НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
сентябрь–октябрь 2025 Том 25 № 5 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
September–October 2025 Vol. 25 No 5 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)

НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

doi: 10.17586/2226-1494-2025-25-5-910-922

Enhanced detection of denial-of-service attacks in Kubernetes: a multi-framework machine learning approach integrating node and application metrics

Ghadeer Darwesh[1], Jaafar Hammoud[2], Alisa A. Vorobeva[3]✉

[1,2,3] ITMO University, Saint Petersburg, 197101, Russian Federation

[1] ghadeerdarwesh32@gmail.com, https://orcid.org/0000-0003-1116-9410
[2] hammoudgj@gmail.com, https://orcid.org/0000-0002-2033-0838
[3] vorobeva@itmo.ru✉, https://orcid.org/0000-0001-6691-6167

Abstract

The widespread adoption of Kubernetes as a platform for orchestrating containerized applications has heightened the need for effective security mechanisms, particularly to counter Denial-of-Service (DoS) attacks. This article proposes an approach to DoS attack detection based on two key components the use of comprehensive metrics and the application of ensemble Machine Learning models. The approach involves the collection and analysis of comprehensive metrics from node-level (CPU, memory) and application-level (network activity, file descriptors) data from containers running on various frameworks (Flask, Django, FastAPI, Node.js, Golang). To implement this approach, a dataset containing 49,990 instances of network activity, characterized by 28 features (comprehensive metrics), was created. Statistical analysis (Student's t-test, Pearson correlation) identified the metrics most relevant for attack detection, including total CPU time (cpu_sec_total) and resident memory usage (resident_memory_total). A comparison of nine Machine Learning models for attack detection was conducted, including ensemble methods (Random Forest, XGBoost, LightGBM) which demonstrated the highest effectiveness, achieving 100 % accuracy (F1-score equals 1.0) and perfect class separation (AUC equals 1.0). The XGBoost model also eliminated false positives (precision equals 1.0). Feature importance analysis revealed the most significant metrics for classification: CPU usage (cpu_sec_total, cpu_sec_idle), network packet transmission (transmit_packets), system load average, and memory usage (virtual_memory_total, resident_memory_total). The work emphasizes the importance of integrating multi-level metrics for building resilient anomaly detection systems. The proposed approach is scalable and independent of specific frameworks, making it applicable for protecting containerized environments. The research results serve as a foundation for developing proactive Kubernetes security systems capable of countering sophisticated attack vectors.

Keywords

Kubernetes, DoS attack detection, machine learning, node-level metrics, application-level metrics, anomaly detection, ensemble models

For citation: Darwesh G., Hammoud J., Vorobeva A.A. Enhanced detection of denial-of-service attacks in Kubernetes: a multi-framework machine learning approach integrating node and application metric. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2025, vol. 25, no. 5, pp. 910–922. doi: 10.17586/2226-1494-2025-25-5-910-922

УДК 004.056

Повышение эффективности обнаружения DoS-атак в Kubernetes: подход на основе машинного обучения с интеграцией метрик уровня узлов и приложений для мультифреймворковых сред

Гадир Дарвиш[1], Жаафар Хаммуд[2], Алиса Андреевна Воробьева[3]✉

[1,2,3] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация

[1] ghadeerdarwesh32@gmail.com, https://orcid.org/0000-0003-1116-9410
[2] hammoudgj@gmail.com, https://orcid.org/0000-0002-2033-0838
[3] vorobeva@itmo.ru✉, https://orcid.org/0000-0001-6691-6167

Аннотация
Широкое распространение Kubernetes как платформы для оркестрации контейнеризированных приложений атакам типа «Отказ в обслуживании» (Denial-of-Service, DoS). В работе предложен подход к обнаружению DoS-атак, основанный на двух ключевых компонентах: использование комплексных метрик и применение ансамблевых моделей машинного обучения. Подход предполагает сбор и анализ комплексных метрик: уровня узлов (Central Processing Unit (CPU), память) и уровня приложений (сетевая активность, файловые дескрипторы) из контейнеров, работающих на различных фреймворках (Flask, Django, FastAPI, Node.js, Golang). Для реализации подхода создан набор данных, содержащий 49 990 экземпляров сетевой активности, охарактеризованных 28 признаками (комплексными метриками). Статистический анализ (t-критерий Стьюдента, корреляция Пирсона) выявил наиболее релевантные для детектирования атак метрики, включая общее время использования CPU (cpu_sec_total) и объем задействованной оперативной памяти (resident_memory_total). Сравнение девяти моделей машинного обучения для детектирования атак, включая ансамблевые методы (Random Forest, XGBoost, LightGBM), показало наивысшую эффективность (F1-мера равна 1,0) и полное разделение классов (AUC равна 1,0). Применение модели XGBoost позволило исключить ложноположительные срабатывания (precision равна 1.0). Анализ важности признаков выявил наиболее значимые для классификации метрики, связанные с использованием CPU (cpu_sec_total, cpu_sec_idle), передачей сетевых пакетов (transmit_packets), средней загрузкой системы и использованием памяти (virtual_memory_total, resident_memory_total). Проведенное исследование показало важность интеграции разноуровневых метрик для создания устойчивых систем обнаружения аномалий. Предложенный подход является масштабируемым и независимым от конкретных фреймворков, что делает его применимым для защиты контейнеризированных сред. Результаты исследования служат основой для разработки проактивных систем безопасности Kubernetes, способных противостоять сложным векторам атак.

Ключевые слова
Kubernetes, обнаружение DoS-атак, машинное обучение, метрики уровня узлов, метрики уровня приложений, обнаружение аномалий, ансамблевые модели

Ссылка для цитирования: Дарвиш Г., Хаммуд Ж., Воробьева А.А. Повышение эффективности обнаружения DoS-атак в Kubernetes: подход на основе машинного обучения с интеграцией метрик уровня узлов и приложений для мультифреймворковых сред // Научно-технический вестник информационных технологий, механики и оптики. 2025. Т. 25, № 5. С. 910–922 (на англ. яз.). doi: 10.17586/2226-1494-2025-25-5-910-922

Introduction

Kubernetes, the de facto standard for container orchestration, has revolutionized cloud-native architectures by automating the deployment, scaling, and management of containerized applications. However, the complexity and dynamic nature of Kubernetes clusters expose them to a wide range of security threats, with Denial-of-Service (DoS) attacks standing out as a prominent risk. These attacks exploit Kubernetes resource scaling mechanisms to inundate cluster resources, potentially leading to service disruptions and substantial financial repercussions [1, 2]. The containerized workloads managed by Kubernetes pose unique challenges in detecting and mitigating DoS attacks. Containers often demonstrate dynamic, ephemeral, and unpredictable resource utilization patterns which blur the line between legitimate traffic bursts and malicious activity. Conventional DoS detection methods, such as static thresholds and signature-based techniques, fall short in such settings due to their inability to adapt to Kubernetes highly dynamic operational environments. These limitations often result in high false-positive rates, rendering traditional approaches unsuitable for production environments [1, 2]. Machine Learning (ML) has emerged as a promising paradigm for tackling these challenges. By leveraging anomaly detection techniques, ML-based systems can dynamically identify deviations in traffic and resource utilization patterns without relying on predefined rules or static thresholds. However, many existing studies focus on specific application frameworks, such as Flask, or narrow workloads, thereby limiting their generalizability across diverse operational contexts [1, 2]. Recent advancements in Kubernetes security emphasize the importance of hybrid approaches that combine ML with runtime monitoring and rule-based mechanisms. Tools such as extended Berkeley Packet Filter (eBPF) and Express Data Path (XDP) offer lightweight, high-performance anomaly detection at the kernel level, enabling real-time security insights. Despite these innovations, significant gaps remain in evaluating the effectiveness of such approaches across diverse frameworks and workloads, leaving room for improvement in generalizability and robustness [3, 4]. Building upon our prior work [5], where we developed an ML-based DoS detection framework tailored to the Flask framework, this study extends the scope to encompass multiple application frameworks, including Django, FastAPI, Flask, Golang, and Node.js. This broader scope addresses the generalizability concerns raised in our earlier work and ensures applicability to a wider range of Kubernetes environments.

