mathematics

MDPI

Article

The Improved Network Intrusion Detection Techniques Using the Feature Engineering Approach with Boosting Classifiers

Hari Mohan Rai [1], Joon Yoo [1,*] and Saurabh Agarwal [2,*]

[1] School of Computing, Gachon University, 1342 Seongnam-daero, Sujeong-gu, Seongnam 13120, Republic of Korea; drhmrai@gachon.ac.kr
[2] Department of Information and Communication Engineering, Yeungnam University, Gyeongsan 38541, Republic of Korea
* Correspondence: joon.yoo@gachon.ac.kr (J.Y.); saurabh@yu.ac.kr (S.A.)

Abstract: In the domain of cybersecurity, cyber threats targeting network devices are very crucial. Because of the exponential growth of wireless devices, such as smartphones and portable devices, cyber risks are becoming increasingly frequent and common with the emergence of new types of threats. This makes the automatic and accurate detection of network-based intrusion very essential. In this work, we propose a network-based intrusion detection system utilizing the comprehensive feature engineering approach combined with boosting machine-learning (ML) models. A TCP/IP-based dataset with 25,192 data samples from different protocols has been utilized in our work. To improve the dataset, we used preprocessing methods such as label encoding, correlation analysis, custom label encoding, and iterative label encoding. To improve the model's accuracy for prediction, we then used a unique feature engineering methodology that included novel feature scaling and random forest-based feature selection techniques. We used three conventional models (NB, LR, and SVC) and four boosting classifiers (CatBoostGBM, LightGBM, HistGradientBoosting, and XGBoost) for classification. The 10-fold cross-validation methods were employed to train each model. After an assessment using numerous metrics, the best-performing model emerged as XGBoost. With mean metric values of 99.54 $\pm$ 0.0007 for accuracy, 99.53 $\pm$ 0.0013 for precision, 99.54 $\pm$ 0.001 for recall, and an F1-score of 99.53 $\pm$ 0.0014, the XGBoost model produced the best performance overall. Additionally, we showed the ROC curve for evaluating the model, which demonstrated that all boosting classifiers obtained a perfect AUC value of one. Our suggested methodologies show effectiveness and accuracy in detecting network intrusions, setting the stage for the model to be used in real time. Our method provides a strong defensive measure against malicious intrusions into network infrastructures while cyber threats keep varying.

Keywords: network intrusion detection; machine learning; XGBoost; feature engineering; cyber security; threat detection

MSC: 68T01

Citation: Rai, H.M.; Yoo, J.; Agarwal, S. The Improved Network Intrusion Detection Techniques Using the Feature Engineering Approach with Boosting Classifiers. *Mathematics* **2024**, *12*, 3909. https://doi.org/10.3390/math12243909

Academic Editors: Cheng-Chi Lee and Dinh-Thuan Do

Received: 29 October 2024
Revised: 6 December 2024
Accepted: 11 December 2024
Published: 11 December 2024

1. Introduction

Cyberattacks constitute an increasing risk to modern organizations, which rely heavily on interconnected networks. As these attacks become increasingly complex and common—cyberattacks scaled by 38% in 2023 alone—there is an urgent need for strong and effective security methods and a strong and proficient protection mechanism. In this context, AI-powered network intrusion detection systems (NIDSs) have become critical for cyberspace security [1,2]. Traditional security measures frequently fall short of threatening attackers' shifting strategies, necessitating the development of innovative systems capable of predicting and preventing cyber-attacks. AI-based solutions address the shortcomings of conventional security measures, which frequently find it difficult to stay up to date with

attackers' changing strategies. For example, up to 30% of zero-day assaults go undetected by conventional intrusion detection systems [3]. This article delves into the field of cybersecurity, investigating how AI-powered NIDSs operate and their critical role in improving an organization's digital protection.

Traditional intrusion detection systems (IDSs) frequently encounter two major issues: limited response abilities and a significant proportion of false alarms [3]. By utilizing enormous volumes of data—analyzing around 500 terabytes of network traffic data annually—AI-powered intrusion detection systems (IDSs) provide a major breakthrough by differentiating between typical network activity and emerging threats [4]. This information enables them to detect suspicious behavior, even if it is a whole new sort of threat. AI's capacity to learn and identify patterns enhances the effectiveness of these systems, lowering the responsibility of security experts and fortifying organizational defenses. AI-driven systems can boost detection rates by up to 20% and minimize false positives by 50%, according to studies [5]. Conventional security measures are frequently insufficient in light of the growing complexity of cyber risks such as ransomware, which encountered a 37% increase in cyberattacks last year, zero-day vulnerabilities, and advanced attacks. Modern attackers may readily bypass rule-based signatures, which are the foundation of traditional intrusion detection systems and result in a 25% increase in missing attacks [6,7]. By adapting to the constantly evolving threat landscape, AI-driven NIDSs that integrate machine-learning (ML) and deep-learning (DL) technologies provide a potential solution to these problems. Response times have improved by 45%, and total detection accuracy has increased by 30% with the use of AI-driven technologies [8]. Artificial intelligence-driven NIDSs that possess ML/DL functionalities serve as strong defenses against the continually developing strategies adopted by hackers.

Literature Review

Table 1 presents an overview of the review conducted on network intrusion detection that demonstrates the efficacy and variety of methods used by ML and DL algorithms. (Faker & Dogdu, 2019) in Ref [9] produced the highest result with an outstanding accuracy of 99.99% for network intrusion detection utilizing the gradient boosting tree (GBT) classifier. On the other hand, Gao [10] produced the lowest detection accuracy with 95.50% using ML methods applied to the KDD CUP 99 dataset. The review table highlights certain areas that require more research and development in intrusion detection, even with the significant advances achieved in this field. For instance, the systematic evaluation conducted by the authors (C. Zhang et al., 2021) [11] exhibited a 99.90% accuracy along with high sensitivity, specificity, and F1-scores. Similarly, (Jia et al., 2019) [12] demonstrated the effectiveness of their proposed New Deep Neural Network (NDNN) technique in practical applications and achieved 99.90% accuracy in classification. (Devendiran et al., 2024) in Ref [13] used the Duag LSTM model on TON-IOT and NSL-KDD datasets to achieve outstanding accuracy performance in network intrusion detection (98.76% on the TON-IOT dataset and 99.65% on the NSL-KDD dataset). (Ren et al., 2022) in Ref [14] used ID-RDRL to increase performance on the CSE-CIC-IDS2018 dataset by identifying essential features, decreasing redundancy by 80%, and improving detection with DRL. However, more testing is necessary for particularly complex or dynamic networks.

Also, we can observe from the review table that a variety of datasets are utilized by the various researchers along with various intrusion detection algorithms. The multiple datasets are used by many researchers [9,12,13,15–18] for the detection of intrusion, among them (Rincy N & Gupta, 2021) [16] consistently displayed good accuracy of classification, highlighting the adaptable nature of their methodology. Similarly, Xu et al. (2018) [13] also achieved high accuracy values across multiple datasets and demonstrated the versatility of their IDS methods. The good accuracy of intrusion detection was also achieved by (Faker & Dogdu, 2019) [9], who emphasized the success of their feature engineering and ensemble methodologies. Similarly, employing a stacking-based ensemble technique (Ali et al., 2023) [15] obtained good performance with an F1-score of 98.24. The practical issues

of reducing the training and detection time required for investigating the authenticity of real-world simulations have been addressed by (Belouch et al., 2018) [19] and (Ali et al., 2023) [15] when evaluating real-world applicability. The study results offer significant contributions to the discourse about the real-world use and validation of intrusion detection techniques. (Carta et al., 2020) in Ref [20] used the LFE approach to improve traditional IDS performance on the NSL-KDD dataset by introducing features and discretizing values. However, the technique struggles to deal with the increasing level of complexity and diversity of network operations. (R.M. et al., 2020) in Ref [21] used a Deep Neural Network (DNN) to achieve a 15% increase in accuracy and a 32% decrease in time complexity on Kaggle data. This resulted in faster alarms and improved security, but the methodology required greater scalability to efficiently and quickly manage growing threats. Saia et al. (2019) [22] employed the DEFS model, which outperformed previous systems on the NSL-KDD dataset, even in biased circumstances. Despite its ability to reduce false positives, it may still fail to detect completely novel attack patterns.

Table 1. Review table for the AI-based network intrusion detection system.

