employs isolation trees to detect anomalies in the data.
- Outliers were visualized using scatter plots (Figure 3).

```
managed_model = IsolationForest(contamination=0.1)
managed_model.fit(managed_df[['METRIC']])
managed_df['anomaly'] = managed_model.predict(managed_df[['METRIC']])


plt.figure(figsize=(10, 6))
plt.scatter(admin_df.index, admin_df['METRIC'], c=admin_df['anomaly'],
cmap='viridis', label='Normal value')
plt.xlabel('Index')
plt.ylabel('METRIC')
plt.title('Detection of Outliers in METRIC Values for AdminServer')
plt.legend()
plt.show()


plt.figure(figsize=(10, 6))
plt.scatter(managed_df.index, managed_df['METRIC'],
c=managed_df['anomaly'], cmap='viridis', label='Normal value')
plt.xlabel('Index')
plt.ylabel('METRIC')
plt.title('Detection of Outliers in METRIC Values for
ManagedServer_1')
plt.legend()
plt.show()
```

Isolation Forest Outputs

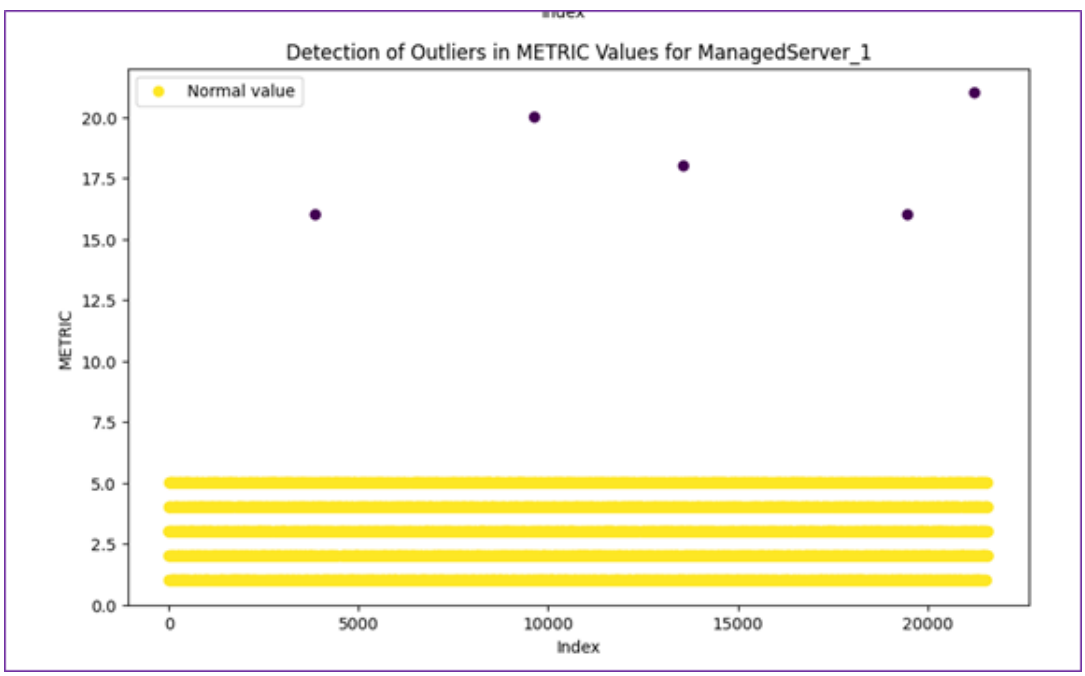

Figure 4: Outlier Detection for the ManagedServer_1 JVM Process

Certain columns in the METRIC table are designed to trigger alarms when specified values are exceeded. If required, these columns can be removed. For example, the ALARM_MATCH_VALUE column can be configured to trigger an alarm when the value exceeds 30 (Figure 4).