This study makes three key contributions: it provides a comparative analysis of ML-based DoS detection across multiple frameworks to improve generalizability and robustness; it integrates lightweight runtime monitoring tools to enhance detection efficiency; and it evaluates advanced classifiers for distinguishing between natural workload variations and attack-induced anomalies in Kubernetes environments [5–7].

Literature Review and Previous Work

DoS attacks are among the most prevalent security threats in containerized environments. Kubernetes, with its dynamic orchestration and auto-scaling capabilities, provides an efficient platform for managing modern workloads but also presents a large attack surface for

adversaries to exploit. DoS attacks in Kubernetes often target resource management mechanisms, overwhelming cluster components like Central Processing Unit (CPU), memory, and network bandwidth to render applications unresponsive [1, 2]. Studies emphasize the difficulty in distinguishing between natural workload variations and attack-induced overloads, as both may manifest as anomalies in resource usage [1, 4].

Conventional DoS detection techniques rely on predefined thresholds or static rules to identify anomalous behaviors. While computationally inexpensive, these methods are rigid and fail to adapt to dynamic environments like Kubernetes [1, 2, 4].

Recent advancements have introduced more sophisticated approaches, such as Host-Based Intrusion Detection Systems (HIDS). Researchers in [8] developed a real-time HIDS for Linux containers that monitors system calls from the host kernel to detect anomalies. Their method achieved a 100 % detection rate with a false positive rate of just 2 %.

Several studies focus on Docker containers, the most widely used container runtime. For example: Researchers in [9] proposed an online anomaly detection system using an optimized isolation forest algorithm. By assigning weights to resource metrics and incorporating weighted feature selection, this approach improved accuracy while maintaining minimal performance overhead, crucial for differentiating between attack-induced and natural resource overloads. In [10], a probabilistic real-time IDS was proposed using *n*-grams of system calls and probabilistic models like Maximum Likelihood Estimator. This system achieved detection accuracy between 87 % and 97 % across datasets. Dynamic approaches using anomaly-based methods have demonstrated significant improvements over static techniques. In [11], researchers evaluated dynamic schemes on 28 real-world container vulnerability exploits, with results indicating that dynamic methods detected 22 out of 28 exploits compared to only three detected by static methods.

ML-based anomaly detection systems offer significant advantages in identifying DoS attacks. Techniques like Random Forest, Gradient Boosting, and Neural Networks have been widely adopted for detecting anomalies in Kubernetes. The introduction of eBPF and XDP has enabled lightweight, kernel-level anomaly detection for real-time insights [1, 2]. Studies such as [12] have demonstrated the potential of ML classifiers like Decision Trees and Random Forests in distinguishing between legitimate and malicious activities at the container level. These approaches achieved F-measures of 99.8 % for attack detection while maintaining low resource overheads.

Most existing research evaluates DoS detection methods on specific frameworks, such as Flask or FastAPI, without accounting for their generalizability to other workloads [1, 2, 4]. Researchers in [13] highlighted the need for cross-framework evaluations by analyzing security mechanisms in Docker containers across multiple deployment scenarios. Their results emphasized that framework-agnostic detection systems are critical for robust Kubernetes security.

Studies are often constrained to single frameworks, making their findings less applicable to diverse Kubernetes workloads [1, 2, 4].

Despite these advancements, the current research landscape reveals several persistent and interconnected limitations that hinder the development of robust, practical detection systems. A primary constraint is the lack of generalizability across technological stacks. The majority of studies evaluate proposed methods within the context of a single application framework, neglecting validation across diverse environments [1, 2, 4, 13]. This significantly limits the applicability of such solutions in real-world Kubernetes clusters which are inherently heterogeneous and host applications built with different languages and frameworks.

Furthermore, a significant efficiency-effectiveness trade-off remains unresolved. While static methods are computationally efficient yet inflexible, the more effective dynamic and ML approaches typically demand large volumes of training data and introduce substantial computational overhead, making them costly to deploy in resource-sensitive environments [10, 14].

Finally, the critical challenge of integration is still largely unaddressed. Only a limited number of studies have explored the combination of real-time monitoring techniques (e.g., eBPF) with ML systems to achieve the dual objectives of high accuracy and low-latency detection in Kubernetes production settings [1].

Thus, the identified gaps — limited generalizability, an unoptimized accuracy-overhead trade-off, and a lack of integrated real-time solutions — form the core research challenge addressed by this work.

This study directly addresses these limitations by proposing a comprehensive detection framework validated across multiple application frameworks and programming languages, including Flask, Django, FastAPI, Golang, and Node.js. Our approach leverages lightweight, eBPF-based runtime monitoring to minimize performance impact and detection latency. We conduct an extensive comparative evaluation of ML classifiers to identify optimal strategies for DoS detection in Kubernetes. Through these contributions, this research provides a foundation for developing scalable, framework-agnostic security solutions capable of protecting complex Kubernetes deployments against evolving DoS threats.

Data and Statistical Study

These contributions aim to provide a framework-agnostic, efficient, and accurate solution to securing Kubernetes environments against evolving threats. The dataset[1] consists of 49,990 instances with 28 features, encompassing both node-level and app-level metrics collected from multiple frameworks deployed in Kubernetes by using a collector developed in [15]. The target variable, *attack*, is binary, indicating the presence (1) or absence (0) of DoS attacks. Table 1 presents the node-level metrics gathered from the frameworks, while Table 2 details the app-level metrics.

We conducted a comprehensive statistical analysis to demonstrate the robustness and reliability of the dataset. This analysis provides insights into the distribution, central

[1] Available at: https://github.com/ghadeerda/Kubernetes-model-agent (accessed: 29.08.2025).

Table 1. This table lists and describes the metrics collected at the node level, including CPU, memory, disk, and network-related features

Column Name	Description
id	Unique identifier for the observation
time	Timestamp of the data record
cpu_sec_idle	Percentage of CPU idle time during the interval
disk_av_per	Available disk space as a percentage
disk_read	Amount of data read from the disk (in bytes)
disk_write	Amount of data written to the disk (in bytes)
net_receive	Network data received (in bytes)
mem_pressure	Memory pressure indicator (value reflects memory load)
mem_av_per	Available memory as a percentage
forks_total	Total number of process forks
intr	Number of interrupts handled by the CPU
load1, load5, load15	CPU load average over 1, 5, and 15 minutes respectively
receive_drop	Number of network packets dropped during reception
receive_errs	Number of network reception errors
transmit_packets	Number of packets transmitted over the network
ipv4_sock_inuse	Number of IPv4 sockets currently in use
est_conn	Number of established connections
lis_conn	Number of listening connections
open_fds	Number of open file descriptors
attack	Indicator for attack presence (1 for attack, 0 for no attack)

Table 2. This table provides details of the application-level metrics, including resource utilization features specific to the application frameworks

Column Name	Description
id	Unique identifier for the observation
time	Timestamp of the data record
cpu_sec_total	Total CPU usage in seconds
virtual_memory_total	Total virtual memory usage (in bytes)
resident_memory_total	Total resident memory usage (in bytes)
open_fds	Number of open file descriptors
attack	Indicator for attack presence (1 for attack, 0 for no attack)

tendencies, and variability of both node-level and app-level metrics, highlighting the dataset ability to capture diverse system behaviors under different conditions. By examining the relationships between features and their correlations with the target variable (*attack*), we confirmed that the dataset effectively encapsulates the characteristics required for detecting DoS attacks. This statistical study not only validates the dataset integrity but also underscores its potential for developing and benchmarking robust anomaly detection models in containerized environments.

To thoroughly understand the dataset characteristics and assess its suitability for detecting DoS attacks, we conducted an in-depth statistical analysis. This analysis aimed to explore key metrics at both node-level and app-level, evaluate their relationships, and identify patterns that distinguish between attack and non-attack states. By combining descriptive statistics, hypothesis testing, and correlation analysis, we established a solid foundation for understanding the dataset structure and its potential for training robust ML models. The following sections detail the findings from these analyses.