References	Years	Techniques	Datasets	Findings	Challenges/Limitations
[13]	2024	Dugat-LSTM	TON-IOT, NSL-KDD	The proposed approach performed well in network intrusion detection, achieving 98.76% accuracy with the TON-IOT dataset and 99.65% accuracy with the NSL-KDD dataset.	The primary issues are assuring the effectiveness of the model in a variety of network environments and managing computational requirements.
[23]	2023	PCA + NIDS	NSL-KDD	Outperforms other methods, particularly Naive Bayes, in terms of accuracy. Achieved a low-performance time, indicating efficiency in real-time applications.	The scalability of the diverse intrusion scenarios and network architectures needs evaluation.
[15]	2023	Stacking	CICIDS2017, Private, CIPMAIDS2023-1	Identified and addressed limitations in existing NIDS datasets. The stacking-based ensemble approach outperforms the overall state-of-the-art.	Real-world simulation fidelity and effectiveness across diverse network scenarios need validation.
[10]	2022	CNN, BiLSTM, AM, C5.0 DT	KDD CUP 99	Improved network intrusion detection effect compared to AE-AlexNet and SGM-CNN methods. Significant enhancement in the performance of the network intrusion detection system.	The generalizability of the proposed method to other datasets or real-world scenarios is not thoroughly discussed.
[24]	2022	Ensemble, SVC, ExtraTree	KDD CUP 99	Voting-based ensemble method outperforms individual approaches. Improved performance compared to an unoptimized implementation.	Scalability to handle large datasets and increased traffic; generalization across various types of attacks needs validation.
[14]	2022	ID-RDRL	CSE-CIC-IDS2018	The ID-RDRL model has remarkable potential in complex network circumstances as it identifies the best subset of features, eliminates around 80% of redundant features, and improves intrusion detection performance using DRL.	Although the model appears promising, more testing and modification may be necessary to ensure its efficacy in extremely complex or dynamic network contexts.

Table 1. *Cont.*

References	Years	Techniques	Datasets	Findings	Challenges/Limitations
[25]	2021	DLA, SDHT, TL, Bootstrapping	UNSW-NB15	Deep-learning-enhanced intrusion detection models, achieving high accuracy.	Opportunities for improvement, particularly in feature reduction methods.
[11]	2021	SAE + RF	CICIDS2017	Improved performance compared to previous work. High-level accuracy was achieved for restored traffic.	Lack of detailed exploration of scalability and adaptability to diverse network scenarios.
[16]	2021	NID-Shield, CAPPER, ML	UNSW-NB15, NSL-KDD	High accuracy rates and low false positive rates were achieved on both UNSW-NB15 and NSL-KDD datasets. The proposed method shows promising performance.	The evaluation focuses on specific datasets; generalization to different network environments and attack scenarios requires validation.
[20]	2020	AB, DT, GB, MP, RF	NSL-KDD	The performance of conventional state-of-the-art intrusion detection systems was improved by the LFE technique by adding extra features and discretizing their values.	It may become difficult to differentiate between acceptable and unauthorized activities due to the method's inability to keep up with the growing complexity and variety of network activities.
[21]	2020	Deep Neural Network (DNN)	Kaggle	In comparison to current ML techniques, the DNN model demonstrated a 15% increase in accuracy and a 32% decrease in time complexity, resulting in fast alerts and the enhanced safety of confidential information.	Scalable solutions are necessary to successfully address developing threats since malicious attacks are unexpected and change quickly, posing problems to IDS models.
[22]	2019	DEFS Model	NSL-KDD	The DEFS model outperforms current state-of-the-art systems in classification, especially in challenging environments with imbalanced datasets.	The DEFS model may still have difficulty detecting completely new attack patterns, even though it reduces the number of false positives by grouping related event patterns.
[17]	2019	caXGB, gcForest, GoogLeNetNP	UNSW-NB15, CICIDS2017	Multiple-layer representation learning model demonstrates superior performance in detecting fine-grained attacks and handling imbalanced datasets.	Adaptability to evolving attack scenarios and robustness in real-world, dynamic network environments pose challenges.
[9]	2019	DNN, Ensemble, RF, GBT	UNSW NB15, CICIDS2017	High accuracy is achieved with DNN for binary and multiclass classification. GBT excelled in binary classification with the CICIDS2017 dataset.	Limited exploration of scalability, computational efficiency, and lack of comprehensive evaluation metrics.
[12]	2019	NDNN	KDD CUP 99, NSL-KDD	The NDNN-based method enhances IDS performance with a remarkable accuracy rate.	A comprehensive set of evaluation metrics and practical feasibility in real-world IDSs need discussion.

Table 1. *Cont.*

References	Years	Techniques	Datasets	Findings	Challenges/Limitations
[19]	2018	SVM, NB, DT, RF	UNSW-NB15	Random forest classifier excels with superior detection accuracy and prediction time compared to other classifiers.	The challenge of reducing training and detection time in intrusion detection systems.
[18]	2018	RNN (GRU), MLP	KDD CUP 99, NSL-KDD	GRU outperforms LSTM as a memory unit for IDSs. Bidirectional GRU achieves the best performance. The proposed IDS model demonstrates leading performance on both datasets.	Lack of interpretability for deep-learning models with complex architectures.

Abbreviations: AM: Attention Mechanism; BiLSTM: Bidirectional Long Short-Term Memory; Bootstrapping: Bootstrapping; caXGB: caXGBoost; CAPPER: CAPPER feature subset selection; C5.0 DT: C5.0 decision tree; CNN: Convolutional Neural Network; DLA: Deep-Learning Architectures; DNN: Deep Feed-Forward Neural Network; DEFS: Discretized Extended Feature Space; DT: decision tree; GBT: gradient boosting tree; gcForest: gcForest; GoogLeNetNP: GoogLeNetNP; Dugat-LSTM: Gated Attention Dual Long Short Term Memory; GRU: Gated Recurrent Units; MLP: Multilayer Perceptron; ML: machine learning; NID-Shield: hybrid network intrusion detection system; NB: Naïve Bayes; NDNN: New Deep Neural Network; PCA: Principal Component Analysis; RF: random forest; RNN: Recurrent Neural Network; SAE: Stacked Autoencoder; SDHT: Semi-Dynamic Hyperparameter Tuning; SVM: Support Vector Machine; TL: Transfer Learning.

While numerous researchers have made significant contributions to the field of network-based intrusion detection, there remains ample room for improvement. In this study, we aim to highlight key findings and contributions that advance the understanding and effectiveness of intrusion detection systems.

1. Open access dataset: we utilized a TCP/IP-based dataset comprising 25,192 samples categorized into various protocols and two main categories: normal network traffic and anomalies.
2. Multiple preprocessing techniques: we applied several unique preprocessing techniques, including label encoding, correlation analysis, custom label encoding, and iterative label encoding, to enhance data quality and model performance.
3. Novel feature engineering approach: we developed a novel feature scaling algorithm for scaling the extracted dataset features, coupled with RF-based feature selection, to improve model accuracy and efficiency.
4. Diverse classifiers: our study incorporated a range of classifiers, including conventional ML classifiers (Naive Bayes, Logistic Regression, and Support Vector Classifier) and boosting classifiers (CatBoost, LightBoost, HistGradientBoosting, XGBoost), to evaluate and compare their performances.
5. 10-Fold cross-validation approach: we employed a 10-fold cross-validation approach to train all the utilized models comprehensively, ensuring robustness and reliability in model training and evaluation.
6. Multiple assessment metrics: we assessed model performance using various metrics, including accuracy, F1-score, AUC-ROC, and precision, to provide a comprehensive evaluation of each classifier's effectiveness in intrusion detection.
7. State-of-the-art comparison: finally, we compared our proposed methodology with state-of-the-art techniques to validate and verify its performance, highlighting its efficacy and potential for practical application in real-world scenarios.
8. NIDS Background: we provided a detailed background of NIDS techniques, including visual diagrams, to help young researchers understand the evolution from traditional rule-based systems to modern machine-learning approaches, supporting future advancements in the field.

The manuscript continues with a structured organization: Section 2 provides the background of the network-based intrusion detection system (NIDS). Section 3 elaborates on the materials and methods, offering insights into the proposed methodology. In Section 3, we present comprehensive results derived from diverse models, covering both training and

testing phases. Section 4 engages in a discussion on state-of-the-art techniques, juxtaposing them with our proposed methodology to underscore its strengths and areas for enhancement. Finally, Section 5 summarizes these concepts in a conclusion and recommends potential future research areas.