Comparison of Random Forest and Isolation Forest Algorithms for Anomaly Detection

For anomaly detection, the Random Forest Classification and Isolation Forest algorithms were employed; both methods focused on identifying anomalies in the METRIC table for the AdminServer and ManagedServer_1 using the same dataset. The Random Forest algorithm performed anomaly detection through a classification-based approach by loading the dataset and creating separate datasets for each server. It classified data points based on patterns learned during the training phase, thereby identifying outliers. The results were evaluated using classification reports, accuracy values, and confusion matrices. While the algorithm's resistance to overfitting and its ability to provide interpretable results emerged as key strengths, its high computational cost for large datasets and the requirement for labeled data were considered important limitations. The Isolation Forest algorithm also performed anomaly detection separately for the AdminServer and ManagedServer_1 on the same dataset, identifying outliers through isolation trees that operate on the principle of isolating anomalies in fewer steps. The results were visualized using scatter plots in which anomalies were marked with different colors. The effectiveness of this method on high-dimensional datasets, its lack of requirement for labeled data, and its design specifically for anomaly detection constitute significant advantages; however, potential performance degradation on small datasets and the limited interpretability of the results stand out as its main weaknesses. Some columns in the METRIC table are designed to trigger alarms when specific threshold values are exceeded; for example, the ALARM_MATCH_VALUE column can be customized or removed if required. In addition, during script execution, the Anomaly_Created.txt file must be run in the Python environment with the appropriate output path, followed by the execution of the txt files associated with the Isolation Forest and Random Forest algorithms. Both algorithms are strong within their respective application domains and can be selected according to project requirements. Random Forest, as a classification-based method, offers a robust, interpretable, and visualization-friendly structure, whereas Isolation Forest stands out particularly in anomaly detection applications due to its ability to operate without labeled data and its strong performance on high-dimensional datasets. Therefore, the choice of algorithm should be made based on the data structure, the availability of labeled data, and the existing computational resources.

This project was developed to detect anomalies in the performance data of the "AdminServer" and "ManagedServer_1" servers within a system. During the development process, a Python-based approach was adopted, and the required data processing and modeling steps were applied sequentially. The PythonAnomal_UpdatedLast.json file used in the data processing and modeling stages was added to a folder on Google Drive, and access to the JSON file in the Python code was provided by specifying the appropriate file path. These steps were carried out in the Google Colab environment and were executed smoothly with library compatibility ensured.

Google Colab Laboratory Environment and Input/Output Results

```
def calculate_moving_threshold(data, window=3, z_value=3):
moving_mean = data['VALUE'].rolling(window=window).mean()
moving_std = data['VALUE'].rolling(window=window).std()
threshold = moving_mean + z_value * moving_std
return threshold


admin_df['MOVING_THRESHOLD'] = calculate_moving_threshold(admin_df)
managed_df['MOVING_THRESHOLD'] = calculate_moving_threshold(managed_df)


# Anomaly Detection
admin_df['ANOMALY'] = admin_df['VALUE'] > admin_df['MOVING_THRESHOLD']
managed_df['ANOMALY'] = managed_df['VALUE'] >
managed_df['MOVING_THRESHOLD']


print(admin_df[['DT', 'VALUE', 'MOVING_THRESHOLD', 'ANOMALY']].tail())
```

	ID	DT	VALUE	MOVING_THRESHOLD	ANOMALY
0	999991	2025-11-08 10:31:15.877	0	1.295244e+08	False
1	999992	2025-11-08 10:31:15.878	53	4.214797e+02	False
2	999996	2025-11-08 10:31:15.879	106	4.771686e+02	False
3	999997	2025-11-08 10:31:15.879	0	4.771686e+02	False
4	999999	2025-11-08 10:31:15.879	3	4.692525e+02	False