Descriptive Statistics

During attacks, CPU usage increases significantly while idle time decreases, indicating elevated system load. Virtual and resident memory also spike, reflecting stress on memory resources. Open file descriptors rise, suggesting heavier application activity. Disk and network I/O metrics show moderate changes but still contribute to overall anomaly detection.

Hypothesis Testing

T-tests revealed statistically significant differences between attack and non-attack states. Features such as

cpu_sec_total, resident_memory_total, and open_fds were highly significant, while disk_read, disk_write, and virtual_memory_total showed moderate significance. These findings confirm that attacks consistently alter resource usage patterns.

Correlation Analysis

Correlation analysis showed that cpu_sec_total, resident_memory_total, and open_fds are positively correlated with attacks, while cpu_sec_idle and disk_av_per are negatively correlated. Some memory-related features showed redundancy, while others like forks_total and disk_av_per contributed unique information.

Key Insights

The most predictive indicators of DoS attacks include increased CPU and memory usage, along with higher counts of open file descriptors. These resource usage patterns correspond to the system stress and resource exhaustion typically induced by attacks. Fig. 1 presents histograms comparing the distributions of key. Fig. 2 shows boxplots of metrics. These visualizations were generated by the authors based on the labeled dataset of 49,990 instances, which includes node- and application-level metrics collected from real Kubernetes workloads using five frameworks (Flask, Django, FastAPI, Node.js, and Golang). The patterns observed in these figures are derived from the statistical analysis discussed earlier (t-tests and correlation), and form the empirical basis for the ML models used in this study.

Sample Representativeness and Realism of Workload Simulation

To ensure a realistic and representative dataset, we designed the data collection process to emulate real

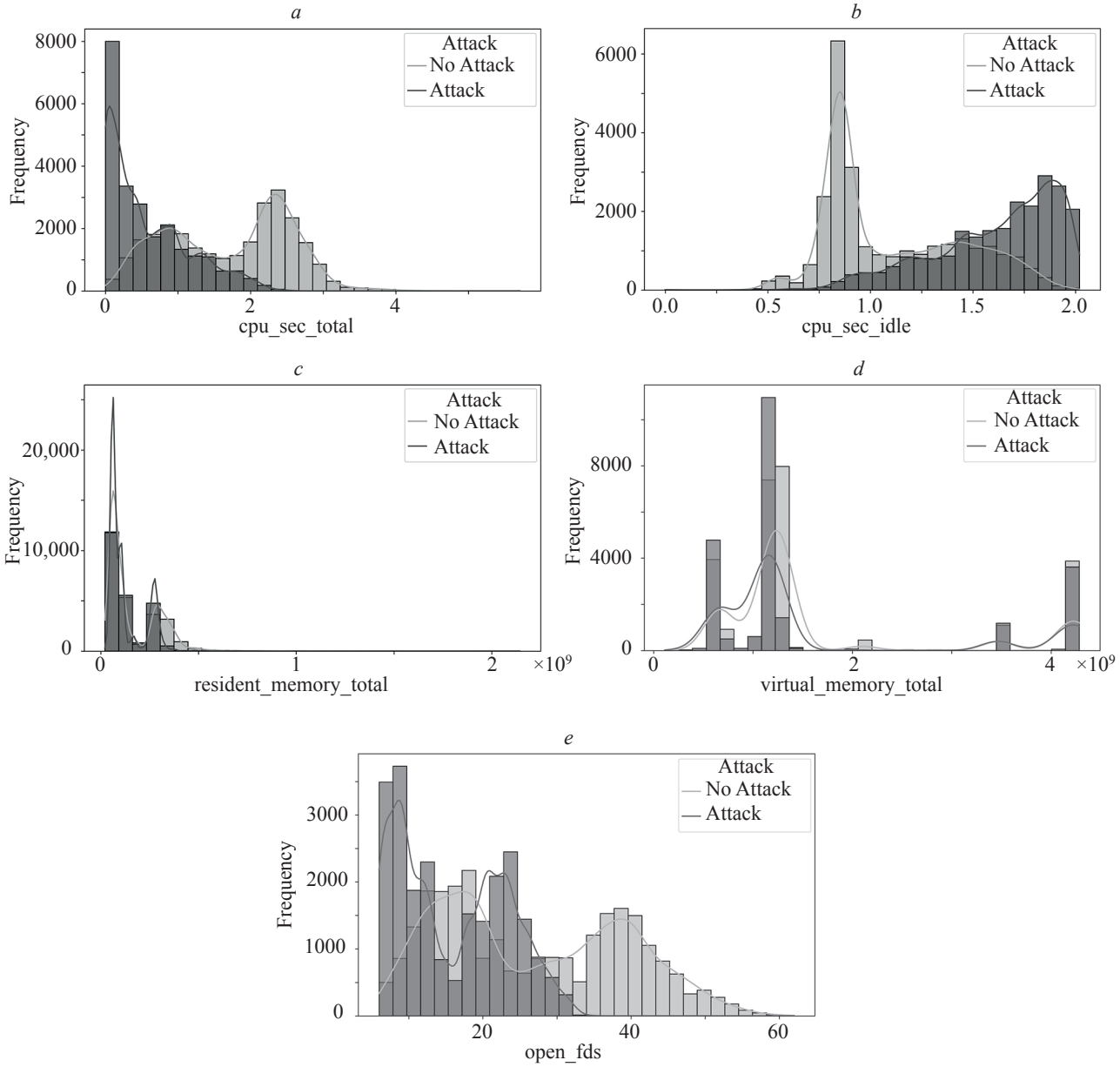

Fig. 1. Histograms of Frequency Distribution of CPU and memory usage metrics (cpu_sec_total, resident_memory_total) under attack and non-attack conditions, based on the collected dataset: cpu_usage_total (*a*); cpu_sec_idle (*b*); resident_memory_total (*c*); virtual_memory_total (*d*); open_fds (*e*)