2. Background

Since its inception in the early 1980s, intrusion detection has seen substantial development [26]. James Anderson and Dorothy Denning introduced the use of audit trails and system logs to monitor and identify unauthorized access and suspect user activity, providing the framework for intrusion detection systems (IDSs). In the mid-1990s, when the internet witnessed exponential expansion, network intrusion detection systems (NIDSs) emerged to track network traffic in real time and detect malicious activity using pre-established attack signatures [26,27]. Although these early NIDS systems were effective at identifying known threats, they had a key flaw: they could not detect emerging or undisclosed attack patterns, making them vulnerable to zero-day vulnerabilities and fast-changing malware.

One key challenge in NIDS design is accurately classifying network traffic data, which can be modeled mathematically using a function that optimizes classification performance $L(f(x), y)$, where L represents the loss function, $f(x)$ is the predicted label from the model, and y is the true label, as given in Equation (1):

$$\min_{f} \sum_{i=1}^{n} L(f(x_i), y_i) \tag{1}$$

Mathematically, the detection process for NIDS can be represented as an optimization problem where the objective is to minimize both false positives (FPs) and false negatives (FNs). The optimization function $f_{\min}$ can be expressed as given in Equation (2):

$$f_{\min} = \alpha \cdot \frac{\text{FP}}{\text{FP} + \text{TN}} + \beta \cdot \frac{\text{FN}}{\text{TP} + \text{FN}} \tag{2}$$

where α and β are weighting factors that balance the importance of false positives and false negatives, respectively, and TP and TN are true positives and true negatives.

NIDSs that relied on signatures had difficulty keeping up with the growing complexity of cyber-attacks. These systems were difficult to detect since they depended on famous attack patterns and, therefore, needed regular upgrades. Improved detection capabilities, fewer false positives, and more flexible responses to security events were made possible by the emergence of increasingly sophisticated NIDS technologies in response to these issues [28,29]. These technologies included hybrid and AI-powered systems. ML and DL approaches, in particular, have revolutionized the field of network security through AI-powered NIDSs, improving threat detection for both known and undiscovered threats while dynamically learning from network data.

2.1. Network Intrusion Detection Systems (NIDSs)

NIDSs are generally classified into the following types according to how they detect threats:

2.1.1. Signature-Based NIDSs

Signature-based NIDSs monitor network traffic and compare it to a database of known threat signatures. Though it has to be upgraded periodically to stay up to date, this method works effectively for identifying well-known threats. In the face of changing attack vectors, their adaptability is reduced as they battle against new threats and zero-day assaults [3]. Figure 1a depicts the fundamental architecture of a signature-based NIDS. It illustrates how network traffic travels via firewalls and switches as well as other important security components on the path from the Internet to an internal network through a router. A

database comprising known attack signatures is used by the signature-based NIDS to compare the traffic to itself. If a match is identified, the system sends out notifications for prospective risks. The NIDS analyzes traffic aimed at internal resources like servers and end users.

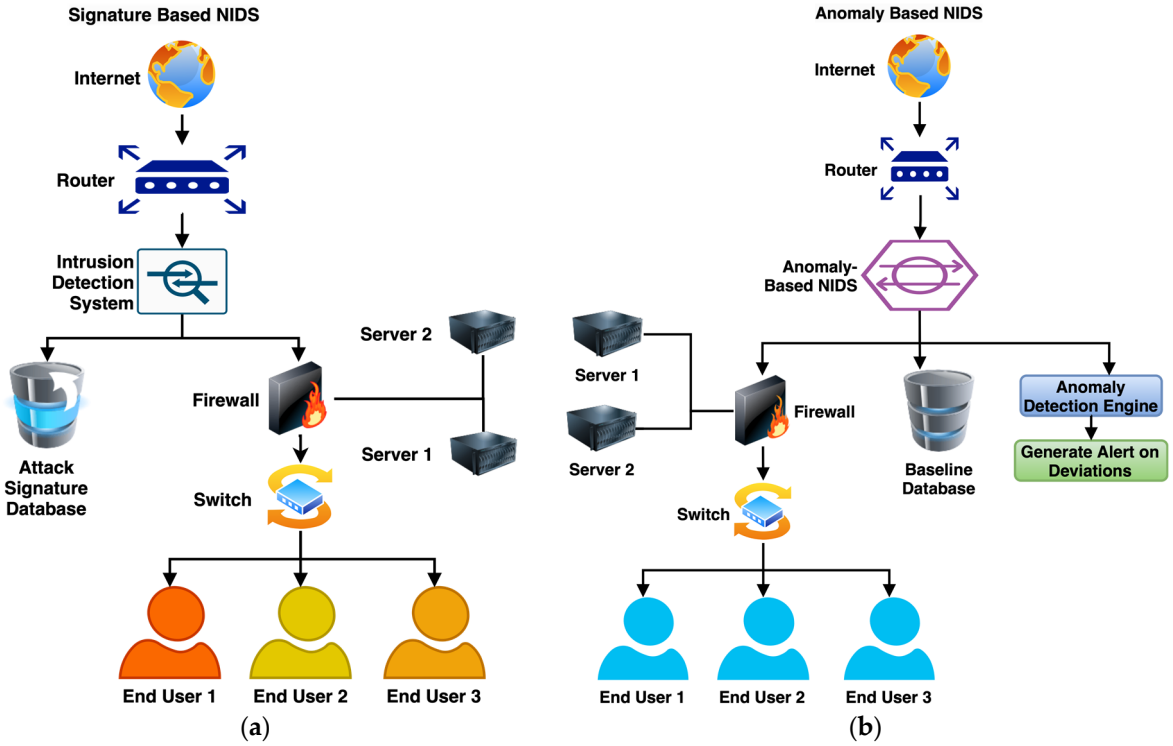


Figure 1. The schematic diagram of (**a**) signature-based NIDSs and (**b**) anomaly-based NIDSs.

To formalize this, let S be the set of known signatures, and T be incoming network traffic. Signature matching can be defined as expressed in Equation (3):

$$M(t) = \begin{cases} 1, & if\ t\ \epsilon\ s \\ 0, & Otherwise \end{cases} \tag{3}$$

where $M(t)$ outputs 1 if a match is found in S; otherwise, it is 0.

2.1.2. Anomaly-Based NIDS

NIDSs that are based on anomalies create a standard of "normal" network activity and identify any changes as possible threats. These systems are more likely to produce false positives even if they are better equipped to identify unidentified attacks or emerging threats. It can be challenging to differentiate between real anomalies and legitimate malicious activities when there is even a small deviation from the normal network behavior [30]. Figure 1b depicts an NIDS that is based on anomalies. It analyzes Internet traffic via a router and compares it to a database of regular behavior. The anomaly detection engine is triggered by a change from this baseline and provides notifications regarding potential threats. In order to ensure that the NIDS analyzes and protects all devices that are connected by identifying unusual activity, traffic must pass via a firewall, switch, and servers before reaching end users.

Anomaly detection is often modeled using statistical or machine-learning techniques. One common approach is based on calculating the Mahalanobis distance D_M (Equation (4)), where deviations from the baseline are flagged as potential threats:

$$D_M(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)} \tag{4}$$

where x is a vector of observed data points, μ is the mean of the baseline data, and Σ is the covariance matrix of the baseline. Any value of $D_M(x)$ exceeding a threshold θ triggers an alert.

2.1.3. Hybrid NIDS

Figure 2a shows the basic block design of a Hybrid NIDS, which combines signature-based and anomaly-based identification methodologies. Switches, routers, and firewalls are used to monitor internet traffic. The system makes use of a baseline database to identify abnormalities in network activity and an attack signature database to identify known threats. Signature-based detection uses predetermined signatures to identify threats and generate warnings when attacks are identified. By identifying unusual network behavior, anomaly-based detection raises the alarm for fresh or unidentified threats. Hybrid NIDS provides a balanced and adaptable strategy by combining various techniques, increasing overall detection accuracy, and reducing false positives [31].