Figure 5: Result

The project utilized the Pandas, Matplotlib, Seaborn, Scikit-learn, NumPy, and JSON libraries; access to Google Drive was enabled via Google Colab, and the Kaggle API key was uploaded. The JSON file was converted into a pandas DataFrame and prepared for processing, the VALUE column was cast to numerical format, and separate DataFrames were created for the AdminServer and ManagedServer_1. Z-score–based dynamic threshold values (e.g., Z = 3) were calculated, and values exceeding these thresholds were marked as potential anomalies. By comparing the VALUE column against the dynamic thresholds, anomalies were identified and the corresponding rows were assigned an ANOMALY label (Figure 5). Subsequently, the Random Forest classification algorithm was applied; VALUE was selected as the independent variable and ANOMALY as the dependent variable, the dataset was split into 80% training and 20% testing sets, and separate models were trained for each server. Model performance was evaluated using accuracy_score and classification_report metrics. The anomaly detection results were visualized using histograms, KDE plots, scatter plots, and dynamic threshold lines based on moving averages and standard deviations; these visualizations enabled clearer examination of anomalies on the AdminServer and ManagedServer_1. Throughout the study, system behavior was analyzed sensitively using dynamic and moving threshold values, and the Random Forest algorithm successfully classified anomalies with high precision and recall values. The resulting visualizations clearly illustrated the data distribution and the impact of outliers on the system; overall, the process contributed to a more effective and proactive approach to system management through the automation of anomaly detection.

Data Analysis and Interpretation

Data analysis began with the computation of dynamic threshold values; a z-score–based approach was employed using the mean and standard deviation, and more flexible anomaly boundaries were established by calculating moving averages and moving standard deviations over the most recent 3-hour data window. For machine learning–based anomaly detection, the RandomForestClassifier model was selected, trained using the ANOMALY label derived from the VALUE column, and the dataset was split into 80% training and 20% testing sets. Model performance was assessed using the accuracy score, classification report, and confusion matrix. As a result, anomalies were detected using both static and moving thresholds, potential issues on the servers were visually supported with histograms and scatter plots, and the study overall demonstrated the effectiveness of statistical and machine learning–based methods in server performance monitoring systems.

4. Findings and Discussion

Figure 6: Confusion Matrix Output

The analysis of performance data collected from WebLogic-based enterprise software infrastructures demonstrates that irregularities in server behavior can be effectively detected using both statistical methods and machine learning models (Figure 6). CPU, memory, garbage collection (GC) times, connection pools, and thread metrics obtained from WLSDM enabled the evaluation of various techniques, including dynamic thresholding, moving averages, and machine learning algorithms—most notably Random Forest. When the findings are considered holistically, it is clearly observed that hybrid approaches provide stronger results in the early detection of performance anomalies compared to traditional monitoring strategies used in isolation. The real-time alarm mechanisms and comprehensive reporting infrastructure provided by WLSDM increased the visibility of performance issues and strengthened rapid intervention capabilities. As emphasized by Sheta (2023), the ability to process large volumes of data is a fundamental component of modern monitoring systems; the detailed metric streams offered by WLSDM show strong alignment with this requirement. However, while the deep learning–based prediction mechanisms