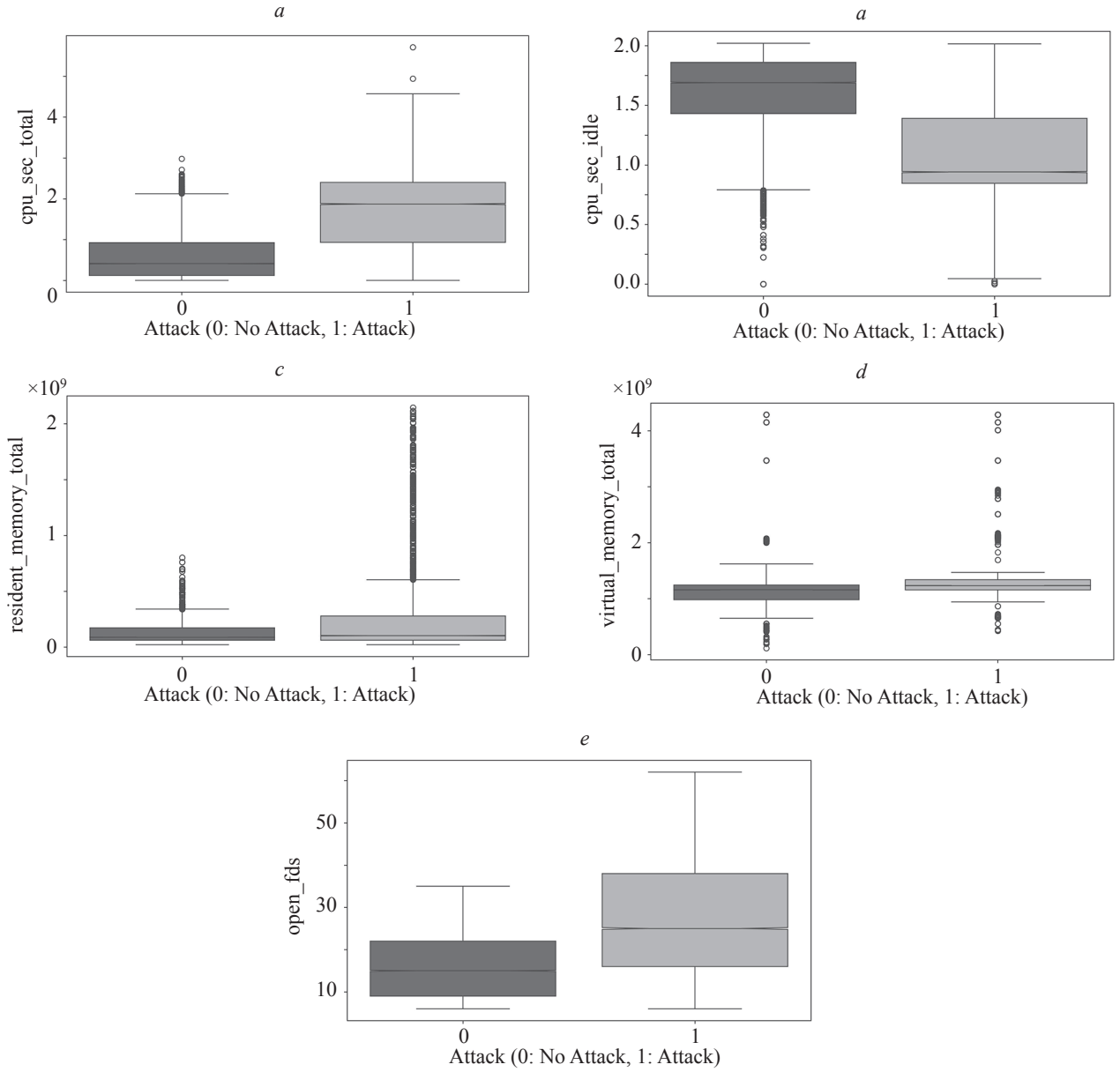

Fig. 2. Distribution shifts for key metrics. Feature Boxplots comparing CPU idle time (cpu_sec_idle) and open file descriptors (open_fds) between normal and attack states in Kubernetes workloads: cpu_usage_total (*a*); cpu_sec_idle (*b*); resident_memory_total (*c*); virtual_memory_total (*d*); open_fds (*e*)

Kubernetes environments. The dataset includes both node- and application-level metrics from applications built with five frameworks — Flask, Django, FastAPI, Golang, and Node.js — covering diverse architectures and performance patterns. Benign traffic was generated using synthetic tools and real user-like behavior, simulating varying request rates, CPU/memory loads, and I/O operations to reflect real-world workload dynamics such as traffic spikes and batch processing. Attack data was produced using standard DoS tools to stress both network and application layers, mimicking real adversarial scenarios like volumetric floods and resource exhaustion. This blend of framework diversity, metric richness, and controlled anomaly injection results in a dataset that reflects real operational conditions and provides a solid foundation for training effective, generalizable ML models.

ML Models and Detection Approach

We evaluated multiple supervised learning algorithms for detecting DoS attacks, focusing on their ability to generalize across different frameworks and languages (Flask, Django, FastAPI, Nodejs, Golang). The classifiers include: Logistic Regression: A simple linear model for baseline comparisons [16]. Random Forest: A tree-based ensemble model known for its robustness to overfitting [17]. Gradient Boosting: An iterative boosting model suitable for handling imbalanced datasets [18]. Support Vector Machine (SVM): Effective for high-dimensional feature spaces [19]. Decision Tree: A fast and interpretable model [20]. Naive Bayes: Suitable for datasets where feature independence can be assumed [21]. K-Nearest Neighbors: A distance-based approach for capturing non-

linear patterns [22]. XGBoost: A high-performance gradient boosting model [23]. LightGBM: A scalable and efficient tree-based model optimized for large datasets [24].

All features were standardized using StandardScaler to ensure uniform scaling across models. Any missing values were imputed based on the mean or median of the respective feature.

A stratified 5-fold cross-validation strategy was employed to ensure robust evaluation across diverse data splits. For tree-based models, feature importance scores were computed to interpret the contribution of individual metrics.

Accuracy, precision, recall, and F1-score were computed for each classifier. Receiver Operating Characteristic (ROC) curves and Area Under the Curve (AUC) scores were used to assess model performance. In addition to the Confusion Matrix which provided a detailed breakdown of true positives, true negatives, false positives, and false negatives for each classifier.

The detection pipeline was implemented using Python, leveraging libraries, such as Scikit-learn, XGBoost, and LightGBM. The training and evaluation processes were automated to facilitate reproducibility and ensure consistent results across multiple frameworks.

Results and Discussion

This section presents a detailed analysis of the results derived from the machine learning classifiers applied to the combined dataset, integrating both application and node-level metrics from multiple frameworks. The discussion includes model performance metrics, feature importance analysis, and insights into classifier behavior for detecting anomalies.

The cross-validation accuracy of the classifiers was evaluated to assess their ability to generalize across different data subsets. The accuracy boxplot shows that ensemble models like Random Forest, Gradient Boosting, XGBoost, and LightGBM consistently outperformed other models, achieving nearly perfect performance with minimal variance. Among all classifiers: Random Forest and Decision Tree achieved perfect classification accuracy with no false predictions. XGBoost achieved an accuracy of 100 % with robust predictive power. Linear classifiers like Logistic Regression and simpler models like Naive Bayes showed relatively lower but acceptable accuracies. The classifier cross-validation accuracy plot highlights the stability of ensemble models compared to others (Fig. 3).

The confusion matrices provide detailed insights into the classification behavior of each ML model. Fig. 4 illustrates these matrices for all evaluated classifiers, showing the distribution of true positives, true negatives, false positives, and false negatives. Notably, XGBoost and Random Forest achieved perfect classification, with zero misclassifications of either attack or non-attack instances. In contrast, models like Naive Bayes and SVM showed some weaknesses. For example, Naive Bayes falsely identified 4,967 normal cases as attacks, reflecting its limitation in environments with overlapping feature distributions. SVM exhibited a slightly elevated false positive rate due to its sensitivity to kernel parameterization. These confusion matrices were generated by the authors using the labeled dataset of 49,990 samples, collected from a realistic Kubernetes cluster and detailed in the dataset description. The results confirm the relative strengths of ensemble models and provide a comparative evaluation of precision and recall across all classifiers.

Feature Importance

Feature importance was analyzed using: tree-based native importance for ensemble models and permutation importance for other classifiers where native importance is unavailable.

The following trends were observed: For tree-based models like Random Forest, LightGBM, and XGBoost, the most critical features were: CPU utilization metrics (cpu_sec_total, cpu_sec_idle), Packet transmission metrics (transmit_packets), System load averages (load1, load5, load15), and Memory metrics (virtual_memory_total, resident_memory_total). XGBoost and Gradient Boosting