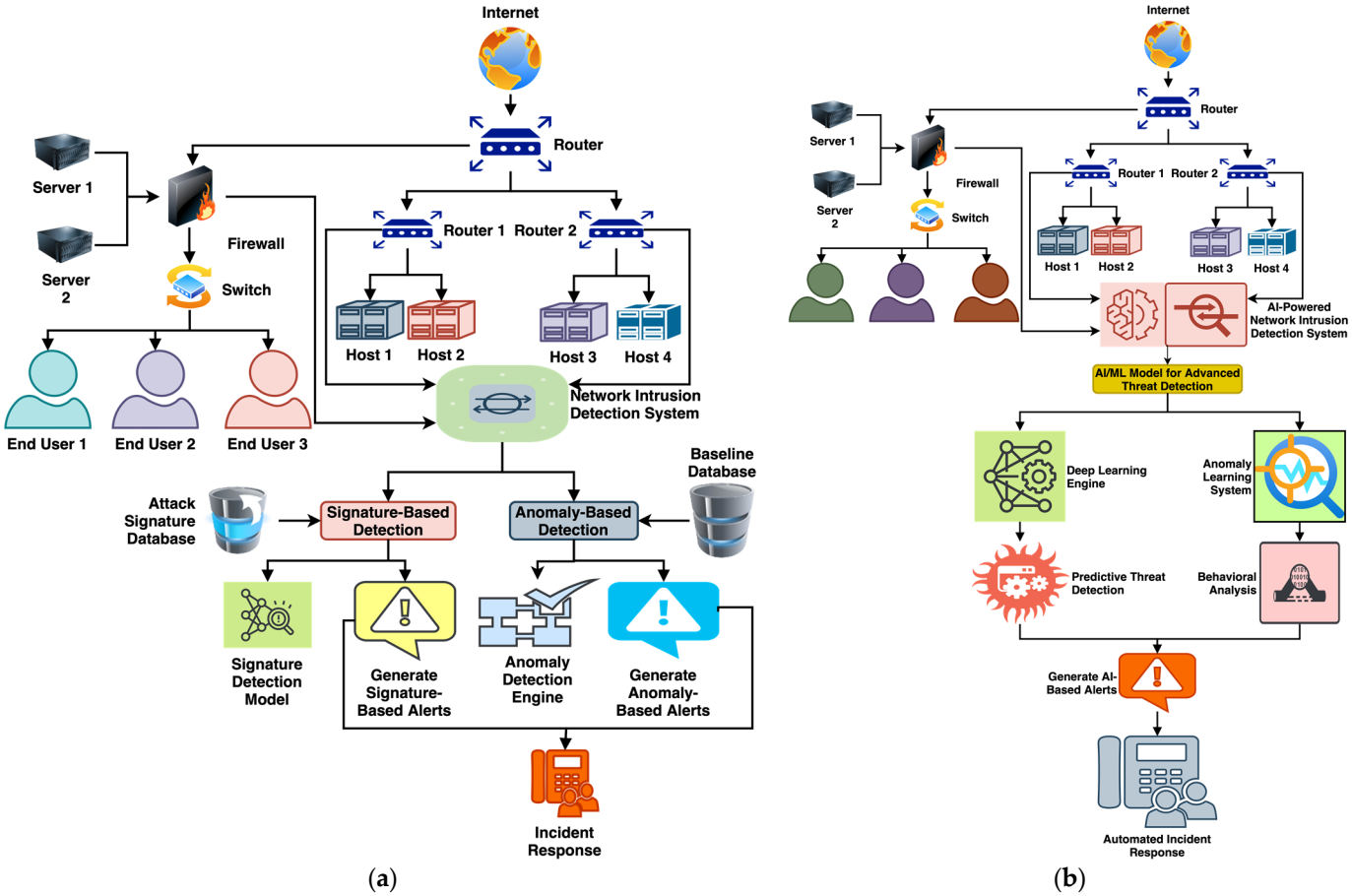


Figure 2. The schematic diagram of (**a**) Hybrid NIDSs and (**b**) AI-powered NIDSs.

A combined detection score for hybrid systems can be modeled as a weighted sum of the detection scores of both methods, as expressed in Equation (5):

$$\text{Hybrid Detection Score} = \alpha \cdot \text{Signature Score} + (1 - \alpha) \cdot \text{Anomaly Score} \quad (5)$$

where α is the weighting factor that balances the contributions of the signature and anomaly detection methods.

2.1.4. AI-Powered NIDS

The state-of-the-art intrusion detection technology is represented by NIDSs with AI capabilities. These systems analyze huge volumes of network data in real time using

complex ML and DL algorithms, which allows them to identify both known and unknown attack patterns. AI-powered NIDSs, compared to conventional systems, constantly acquire knowledge from network activity, enhancing their detection skills over time. Moreover, they minimize the need for periodic signature updates and are more capable than their predecessors of adjusting to new cyber threats [31]. Figure 2b depicts the block design for an AI-powered network intrusion detection system (NIDS). In this architecture, data are routed from the internet to various servers and end users via routers, firewalls, and switches. In order to detect advanced threats, network traffic, and behavior patterns are analyzed by AI and machine-learning (ML) models. This system combines a behavioral analysis engine that determines possible risks based on user activities with an anomaly learning system that recognizes deviations from typical behavior. The detection of complex attacks is greatly enhanced while limiting false positives because of the model's ability to identify both existing and unexplored threats. The system sends notifications in order to enable quick incident reactions when threats are identified.

In deep-learning-based NIDSs, for example, data X are passed through multiple layers of the neural network to obtain a prediction probability $P(Y = 1|X)$, as expressed in Equation (6):

$$P(Y = 1|X) = \sigma(W_n^T \sigma(W_{n-1}^T \ldots \sigma(W_1^T X + b_1) + \cdots + b_{n-1}) + b_n) \quad (6)$$

where W_i and b_i are the weight matrices and bias vectors for each layer i, and σ is the activation function (e.g., sigmoid or ReLU).

2.2. Evolution of NIDS and the Role of AI

The incorporation of AI into NIDSs has transformed network security by solving the limitations of previous detection approaches. Although they were successful in detecting known threats, early signature-based systems proved to be less effective when confronted with more complex attack methods such as zero-day vulnerabilities, polymorphic malware, and advanced persistent threats (APTs) [32]. These systems handled cyber threats rapidly rather than proactively due to their strict adherence to pre-established regulations. But because of AI-driven NIDSs, systems are now more dynamic, able to recognize previously undisclosed attack pathways and learn from data trends. Utilizing complex pattern recognition techniques, these systems examine large volumes of network traffic in order to identify minute variations that could be signs of malevolent activity. Because of this, AI-powered NIDSs are better at spotting complex cyberattacks and quickly adjusting to changing threats, which lowers the number of false positives and increases detection accuracy [33]. AI-enhanced NIDSs are a crucial part of contemporary network security architectures since studies have demonstrated that they may minimize false positives by up to 50% and increase detection accuracy by 20% [34].

2.3. The Modern Cybersecurity

As attacks become more complex and frequent, the requirement for more effective NIDS solutions becomes increasingly crucial. This increasingly threatening environment has led to the emergence of AI-powered NIDSs, which offer real-time detection and response capabilities against both established and new threats. The number of cyberattacks rose by 38% in 2023 alone, highlighting the need for more flexible, AI-driven solutions that can keep up with attackers' changing tactics [35]. AI-powered NIDSs provide a more thorough and flexible approach to network security by combining ML, DL, and anomaly detection approaches. These technologies offer real-time information, which shortens reaction times and allows organizations to take quick action to neutralize possible risks. The ongoing advancement of AI in NIDS technology is projected to play an important role in network protection in an increasingly linked world.

3. Materials and Methods

The methods used to classify network intrusions using proposed ML models are presented in this section. Figure 3 shows a block diagram that represents the proposed methodology utilized in this work. First, the method starts with gathering data using datasets that are obtained from a publicly accessible platform. This maintains data integrity and separates the dataset for training and testing. It contains both normal and anomalous data based on different protocols (TCP/IP). In the 2nd stage, we utilized various data preprocessing tasks to prepare the dataset for the classification. In this stage, we employed various preprocessing techniques, including data cleaning, label encoding, correlation analysis, custom label encoding, and iterative label encoding, in addition to exploratory data analysis (EDA). In the 3rd step, we performed rigorous feature engineering approaches, including feature extraction, feature scaling, and feature selection using the random forest (RF) classifier. In the 4th step, we utilized various boosting classifiers to detect the intrusion from the network data. The utilized ML models were LightGBM, XGBoost, CatBoost, HistGradientBoosting, Logistic Regression, SVC, and Naive Bayes. Subsequently, 10-fold cross-validation was employed to train all the employed ML models on 70% of the training data. Metrics like F1-score, precision, recall, AUC-ROC, and accuracy were then used to evaluate the model's performance. To adjust the model parameters for improved performance, tuning for the parameters was carried out. The trained models were then used to make predictions in situations that happened in real time. Ensuring the efficient identification and classification of network attacks using a structured methodology complements improved cybersecurity. The section below provides additional details on each step used to implement ML methodologies for NIDSs.