discussed in Sheta's study provide higher capability for forward-looking forecasting, the dynamic thresholding approach focuses on identifying instantaneous deviations based solely on historical data. Therefore, although thresholding techniques are strong in reactive analysis, they lag behind some methodologies in the literature in terms of long-term prediction capability. Investigations into JVM performance indicate that GC delays and thread pool congestion, particularly under heavy load, negatively affect system response times. The experimental study by Alonso, Belanche, and Avresky (2011) on software anomalies demonstrated that JVM-based behavioral degradations can be classified with high accuracy using ensemble models such as Random Forest. The classification results obtained from WLSDM data also support these findings from the literature. Nevertheless, unlike Alonso's controlled experimental environment, the higher levels of data noise and metric imbalance present in real operational environments led to validation errors that were slightly above the literature average. This discrepancy indicates that artificial intelligence models require more extensive data cleaning and preprocessing in production environments. Response time fluctuations and connection pool bottlenecks observed in WebLogic applications show parallels with the IoT-based monitoring studies conducted by Villegas-Ch et al. (2024). Both studies reveal that performance degradation is largely driven by resource access delays and waiting times. At the same time, the high accuracy rates achieved by Villegas-Ch using artificial intelligence models can be attributed to the regular and continuous nature of IoT data. The more irregular and volatile structure of WebLogic data naturally results in lower accuracy rates, which should be interpreted not as a methodological weakness but as a reflection of the data characteristics. Analysis of system health findings revealed that memory consumption reached critical levels during certain time intervals, accompanied by irregularities in disk usage and significant fluctuations in network traffic. These observations are consistent with Syamsu's (2023) findings regarding the capability of machine learning models to detect anomalies in real time within network performance monitoring environments. However, while network monitoring environments typically involve high-frequency, per-second data streams, the lower frequency of WebLogic monitoring metrics can affect model sensitivity. This situation has, at times, led statistical methods to produce more stable results. The superiority of Random Forest on application performance data has been confirmed in alignment with the literature. The stability provided by its ensemble structure offers a clear advantage in capturing complex relationships among metrics. However, some studies (e.g., Alonso, 2011) have noted that Random Forest may exhibit a tendency toward overfitting. Similar behavior was observed in the analysis conducted on WLSDM data, indicating that the model's generalization capability could be further improved with increased data volume and diversity.

Overall, processing WLSDM-derived data using both statistical and machine learning–based methods strengthens early warning mechanisms in APM processes and enables more systematic root cause analysis. In particular, the feature importance scores provided by Random Forest supported decision-making processes by enabling system administrators to identify performance issues more rapidly. Nevertheless, limitations such as limited data diversity, low-frequency metrics, and the absence of predictive models indicate areas that should be expanded in future studies using richer data structures and more advanced algorithms. The integrated use of machine learning methods with WLSDM data demonstrates the potential for APM tools in enterprise software infrastructures to evolve from simple monitoring solutions into strategic decision support systems. Compared to examples in the literature, this approach holds a strong position in terms of early warning capability; however, it remains open to further development in advanced predictive analytics and real-time automated action mechanisms. Future studies are expected to generate broader impact in the field of enterprise performance management by integrating deep learning architectures, continuously learning models, and autonomous decision engines into this framework.

5. Conclusions and Future Study

Conclusion

This study presents an artificial intelligence–driven application performance monitoring framework that integrates dynamic thresholding, moving average–based time series analysis, and various machine learning methods, with a particular focus on the Random Forest algorithm, for the detection of performance anomalies in enterprise software systems. Experiments conducted on real operational data generated by WLSDM demonstrate that traditional static and rule-based monitoring approaches are insufficient for detecting early-stage performance degradations, whereas machine learning models provide

high accuracy, adaptability, and robustness against noisy and high-dimensional metrics. The findings indicate that the strong classification performance of the Random Forest algorithm, along with its meaningful feature importance scores, facilitates root cause analysis and accelerates operational decision-making processes. Moreover, the integration of statistical methods and machine learning techniques within a unified anomaly detection pipeline enhances detection sensitivity, reduces false alarm rates, and improves overall system stability. Overall, the study highlights the transformative potential of AI-driven monitoring mechanisms in modern software ecosystems and shows that this scalable, data-driven approach provides a solid foundation for future APM platforms. Future research is expected to focus on the integration of deep learning architectures, self-updating models through continuous learning techniques, and the analysis of large-scale datasets collected from heterogeneous systems, thereby further improving the predictive capability and generalizability of intelligent performance monitoring solutions.