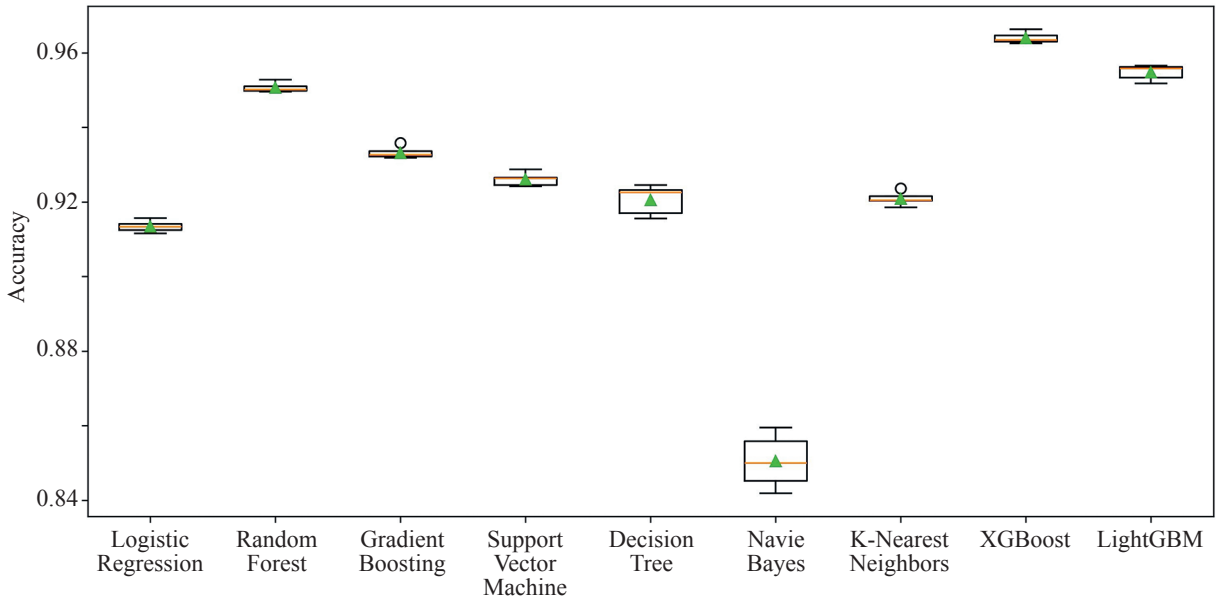

Fig. 3. Boxplot comparing cross-validation accuracy across all evaluated machine learning models

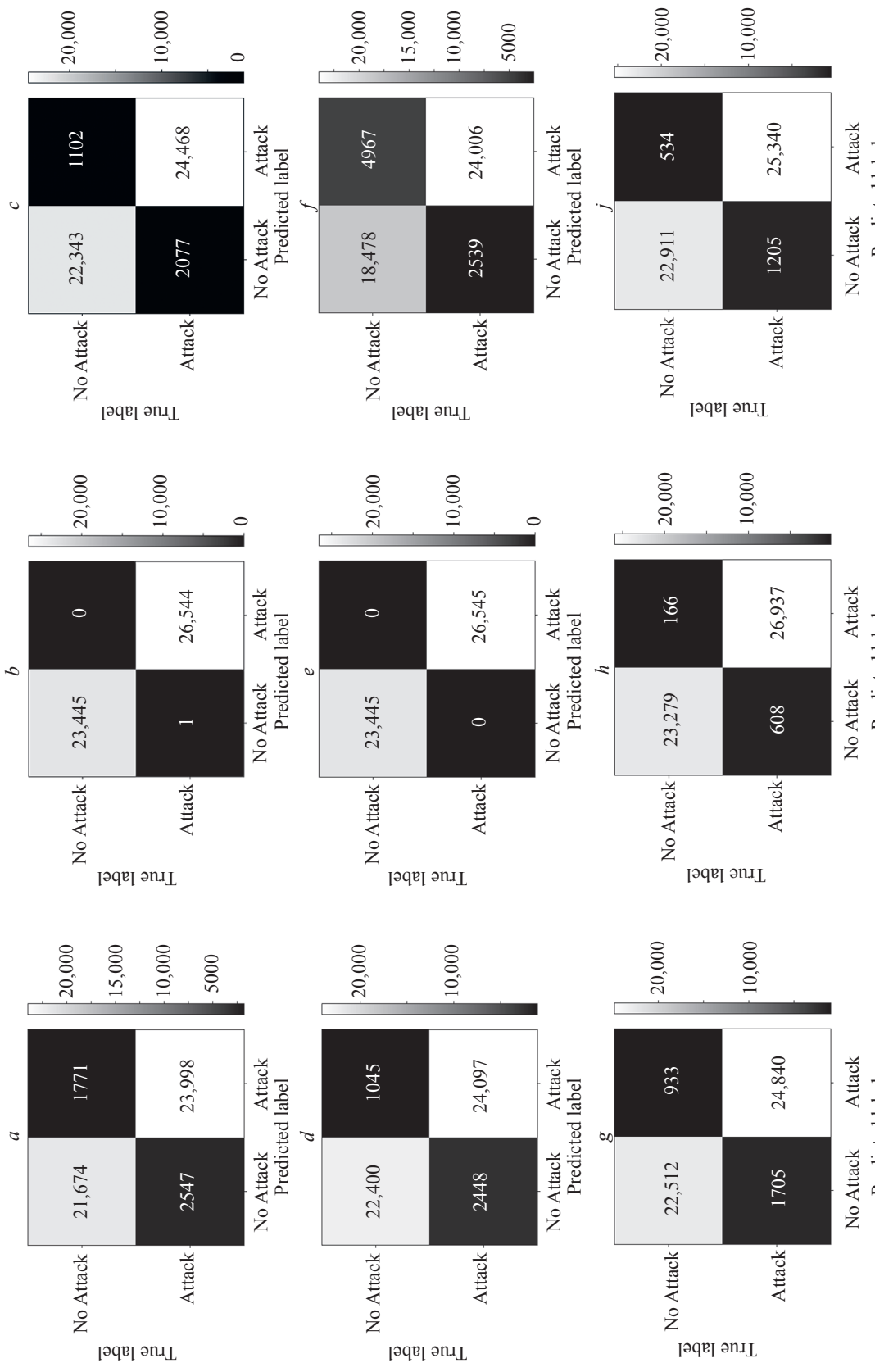

Fig. 4. Individual confusion matrices for each classifier, highlighting true positives, false positives, true negatives, and false negatives: Logistic Regression (*a*) Random Forest (*b*); Gradient Boosting (*c*); Support Vector Machine (*d*); Decision Tree (*e*); Naive Bayes (*f*); K-Nearest Neighbors (*g*); XGBoost (*h*); LigthGBM (*j*)