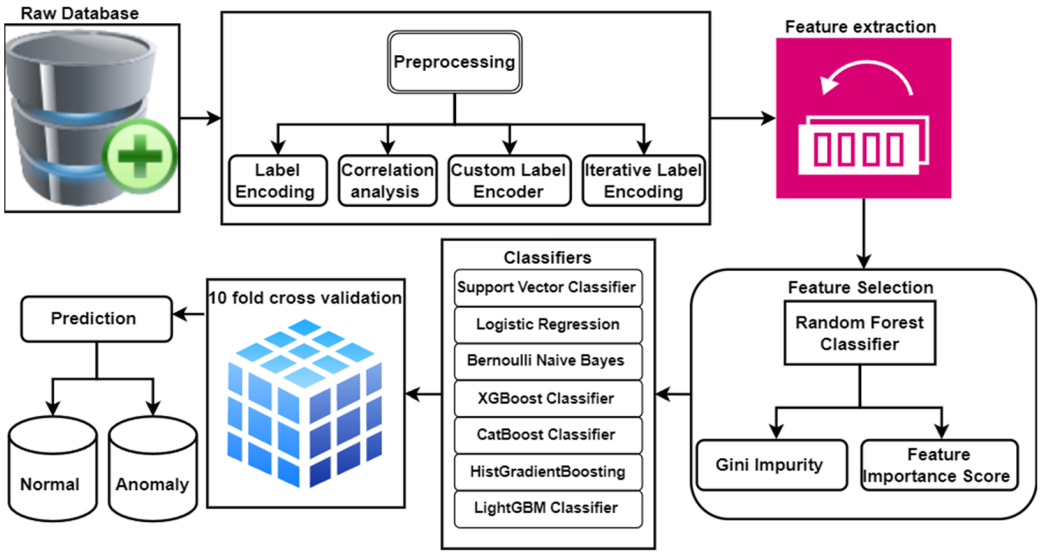


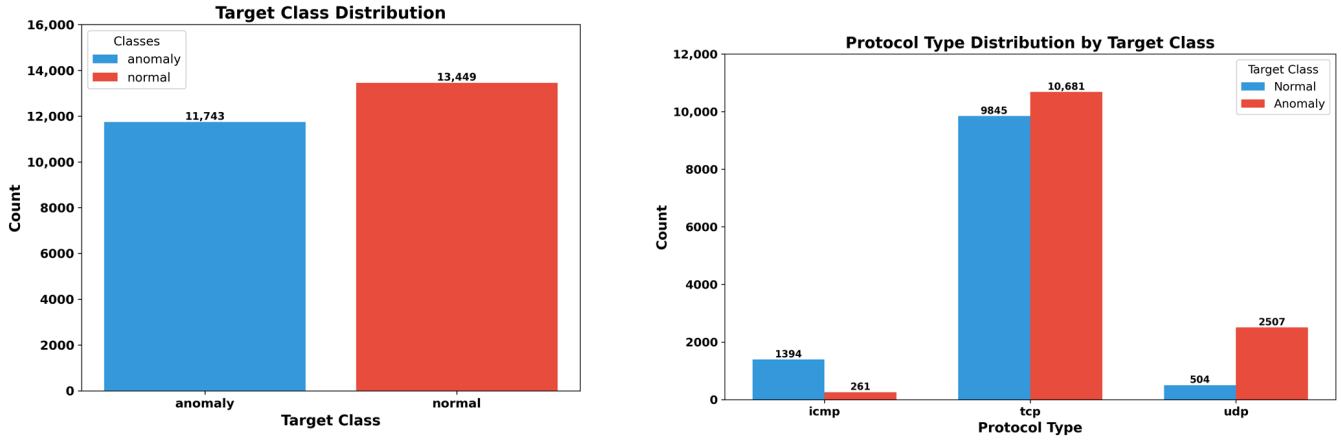
Figure 3. The block diagram of the proposed methodology utilized for the NIDS using the ML approach.

3.1. NIDS Dataset

The utilized database comprises a diverse array of intrusions meticulously simulated within a military network setting and sourced from the Kaggle repository [36]. This environment was created to gather raw TCP/IP dump data, emulating the structure of a typical Local Area Network (LAN). The LAN was meticulously structured to resemble a genuine operational environment and subjected to a barrage of multiple attacks. A connection within this context is delineated as a sequence of TCP packets that initiate and conclude within a defined time duration. During this timeframe, data traverse between a source IP address and a target IP address, adhering to specific and well-defined protocols. Notably, each connection is categorized as either normal or indicative of an attack, with

a singular, specific attack type assigned to it. Each record corresponding to a connection comprises approximately 100 bytes.

Within the realm of TCP/IP connections, a total of 41 quantitative and qualitative features were extracted from both normal and cyber-attacked data, encompassing 3 qualitative and 38 quantitative features. The class variable is binarily categorized into 'Normal' and 'Anomalous'. The 'Normal' category comprises 13,449 data samples, whereas the 'Anomalous' category incorporates 11,743 data samples, as visually depicted in Figure 4a through a bar graph. Furthermore, granularity is applied to the dataset, where it is categorized into three subgroups based on distinct protocols: ICMP, TCP, and UDP. Notably, the TCP protocol dominates the dataset, constituting approximately 80–90% of the data, as elucidated in Figure 4b. Additionally, the distribution of destination, host, and service count within the dataset is visually represented in Figure 5.



(**a**) (**b**)

Figure 4. Comparative distribution of dataset in (**a**) normal and anomaly classes and (**b**) protocol types.

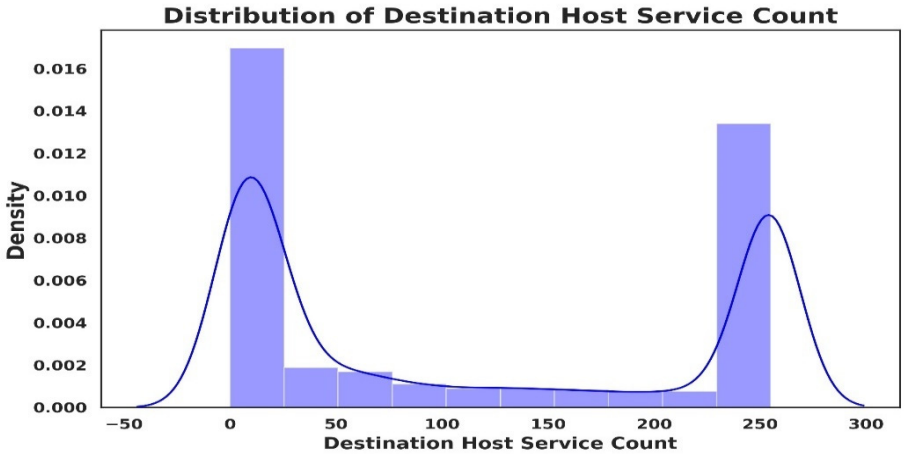


Figure 5. Distribution patterns of destination, host, and service count in the dataset.

3.2. Data Preprocessing

Preprocessing raw data is essential for standard ML models and boosting classifiers for intrusion detection. In this work, we employed a series of preprocessing techniques to ensure the input data were well prepared for the training and evaluation of the utilized models. Before performing the primary preprocessing steps, we handled missing or incomplete data, which might result in biased or incorrect model results. We removed rows with a large percentage of missing data or imputed missing values to address the

missing values that were found in the dataset. The following subsections explain the specific preprocessing techniques we performed, along with their logical sequence.