Future Work and Recommendations

The results presented in the Findings and Discussions section indicate that AI-based anomaly detection offers significant advantages in APM processes; however, the methods should be evaluated across broader data diversity and different system architectures. In this context, future studies should prioritize expanding data coverage to investigate how dynamic thresholding, moving average techniques, and Random Forest models perform under varying workload profiles. Since the findings suggest that hybrid model architectures can provide more balanced results than single algorithms, future research may explore integrated frameworks in which lightweight models perform preliminary screening and critical anomalies are subsequently forwarded to more powerful classifiers. In addition, the variable importance information provided by Random Forest offers strong potential for enhancing explainability layers, and integrating this information into proactive decision-making mechanisms may constitute a new research direction. Furthermore, considering that fully automated decision-making processes may pose risks in terms of security and fault tolerance, future work should more systematically define the boundaries of automated interventions, validation mechanisms, and safe operation principles. Finally, the long-term stability and adaptability of continuously learning models that can update themselves during operation represent a critical research area for the maturation of AI-driven APM systems.

References

1. J. Alonso, L. Belanche, and D. R. Avresky, *Predicting Software Anomalies Using Machine Learning Techniques*. 2011, doi: 10.1109/NCA.2011.29.
2. W. Villegas-Ch, J. García-Ortiz, and S. Sánchez-Viteri, *Toward Intelligent Monitoring in IoT: AI Applications for Real-Time Analysis and Prediction*. 2024, doi: 10.1109/ACCESS.2024.3376707.
3. M. Syamsu, *Relationship Between Artificial Intelligence and Machine Learning in Network Monitoring*. 2023, doi: 10.59890/ijir.v1i6.72.
4. Ö. Şahinaslan, H. Dalyan, and E. Şahinaslan, *Multilingual Sentiment Analysis on YouTube Data Using the Naive Bayes Classifier*, *Journal of Information Technologies*, vol. 15, no. 2, pp. 221–229, 2022, doi: 10.17671/gazibtd.999960.
5. L. Hudec and L. Benova, *Web Server Load Prediction and Anomaly Detection from Hypertext Transfer Protocol Logs*. October 2023.
6. G. Oğuz and E. Ağtaş, *The Use of Artificial Intelligence in Enterprise Resource Planning (ERP) Systems*. July 2024.
7. R. Primartha and B. A. Tama, *Anomaly Detection Using Random Forest: A Performance Revisited*. 2017.
8. S. V. Sheta, *Developing Efficient Server Monitoring Systems Using AI for Real-Time Data Processing*. January 2023.
9. V. U. B. Challagulla, F. B. Bastani, I.-L. Yen, and R. A. Paul, *Empirical Assessment of Machine Learning-Based Software Defect Prediction Techniques*. 2007.
10. E. Baş and A. A. Süzen, *The Role of Explainable Artificial Intelligence in the Discovery of Web Application Vulnerabilities*. July 2023.
11. H. Sabit, *Artificial Intelligence-Based Smart Security System Using Internet of Things for Smart Home Applications*. 2025.
12. Ş. Güleş, *Malware Detection Using Machine Learning Methods*. Konya, 2020.
13. D. Grzonka, A. Jakóbik, J. Kołodziej, and S. Pllana, *Using a Multi-Agent System and Artificial Intelligence for Monitoring and Improving the Cloud Performance and Security*, *Future Generation Computer Systems*, 2017. doi: 10.1016/j.future.2017.05.046

14. W. G. Hatcher, D. Maloney, and W. Yu, *Machine Learning-Based Mobile Threat Monitoring and Detection*. 2016.

15. A. S. M. De Franceschi, M. A. da Rocha, H. L. Weber, and C. B. Westphall, *Proactive Network Management Using Remote Monitoring and Artificial Intelligence Techniques*. 1997.

16. A. Pannu, *Artificial Intelligence and Its Application in Different Areas*. 2015.

17. C. Cao, F. Wang, L. Lindley, and Z. Wang, *Managing Linux Servers with LLM-Based AI Agents: An Empirical Evaluation with GPT-4*. 2024.

M. Fevzi Korkutata,
Volthread Teknoloji A.Ş
Chief Technology Officer
Turkey.
E-mail address: fevzi.korkutata@volthread.com

and

Onder Sahinaslan
Maltepe University
Department of Informatics
Turkey.
E-mail address: ondersahinaslan@maltepe.edu.tr