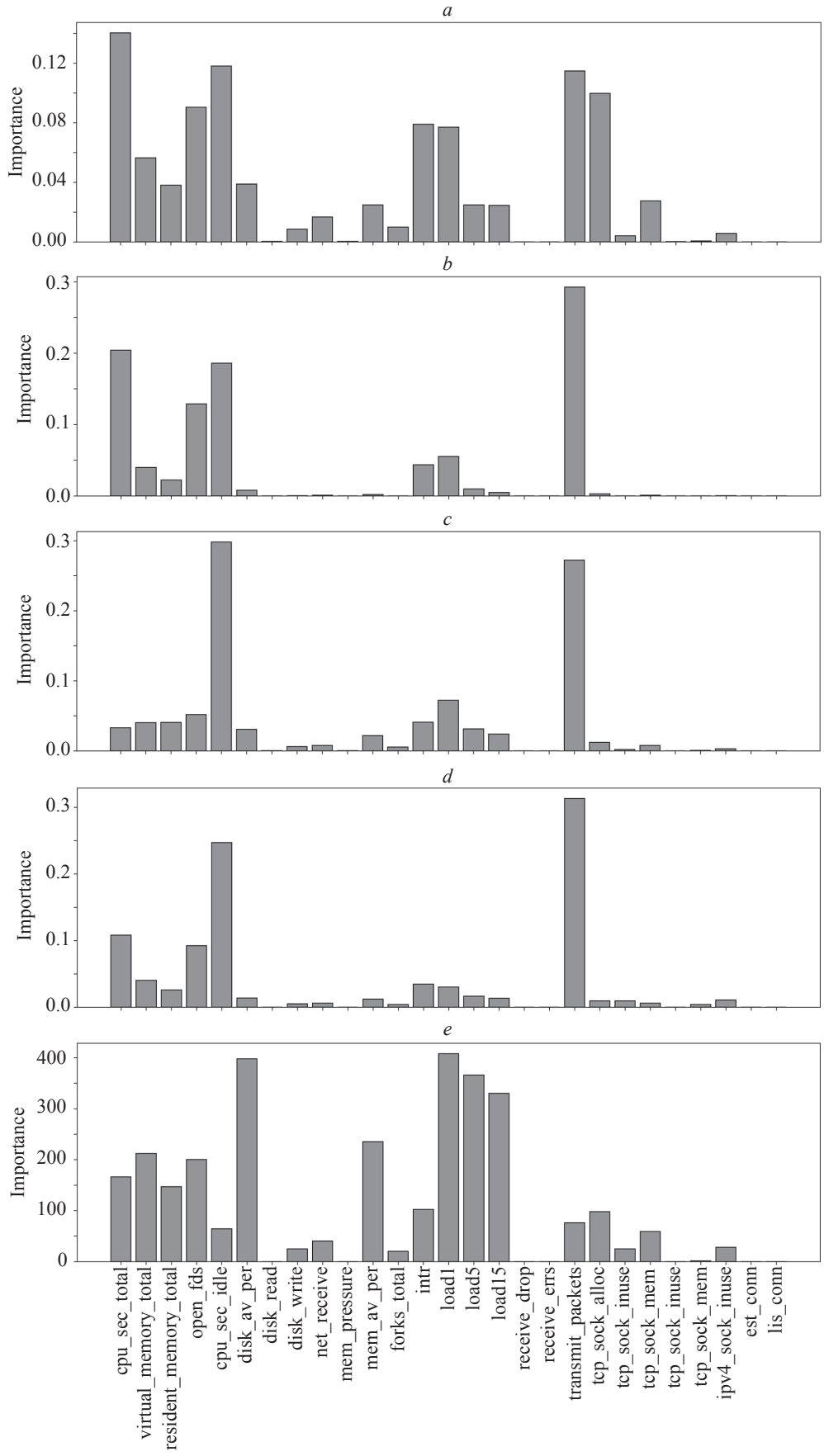

Fig. 5. Bar plots showing feature importance rankings for models that provide native importance measures (e.g., Random Forest, XGBoost, LightGBM): Random Forest (*a*); Gradient Boosting (*b*); Decision Tree (*c*); XGBoost (*d*); LigthGBM (*e*)

highlighted the dominance of network-level metrics such as transmit_packets and tcp_sock_alloc. Decision Tree models underscored similar features, with additional emphasis on resource allocation metrics (open_fds, mem_av_per). The variation across feature importance profiles indicates that different models prioritize features differently, depending on their intrinsic algorithms and data processing mechanics.

Separate plots for feature importances across classifiers and a combined feature importance comparison provide detailed insights (Fig. 5).

Receiver Operating Characteristic Analysis

The ROC curves demonstrated high AUC values for all classifiers: XGBoost, Random Forest, and Decision Tree exhibited an AUC of 1.0, confirming perfect separability of the classes. Models like Naive Bayes and Logistic Regression achieved slightly lower AUCs (about 0.94 and 0.97, respectively), indicating lower sensitivity to certain features. The ROC curves for all classifiers are included for a holistic view of performance across false-positive and true-positive rates (Fig. 6).

Quantitative Evaluation of Early Detection Capabilities

In addition to accuracy, a key strength of a detection system is how early it can identify anomalies before traditional alerts are triggered. We analyzed the time gap between model predictions and system-level resource exhaustion warnings. XGBoost and Random Forest detected DoS attack signatures from 3 to 12 seconds before critical signs like CPU saturation, memory exhaustion, or application failures occurred. This early warning enables proactive actions, such as auto-scaling, traffic throttling, or container isolation — reducing the risk of service disruption. Detection lead time was measured by aligning attack labels with resource usage traces and locating inflection points in key metrics (cpu_sec_total, resident_memory_total, open_fds). Fig. 7 illustrates this timeline, showing that our models consistently flag anomalies before threshold breaches, confirming their effectiveness in real-time Kubernetes defense systems.

Key Insights and Implications

Ensemble models like Random Forest, XGBoost, LightGBM, and Gradient Boosting consistently outperformed other classifiers in accuracy, robustness, and feature prioritization, proving highly effective for high-dimensional, multi-source datasets. In contrast, simpler models such as Naive Bayes provided baseline performance but lacked the sophistication needed for complex environments. Resource and network metrics — including CPU utilization, packet transmission, and system load — were key indicators for anomaly detection. Integrating node- and application-level data significantly improved detection accuracy and scalability, making ensemble approaches well-suited for real-time deployment in Kubernetes and edge environments. These results establish a strong foundation for developing generalizable frameworks that can distinguish between attack-induced and natural workload fluctuations in containerized systems.

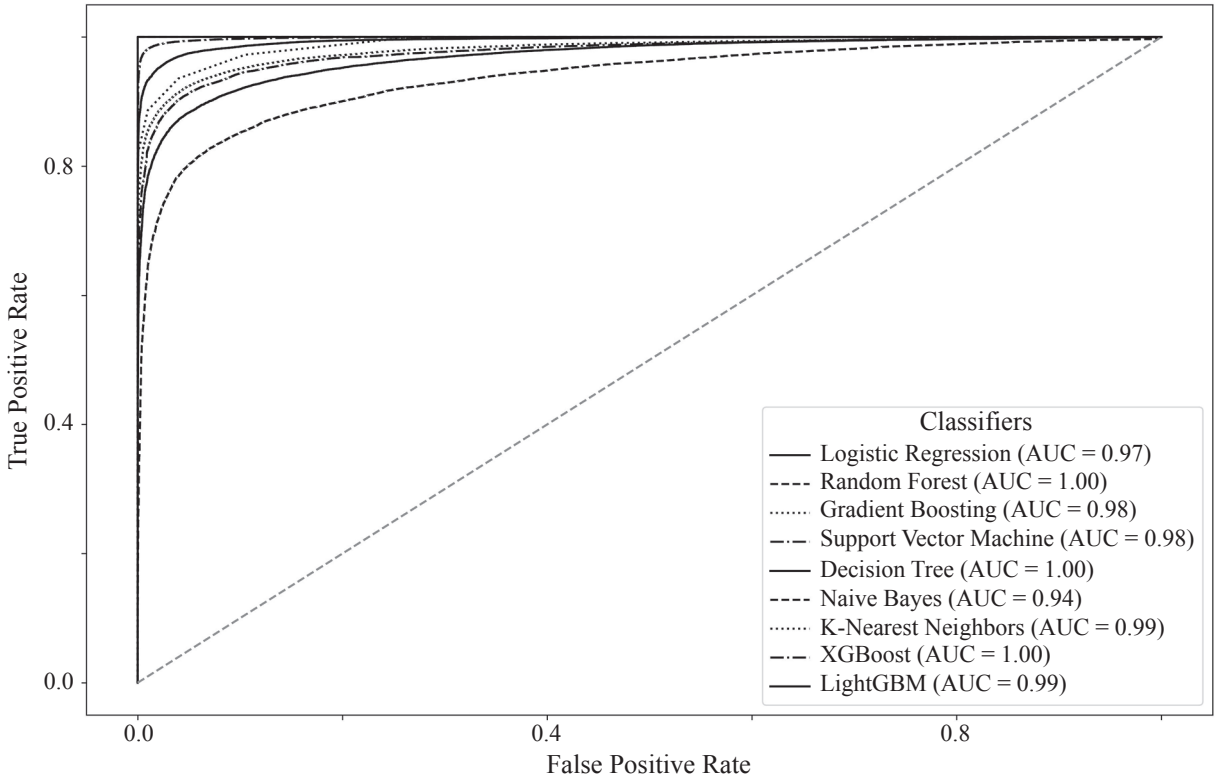

Fig. 6. ROC curves for all classifiers, illustrating their ability to differentiate between attack and non-attack states