3.2.1. Label Encoding

After cleaning the dataset, we used label encoding to convert the categorical attributes into a numerical representation that ML algorithms would recognize. Within intrusion detection, the class variable representing network connections was categorized into two distinct groups: abnormal and normal [37]. The term "LabelEncoder" usually designates a procedure or function found in Python machine-learning libraries like scikit-learn. LabelEncoder can be comprehended within a broad mathematical framework. The LabelEncoder is given a set of categorical labels, y, where $y \in \{y_1, y_2, \ldots, y_K\}$. For each categorical label y_i, it assigns a unique integer z. This can be expressed mathematically, as given in Equation (7).

$$z = \text{LabelEncoder}(Y[i]) \tag{7}$$

To put it practically, the LabelEncoder makes sure that every distinct categorical label, y_i, is mapped to a distinct integer, z, in the interval $[0, K-1]$, in which K is the total number of distinct categories. This label encoding was utilized for cleaned data, which contain only a specific number of unique values, preparing the dataset for the next stage in correlation analysis.

3.2.2. Correlation Analysis

After label encoding, we performed correlation analysis to detect and handle associations between the encoded attributes [38]. The main goal was to find attributes that significantly correlate with the encoded target variable, Y_{encoded}. This variable, which is appropriate for binary classification tasks in intrusion detection, reflects transformed class labels in the numerical format [39]. Equation (8) quantifies the correlation coefficient ρ_j and expresses the importance as well as the trajectory of the proportional relationship between each characteristic X_j and Y_{encoded}.

$$\rho_j = \frac{\text{cov}\left(X_j, Y_{\text{encoded}}\right)}{\sigma_{X_j} \sigma_{Y_{\text{encoded}}}} \tag{8}$$

where ρ_j stands for the correlation coefficient across a feature X_j and the encoded target variable Y_{encoded}. This coefficient is calculated by dividing the covariance $\text{cov}\left(X_j, Y_{\text{encoded}}\right)$ by the product of the standard deviations σ_{X_j} and $\sigma_{Y_{\text{encoded}}}$. ρ_j normalizes covariance to measure both the magnitude and the direction of a linear relationship between X_j and Y_{encoded}.

3.2.3. Custom Label Encoding

Following the correlation analysis, we used custom labels encoding categorical variables with a high number of distinct categories. Unlike conventional label encoding, custom label encoding is intended to handle high-cardinality attributes more efficiently by combining similar groups or assigning weights depending on the specific knowledge or frequency of occurrence. Real-world datasets frequently include categories (labels) that are unique to them. This may lead to issues when label encoding takes place, which is the process of converting text labels into numerical values for ML models. This problem is addressed by a custom label encoder, also known as LabelEncoderExt [40]. This encoder assigns a specific label to the unknown values (denoted as C, as shown in Equation (9)), ensuring the consistent handling of the missing or unknown data points [41].

$$X_{encoded\_col}[i] = \begin{cases} LabelEncoder(X_{col}[i]), & if\ X_{col}[i]\ is\ known \\ LabelEncoder(X_{col}[i] + C), & if\ X_{col}[i]\ is\ unknown \end{cases} \tag{9}$$

By addressing high-cardinality categories, custom label encoding significantly simplifies the data and prepares it for Iterative Label Encoding.

3.2.4. Iterative Label Encoding

Finally, we used iterative label encoding to handle instances when categorical variables have interconnections that must be retained. It is an important preprocessing step for converting raw TCP/IP dump data into an appropriate structure for intrusion detection classifiers. This complicated approach involves successively assigning a unique label encoder to every categorical column of the feature matrix (X). Because of the dataset's many categorical attributes, this encoding needs to be iterative. By providing uniform label encoding for every column, this technique preserves data integrity and provides an exclusive encoding framework [40,41]. The main goal of this iterative application is to find a consistent encoding scheme that applies to all the different types of categorical columns in the dataset.

The result of these methods, as generously organized by Algorithm 1, is a dataset that is not only ready but also finely tuned to the complex performance of network intrusion detection. The overall theme of careful preparation is reinforced by the algorithmic symphony, which iteratively harmonizes labels across categorical columns and makes sure the dataset is not only well-prepared but also in the best possible position to work in harmony with the ML algorithms that come after.

Algorithm 1: Utilized preprocessing techniques.

Input: *X:* cleaned dataset, *Y:* class labels, and *T:* number of iterations
Output: H_T = final set of encoded features

1. **Procedure** PREPROCESSING(X, Y, T)
2. LabelEncoder ← initialize_label_encoder()
3. $X^{(0)}_{\text{encoded}} \leftarrow$ X ◂ Set the initial encoded feature dataset to X
4. $Y^{(0)}_{\text{encoded}} \leftarrow$ LabelEncoder(Y) ◂ Encode class labels Y using the label encoder
5. **For** t = 0: T-1, **do**
6. **For each** feature X_j **in** $X^{(t-1)}_{\text{encoded}}$, **do** ◂ Calculate correlation with the target
7. corr_with_target$^{(t)}[j] \leftarrow$ calculate_correlation($X^{(t-1)}_j$, $Y^{(t-1)}_{\text{encoded}}$)
8. **If** $\left(X^{(t-1)}_{col}[i]\right)$ is known, **then** ◂ Encode attributes
$X^{(t)}_{encoded\_col}[i] \leftarrow LabelEncoder\left(X^{(t-1)}_{col}[i]\right)$
9. Or **else,**
$X^{(t)}_{encoded\_col}[i] \leftarrow LabelEncoder\left(X^{(t-1)}_{col}[i]\right) + C$
10. **If** corr_with_target$^{(t)}[j]$ > threshold, **then**
keep the feature
11. Or **else,**
discard the feature
12. **End for**
13. $X^{(t)}_{\text{encoded}}[i] \leftarrow$ iterative_label_encoder$\left(X^{(t-1)}_{\text{encoded}}[i]\right)$ ◂ Apply Iterative Label Encoding:
14. **End for**
15. **Return** H_T
16. **End procedure**

After carefully examining our dataset, we identified an undetected imbalance in the "class" column, which represents the target variable. Although not overt, it prompted the contemplation of oversampling methodologies; however, the extent did not necessitate substantial intervention. Delving into protocol distribution, TCP constituted roughly 80% of the traffic, UDP accounted for about 12%, and the residual was attributed to ICMP. Intriguingly, a majority of ICMP traffic manifested anomalous patterns, contrasting with the norm for UDP traffic, which predominantly exhibited normal behavior. Within the TCP domain, a relatively equitable distribution prevailed, with no conspicuous prevalence of either normal or anomalous instances. Further exploration into flag-based distribution unveiled an asymmetric pattern. A notable segment featured the SF (Sign Flag), with the

preponderance of such instances categorized as normal. Conversely, traffic characterized by the S_0 flag predominantly signaled anomalous behavior.

3.3. Feature Extraction

One of the most important steps in getting data prepared to handle intrusion detection is feature extraction. Within the given dataset, every TCP/IP connection goes through an extensive feature extraction procedure that produces 41 features in total. The dataset contains two types of features, quantitative and qualitative, with 38 quantitative features, such as byte counts, packet counts, and connection durations, which describe network traffic properties. Additionally, it contains three qualitative features representing categorical variables like protocol types, service types, or TCP/IP connection flags. The integration of both quantitative and qualitative features allows for a comprehensive illustration of the complexities involved in network interactions. The utilization of a hybrid technique for feature extraction enables a comprehensive examination of network activity, taking into account the varied characteristics of both regular and irregular connections [42]. By means of feature extraction, the dataset converts unstructured TCP/IP dump data into a structured representation that encompasses the fundamental elements of network interactions.