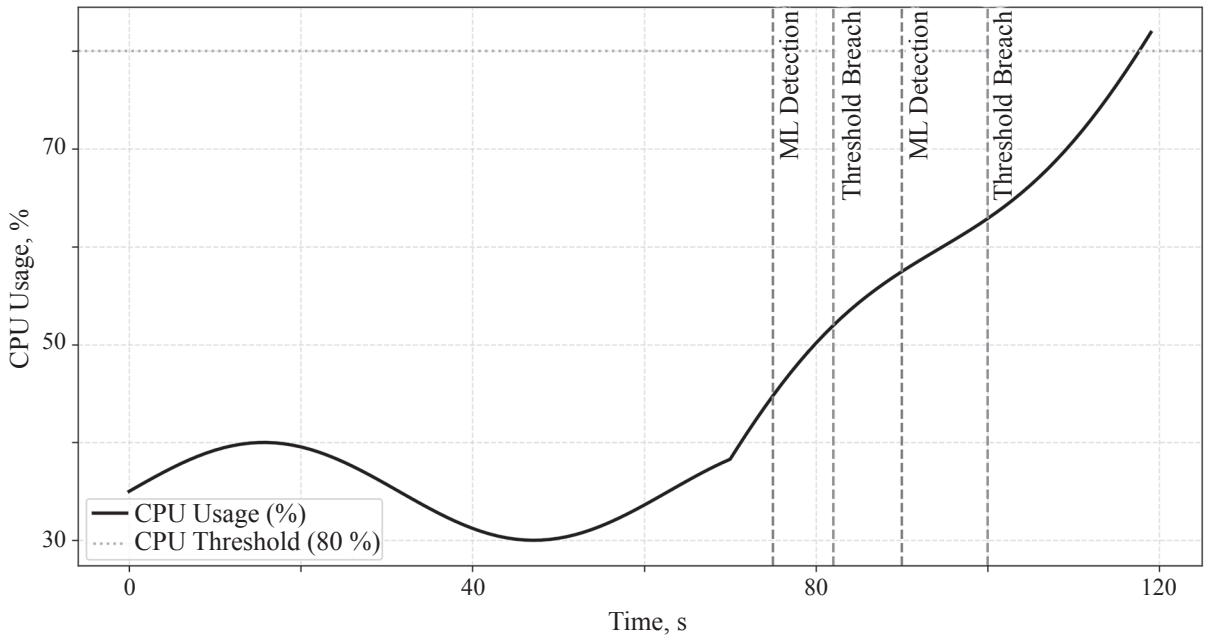

Fig. 7. Simulated comparative timeline of CPU usage during Denial-of-Service (DoS) attack scenarios. The figure illustrates two events where ML models (XGBoost, Random Forest) detect anomalous activity several seconds before the system reaches critical CPU utilization thresholds (80 %). This early detection window — ranging from 3 to 12 seconds — enables proactive mitigation before visible service degradation occurs, demonstrating the practical value of the proposed approach in real-time Kubernetes environments

Conclusion

This study evaluated machine learning methods for detecting DoS attacks in Kubernetes using metrics from five frameworks — Flask, Django, FastAPI, Node.js, and Golang. By combining node- and application-level metrics, we built a robust dataset that captures diverse workload behaviors. Statistical and correlation analyses highlighted the predictive power of CPU usage, memory consumption, and open file descriptors. Ensemble classifiers, especially XGBoost, Random Forest, and LightGBM, achieved the highest performance in identifying attacks. The approach proved scalable and adaptable across frameworks, advancing beyond static or framework-specific methods. Future work will focus on integrating lightweight monitoring tools and testing in resource-constrained edge environments to further improve real-time detection in cloud-native systems.

References

1. Sadiq A., Syed H.J., Ansari A.A., Ibrahim A.O., Alohaly M., Elsadig M. Detection of denial of service attack in cloud based kubernetes using eBPF. *Applied Sciences*, 2023, vol. 13, no. 8, p. 4700. https://doi.org/10.3390/app13084700
2. Cao C., Blaise A., Verwer S., Rebecchi F. Learning state machines to monitor and detect anomalies on a kubernetes cluster. *Proc. of the 17th International Conference on Availability, Reliability and Security*, 2022, pp. 1–9. https://doi.org/10.1145/3538969.3543810
3. Koksal S., Catak F. O., Dalveren Y. Flexible and lightweight mitigation framework for distributed denial-of-service attacks in container-based edge networks using Kubernetes. *IEEE Access*, 2024, vol. 12, pp. 172980–172991. https://doi.org/10.1109/ACCESS.2024.3501192
4. Tripathi A.A. *Attacking and Defending Kubernetes*. PhD thesis. Dublin Business School, 2024. Available at: https://esource.dbs.ie/items/eda4ea15-cedf-456b-93f9-6ce67e25c4bb (accessed: 02.12.2024).
5. Darwesh G., Hammoud J., Vorobeva A.A. Enhancing Kubernetes security with machine learning: a proactive approach to anomaly detection. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2024, vol. 24, no. 6, pp. 1007–1015. https://doi.org/10.17586/2226-1494-2024-24-6-1007-1015
6. Ghadeer D., Jaafar H., Vorobeva A.A. Security in Kubernetes: best practices and security analysis. *Journal of the Ural Federal District. Information Security*, 2022, no. 2 (44), pp. 63–69. https://doi.org/10.14529/secur220209

Литература

1. Sadiq A., Syed H.J., Ansari A.A., Ibrahim A.O., Alohaly M., Elsadig M. Detection of denial of service attack in cloud based kubernetes using eBPF // Applied Sciences. 2023. V. 13. N 8. P. 4700. https://doi.org/10.3390/app13084700
2. Cao C., Blaise A., Verwer S., Rebecchi F. Learning state machines to monitor and detect anomalies on a kubernetes cluster // Proc. of the 17th International Conference on Availability, Reliability and Security. 2022. P. 1–9. https://doi.org/10.1145/3538969.3543810
3. Koksal S., Catak F. O., Dalveren Y. Flexible and lightweight mitigation framework for distributed denial-of-service attacks in container-based edge networks using Kubernetes // IEEE Access. 2024. V. 12. P. 172980–172991. https://doi.org/10.1109/ACCESS.2024.3501192
4. Tripathi A.A. Attacking and Defending Kubernetes. PhD thesis. Dublin Business School, 2024. [Online]. URL: https://esource.dbs.ie/items/eda4ea15-cedf-456b-93f9-6ce67e25c4bb (accessed: 02.12.2024).
5. Darwesh G., Hammoud J., Vorobeva A.A. Enhancing Kubernetes security with machine learning: a proactive approach to anomaly detection // Scientific and Technical Journal of Information Technologies, Mechanics and Optics. 2024. V. 24. N 6. P. 1007–1015. https://doi.org/10.17586/2226-1494-2024-24-6-1007-1015
6. Ghadeer D., Jaafar H., Vorobeva A.A. Security in Kubernetes: best practices and security analysis // Journal of the Ural Federal District. Information Security. 2022. N. 2 (44). P. 63–69. https://doi.org/10.14529/secur220209