3.4. Feature Scaling

In the complex field of network intrusion detection, feature scaling is an advanced feature engineering procedure that is carefully designed to normalize the extracted input features that are used by a machine-learning model that is devoted to intrusion detection. The overall objective is to create a scale of comparability that is harmonic amongst characteristics, with each one acting as a discreet filter that reflects various aspects of network traffic and activity. Let X_{ij}, represent the features distinguishing the network traffic data in this work's mathematical tapestry, where i denotes a particular feature and j denotes a data sample (such as a network connection). The essence of this standardization is captured by the mathematical arrangement of Z-score normalization, as shown by Equation (10):

$$Z_{mn} = \frac{(X_{mn} - \mu_m)}{\sigma_m} \tag{10}$$

where X_{mn} is the initial value of the mth feature for the nth network connection, Z_{mn} is the standardized value (z-score), μ_m is the mean of the mth feature across all network connections, and σ_m is the standard deviation of the mth feature across all network connections. Applying Z-score normalization for network intrusion detection involves the following steps:

1. For every feature m in the training set, calculate the mean μ_m and standard deviation σ_m using the formulas in Equations (11) and (12).

$$\mu_m = \frac{1}{N}\sum_{n=1}^{N} X_{mn} \tag{11}$$

$$\sigma_m = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(X_{mn} - \mu_m)^2} \tag{12}$$

   where N denotes the training set's total number of network connections.
2. Equation (13) represents the standardization of the features in the training set.

$$x_{mn}^{\text{std}} = \frac{x_{mn} - \overline{x_n}}{s_n} \tag{13}$$

3. Apply the same transformation to the features in the test dataset.

In order to ensure homogeneous feature scaling, the mean (μ_m) and standard deviation (σ_m) are utilized to normalize the test set from the training dataset. Algorithm 2 describes the detailed steps involved in completing this vital preprocessing task.

Algorithm 2: Feature scaling.

Input: X: Feature vector; T: Number of iterations
Output: X_T: Scaled feature vector

Procedure FEATURESCALE(X, T)
$\mu_0, \sigma_0 =$ ← calculate_mean_and_std(X) ◃ Initialize mean and standard deviation
X_0 ← X ◃ Initialize input feature vector
For t = 0: T-1, **do**
    r_t ← calculate_gradient(X_t) ◃ Determine the gradient of the feature vector
    $\alpha_t \leftarrow \alpha_0/(1 + t)$ ◃ Calculate the learning rate
    $X_{t+1} \leftarrow X_t - \alpha_t \cdot r_t$ ◃ Update the feature vector
End for
Return X_T
End procedure

Here, the learning rate α_t is set to decrease with each iteration. The starting learning rate is α_0, and as the iterations proceed, the learning rate is guaranteed to decrease according to the decay term $(1 + t)$. This decay facilitates a more efficient convergence of the optimization process. Depending on the details of your issue, you can choose a different learning rate function α_t. Depending on the features of the optimization landscape, other decay schedules or adaptive learning rate approaches can be more appropriate.

3.5. Feature Selection

In order to identify the significance of each feature for the purpose of network intrusion detection, the feature selection approach used in the given approach makes use of a random forest (RF) classifier. After being trained on the scaled training data, the RF classifier gives each feature an important score [43]. How much each attribute helps to decrease impurity throughout the random forest creation process determines the crucial score. Features that result in a notable decrease in impurity are seen to be of greater significance. Gini impurity is frequently used to determine the feature relevance of the RF classifier. The following formula is used to obtain each feature's importance score:

3.5.1. Gini Impurity ($I(t)$)

In the context of decision trees, the idea of Gini impurity presents itself as a complex measure that explores the likelihood that an element chosen at random would be mislabeled [44]. When considering a binary classification problem—where the distribution of labels inside a node determines the domain of randomness—this measure reveals its subtleties. Equation (14) provides a beautiful representation of the mathematical expression that captures this complex measure.

$$I(t) = 1 - \sum_{i=1}^{c} p(i|t)^2 \tag{14}$$

The Gini impurity of node t in the decision tree is personified here by $I(t)$, the number of classes is denoted by c, and the percentage of samples in node t that correspond to class i is indicated by p_i. This formula provides important insights into binary categorization by offering a comprehensive framework for analyzing the complex interactions between label errors and uncertainty in the decision tree model [45].

3.5.2. Feature Importance Score (F_{Imp_i})

Determining the corresponding relevance of features in a random forest (RF) classifier takes careful consideration. Equation (15) determines the significance of features by adding the improvement in the decision tree performance resulting from each feature throughout all trees in the forest.

$$F_{Imp_i} = \frac{\sum_t I(t) \times \text{samples}(t)}{N_{\text{total}}} \quad (15)$$

The importance of feature i in the decision tree splitting process is measured by the significance score (F_{Imp_i}). This score is determined by adding the feature importance increase, weighted by the node's sample size in comparison to the entire dataset (samples(t)/N_{total}) at every node, t, where the feature is employed. Higher values represent more feature importance in decreasing model impurity and increasing predictive accuracy; normalizing these scores ensures a comparable scale. This quantitative measure gives the complex RF model an actual metric to represent the abstract idea of feature relevance. The coordination of feature selection in the context of network intrusion detection is a laborious process, which is succinctly summarized by Algorithm 3. This technique acts as a guidance, pointing the model in the direction of the important elements that are necessary for the intricate process of locating intrusions in network traffic data.

Algorithm 3: Feature selection

Input: X: Features, y: labels; T: number of iterations
Output: S_T = Set of selected features

1. **Procedure FEATURESELECTION(X, y, T)**
2. RF ← random forest classifier($n_{estimators}$ = T) ◃ Initialize random forest classifier
3. $S_0 \leftarrow initialize_{features}$(X) ◃ Initialize the set of selected features
4. **For** t = 0: T-1, **do**
5. RF.fit(X[:, S_t], y) ◃ Train the random forest classifier
6. $importance_{scores} \leftarrow$ RF.$feature_{importances}$ ◃ Compute feature importances
7. $S_{(t+1)} \leftarrow \text{argmax}_k \left(\sum_{i \in S_t} Gini_{\text{Impurity}}(X[:, i], y) \times importance_{\text{scores}}[i] \right)$ ◃ Update selected features
8. **For** i = 1 to k, **do**
9. **if** Gini_Impurity(X[:, S_{t + 1}[i]], y) > threshold, **then**
10. Keep S_{t + 1}[i] in the selected features
11. Or **else**
12. Discard S_{t + 1}[i] from the selected features
13. **End for**
14. **End for**
15. **Return** S_T
16. **End procedure**

Figure 6 visually displays the identifying perspective of the feature importance used in the present study.

3.6. Machine-Learning Classifiers

In this study, a set of seven independent machine-learning classifiers was carefully applied to precisely preprocessed and feature-engineered network intrusion detection datasets [46,47]. The following section discusses numerous ML classifiers, emphasizing their abilities, limitations, and applicability for various intrusion detection scenarios [48].

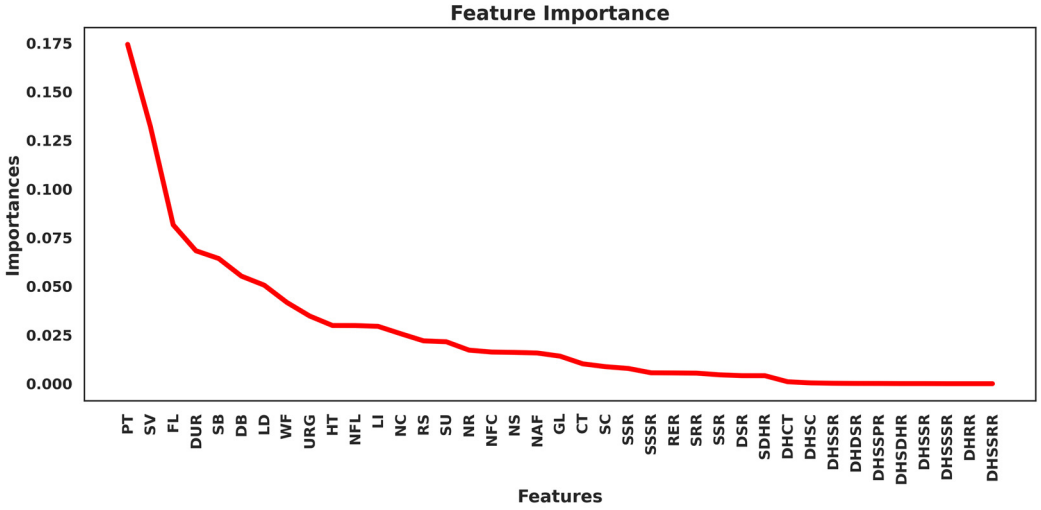


Figure 6. Visualization of feature importance in NIDSs using the proposed approach.

3.6.1. Support Vector Classifier (SVC)

SVC is a strong and adaptable classifier known for its ability to find efficient hyperplanes within feature spaces while successfully navigating both non-linear and linear decision boundaries. SVC's adaptability, enabled by an extensive variety of kernel functions, allows it to handle minute trends throughout data, making it an effective tool in this challenging area [49] Notably, its ability to accommodate a wide range of kernel functions enables SVC to resolve complex interdependencies between features [50].