7. Darwesh G., Hammoud J., Vorobeva A.A. Enhancing kubernetes security: the crucial role of DevSecOps. *Proc. of the Institute for Systems Analysis Russian Academy of Sciences*, 2024, vol. 74, no. 3, pp. 78-88. https://doi.org/10.14357/20790279240309
8. Abed A.S., Clancy C., Levy D.S. Intrusion detection system for applications using linux containers. *Lecture Notes in Computer Science*, 2024, vol. 9331, pp. 123–135. https://doi.org/10.1007/978-3-319-24858-5_8
9. Zou Z., Xie Y., Huang K., Xu G., Feng D., Long D. A docker container anomaly monitoring system based on optimized isolation forest. *IEEE Transactions on Cloud Computing*, 2022, vol. 10, no. 1, pp. 134–145. https://doi.org/10.1109/TCC.2019.2935724
10. Srinivasan S., Kumar A., Mahajan M., Sitaram D., Gupta S. Probabilistic real-time intrusion detection system for docker containers. *Communications in Computer and Information Science*, 2019, vol. 969, pp. 336–347. https://doi.org/10.1007/978-981-13-5826-5_26
11. Tunde-Onadele O., He J., Dai T., Gu X. A study on container vulnerability exploit detection. *Proc. of the IEEE International Conference on Cloud Engineering (IC2E)*, 2019, pp. 121–127. https://doi.org/10.1109/IC2E.2019.00026
12. Flora J., Gonçalves P., Antunes N. Using attack injection to evaluate intrusion detection effectiveness in container-based systems. *Proc. of the IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2020, pp. 60–69. https://doi.org/10.1109/PRDC50213.2020.00017
13. Haq M.S., Nguyen T.D., Tosun A.S., Vollmer F., Korkmaz T., Sadeghi A.-R. SoK: a comprehensive analysis and evaluation of docker container attack and defense mechanisms. *Proc. of the IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 4573–4590. https://doi.org/10.1109/sp54263.2024.00268
14. Lin Y., Tunde-Onadele O., Gu X. Cdl: Classified distributed learning for detecting security attacks in containerized applications. *Proc. of the 36th Annual Computer Security Applications Conference*, 2020, pp. 179–188. https://doi.org/10.1145/3427228.3427236
15. Darwesh G., Hammoud J., Vorobeva A.A. A novel approach to feature collection for anomaly detection in Kubernetes environment and agent for metrics collection from Kubernetes nodes. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2023, vol. 23, no. 3, pp. 538–546. https://doi.org/10.17586/2226-1494-2023-23-3-538-546
16. LaValley M.P. Logistic regression. *Circulation*, 2008, vol. 117, no. 18, pp. 2395–2399. https://doi.org/10.1161/circulationaha.106.682658
17. Rigatti S.J. Random Forest. *Journal of Insurance Medicine*, 2017, vol. 47, no. 1, pp. 31–39. https://doi.org/10.17849/insm-47-01-31-39.1
18. Natekin A., Knoll A. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 2013, vol. 7, pp. 21. https://doi.org/10.3389/fnbot.2013.00021
19. Suthaharan S. Support vector machine. *Integrated Series in Information Systems*, 2016, vol. 36, pp. 207–235. https://doi.org/10.1007/978-1-4899-7641-3_9
20. Song Y., Lu Y. Decision tree methods: applications for classification and prediction. *Shanghai Archives of Psychiatry*, 2015, vol. 27, no. 2, pp. 130–135. https://doi.org/10.11919/j.issn.1002-0829.215044
21. Rish I. An empirical study of the naive Bayes classifier. *Proc. of the IJCAI-2001 Workshop on Empirical Methods in Artificial Intelligence*, 2001, pp. 41–46.
22. Kramer O. K-Nearest neighbors. *Intelligent Systems Reference Library*, 2013, vol. 51, pp. 13–23. https://doi.org/10.1007/978-3-642-38652-7_2
23. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794. https://doi.org/10.1145/2939672.2939785
24. Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., Liu T.-Y. LightGBM: a highly efficient gradient boosting decision tree. *Proc. of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 3149–3157.

7. Darwesh G., Hammoud J., Vorobeva A.A. Enhancing kubernetes security: the crucial role of DevSecOps // Proc. of the Institute for Systems Analysis Russian Academy of Sciences. 2024. V. 74. N 3. P. 78-88. https://doi.org/10.14357/20790279240309
8. Abed A.S., Clancy C., Levy D.S. Intrusion detection system for applications using linux containers // Lecture Notes in Computer Science. 2024. V. 9331. P. 123–135. https://doi.org/10.1007/978-3-319-24858-5_8
9. Zou Z., Xie Y., Huang K., Xu G., Feng D., Long D. A docker container anomaly monitoring system based on optimized isolation forest // IEEE Transactions on Cloud Computing. 2022. V. 10. N 1. P. 134–145. https://doi.org/10.1109/TCC.2019.2935724
10. Srinivasan S., Kumar A., Mahajan M., Sitaram D., Gupta S. Probabilistic real-time intrusion detection system for docker containers // Communications in Computer and Information Science. 2019. V. 969. P. 336–347. https://doi.org/10.1007/978-981-13-5826-5_26
11. Tunde-Onadele O., He J., Dai T., Gu X. A study on container vulnerability exploit detection // Proc. of the IEEE International Conference on Cloud Engineering (IC2E). 2019. P. 121–127. https://doi.org/10.1109/IC2E.2019.00026
12. Flora J., Gonçalves P., Antunes N. Using attack injection to evaluate intrusion detection effectiveness in container-based systems // Proc. of the IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC). 2020. P. 60–69. https://doi.org/10.1109/PRDC50213.2020.00017
13. Haq M.S., Nguyen T.D., Tosun A.S., Vollmer F., Korkmaz T., Sadeghi A.-R. SoK: a comprehensive analysis and evaluation of docker container attack and defense mechanisms // Proc. of the IEEE Symposium on Security and Privacy (SP). 2024. P. 4573–4590. https://doi.org/10.1109/sp54263.2024.00268
14. Lin Y., Tunde-Onadele O., Gu X. Cdl: Classified distributed learning for detecting security attacks in containerized applications // Proc. of the 36th Annual Computer Security Applications Conference. 2020. P. 179–188. https://doi.org/10.1145/3427228.3427236
15. Darwesh G., Hammoud J., Vorobeva A.A. A novel approach to feature collection for anomaly detection in Kubernetes environment and agent for metrics collection from Kubernetes nodes // Scientific and Technical Journal of Information Technologies, Mechanics and Optics. 2023. V. 23. N 3. P. 538–546. https://doi.org/10.17586/2226-1494-2023-23-3-538-546
16. LaValley M.P. Logistic regression // Circulation. 2008. V. 117. N 18. P. 2395–2399. https://doi.org/10.1161/circulationaha.106.682658
17. Rigatti S.J. Random Forest // Journal of Insurance Medicine. 2017. V. 47. N 1. P. 31–39. https://doi.org/10.17849/insm-47-01-31-39.1
18. Natekin A., Knoll A. Gradient boosting machines, a tutorial // Frontiers in Neurorobotics. 2013. V. 7. P. 21. https://doi.org/10.3389/fnbot.2013.00021
19. Suthaharan S. Support vector machine // Integrated Series in Information Systems. 2016. V. 36. P. 207–235. https://doi.org/10.1007/978-1-4899-7641-3_9
20. Song Y., Lu Y. Decision tree methods: applications for classification and prediction // Shanghai Archives of Psychiatry. 2015. V. 27. N 2. P. 130–135, https://doi.org/10.11919/j.issn.1002-0829.215044
21. Rish I. An empirical study of the naive Bayes classifier // Proc. of the IJCAI-2001 Workshop on Empirical Methods in Artificial Intelligence. 2001. P. 41–46.
22. Kramer O. K-Nearest neighbors // Intelligent Systems Reference Library. 2013. V. 51. P. 13–23. https://doi.org/10.1007/978-3-642-38652-7_2
23. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System // Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016. P. 785–794. https://doi.org/10.1145/2939672.2939785
24. Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., Liu T.-Y. LightGBM: a highly efficient gradient boosting decision tree // Proc. of the 31st International Conference on Neural Information Processing Systems. 2017. P. 3149–3157.

Authors

Ghadeer Darwesh — PhD Student, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 57226287648, https://orcid.org/0000-0003-1116-9410, ghadeerdarwesh32@gmail.com
Jaafar Hammoud — PhD Student, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 57222044000, https://orcid.org/0000-0002-2033-0838, hammoudgj@gmail.com
Alisa A. Vorobeva — PhD, Associate Professor, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 57191359167, https://orcid.org/0000-0001-6691-6167, vorobeva@itmo.ru

Авторы

Дарвиш Гадир — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 57226287648, https://orcid.org/0000-0003-1116-9410, ghadeerdarwesh32@gmail.com
Хаммуд Жаафар — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 57222044000, https://orcid.org/0000-0002-2033-0838, hammoudgj@gmail.com
Воробьева Алиса Андреевна — кандидат технических наук, доцент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 57191359167, https://orcid.org/0000-0001-6691-6167, vorobeva@itmo.ru

Received 26.03.2025
Approved after reviewing 02.09.2025
Accepted 30.09.2025

Статья поступила в редакцию 26.03.2025
Одобрена после рецензирования 02.09.2025
Принята к печати 30.09.2025