To mathematically formalize SVC's approach, the optimization problem for finding the optimal hyperplane in a linearly separable case can be expressed as shown in Equation (16):

$$\begin{gathered} \min_{w,b} \frac{1}{2}|w|^2 \\ \text{subject to : } y_i(w \cdot x_i + b) \geq 1, \ \forall i \end{gathered} \tag{16}$$

where w is the weight vector orthogonal to the hyperplane, b is the bias term, x_i are the input feature vectors, and y_i are the class labels ($y_i \in \{-1, 1\}$). In cases where data are not linearly separable, SVC introduces a slack variable ξ_i to allow for some misclassification, leading to the soft-margin optimization expressed by Equation (17):

$$\begin{gathered} \min_{w,b,\xi} \frac{1}{2}|w|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{subject to : } y_i(w \cdot x_i + b) \geq 1 - \xi_i, \ \forall i, \ \xi_i \geq 0 \end{gathered} \tag{17}$$

where C is a regularization parameter controlling the trade-off between margin maximization and classification error. When applied to non-linear decision boundaries, SVC utilizes a kernel function $K(x, x')$ to map input data into a higher-dimensional space. The SVC decision function with kernel trick becomes the following, as expressed in Equation (18):

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + b\right) \tag{18}$$

where α_i are Lagrange multipliers, and $K(x, x_i)$ (e.g., linear, polynomial, radial basis function) captures the complex relationships within the data.

3.6.2. Logistic Regression (LR)

LR, an integral component of statistical modeling, appears as a powerful force in the complex environment of binary classification [51]. Based on the concept of estimating the chance of an instance belonging to a certain class, LR navigates this setting by precisely

modeling the logarithmic distribution of odds for the response variable. Its effectiveness becomes apparent when the association between characteristics and the logarithm of odds follows a roughly linear trajectory [52].

The Logistic Regression model estimates the probability $P(Y = 1|X = x)$ of a binary classification, where Y is the dependent variable and X represents the input features [53]. The relationship between X and the probability is given by the sigmoid function, as defined by Equation (19):

$$P(Y = 1|X = x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p)}} \tag{19}$$

where β_0 is the intercept and $\beta_1, \beta_2, \ldots, \beta_p$ are the coefficients for the predictors $x_1, x_2, \ldots, x_p$. To linearize the relationship, we can use the log-odds (logit) transformation, as given by Equation (20):

$$\text{logit}(P) = \ln\left(\frac{P(Y = 1|X = x)}{1 - P(Y = 1|X = x)}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p \tag{20}$$

Here, each coefficient β_i represents the change in log-odds for a one-unit increase in x_i, holding other variables constant.

The parameters β are estimated by maximizing the likelihood function, as given by Equation (21):

$$L(\beta) = \prod_{i=1}^{n} P(Y_i|X = x_i)^{Y_i} \cdot (1 - P(Y_i|X = x_i))^{1-Y_i} \tag{21}$$

This likelihood maximization results in optimal values for β, which helps to efficiently classify instances within the network intrusion detection context.

3.6.3. Bernoulli Naive Bayes (BNB)

BNB is a successful classification method for handling datasets with binary features. By evaluating the likelihood of an occurrence (such as a network intrusion) by considering the availability or lack of specified features, BNB successfully uses Bayes' theorem [54]. The primary presumption of BNB is featuring independence inside a particular class, which is frequently realistic in network intrusion detection (NID), where features such as "port scan detected" or "unusual traffic pattern" are typically binary indicators [55]. BNB is a prevalent solution for practical intrusion detection systems since it is simple, efficient, and can handle categorical data [56,57].

In BNB, given a binary feature vector $X = (x_1, x_2, \ldots, x_n)$, each feature $x_i \in \{0, 1\}$, and the probability of a class y given X is calculated as per Equation (22):

$$P(y|X) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)^{x_i} \cdot (1 - P(x_i|y))^{1-x_i}}{P(X)} \tag{22}$$

where $P(y)$ is the prior probability of class y, $P(x_i|y)$ is the conditional probability of feature x_i being 1 given class y, and $P(X)$ is the marginal probability of observing X, calculated as $P(X) = \sum_y P(X|y)P(y)$.

The classifier predicts the class $\hat{y}$ as given in Equation (23):

$$\hat{y} = \underset{y}{\text{argmax}} P(y) \prod_{i=1}^{n} P(x_i|y)^{x_i} \cdot (1 - P(x_i|y))^{1-x_i} \tag{23}$$

This approach leverages feature independence, assuming each x_i is conditionally independent given y, which simplifies the computation and makes BNB efficient for binary feature datasets, as commonly found in NID applications.

3.6.4. XGBoost Classifier (XGBC)

XGBoost is an extremely effective ensemble learning approach for improving decision tree performance. XGBoost's capacity to deal with missing data, as well as its amazing speed and accuracy, has strengthened its place as the best option among ML classifiers [58]. Aside from its computational efficiency, XGBoost's ability to process structured data, a popular format for intrusion detection datasets, provides an advantage particularly appropriate to this sector. These features make XGBoost suitable for network intrusion detection [59]. In XGBoost, the objective function (Equation (24)) for a given model $f(x)$ is minimized over the loss function L and regularization term Ω:

$$\text{Objective} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{24}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value for instance i, K is the number of trees, and each f_k is an individual tree. The regularization term $\Omega(f)$ helps prevent overfitting and is defined as shown by Equation (25):

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{25}$$

where T represents the number of leaves in the tree, w_j is the weight associated with each leaf j, γ, and λ are regularization parameters. The model updates in XGBoost iteratively, with the tth tree focusing on minimizing the Equation (26):

$$\text{Loss}_t = \sum_{i=1}^{n} \left(g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right) \tag{26}$$

where $g_i = \frac{\partial L\left(y_i, \widehat{y_i^{(t-1)}}\right)}{\partial \widehat{y_i^{(t-1)}}}$ and $h_i = \frac{\partial^2 L\left(y_i, \widehat{y_i^{(t-1)}}\right)}{\partial \widehat{y_i^{(t-1)^2}}}$ are the first- and second-order gradients of the loss function.

3.6.5. CatBoost Classifier (CatBoost)

CatBoost is a highly effective ML algorithm that is especially useful for dealing with data that contain categorical information, such as text labels [60]. CatBoost's improved ability to interpret categorical features gives it a promising alternative for NID, as datasets usually include a large number of categories. Even without considerable parameter adjustment, the algorithm frequently produces superior outcomes. In CatBoost, the objective function is typically based on gradient boosting over decision trees. The objective function for the tth iteration is defined as given by Equation (26):

$$\text{Objective} = \sum_{i=1}^{n} L\left(y_i, \widehat{y_i^{(t)}}\right) + \Omega\left(f^{(t)}\right) \tag{27}$$

where $L\left(y_i, \widehat{y_i^{(t)}}\right)$ represents the loss function between the actual label, y_i, and predicted value, $\widehat{y_i^{(t)}}$; $\Omega\left(f^{(t)}\right)$ is a regularization term for the tth model to avoid overfitting. The split selection in CatBoost is formulated to maximize the information gain, as defined by Equation (28):

$$\text{Information Gain} = H(P) - \sum_{k=1}^{K} \frac{|P_k|}{|P|} H(P_k) \tag{28}$$

where $H(P)$ is the entropy of the parent node, $H(P_k)$ is the entropy of each child node k, and $|P|$ and $|P_k|$ are the sizes of the parent and child nodes. For each categorical feature value c, the encoded feature $E(c)$ is given by Equation (29):

$$E(c) = \frac{\sum_{j=1}^{N} y_j + \alpha \cdot P}{N + \alpha} \tag{29}$$

where N is the number of occurrences of category c; $\sum_{j=1}^{N} y_j$ is the sum of the target values for c;, P is the global average of the target; and α is a regularization parameter to balance between the global mean and category mean.

3.6.6. LightGBM Classifier (LGBMC)

LGBM is an outstanding performance classifier capable of handling large and complex datasets. This method is very useful for NID, as datasets are frequently multidimensional [58,61]. LGBM emphasizes data points that have the greatest predictive influence, making it good at capturing complex feature correlations. LGBM's ability to handle complex data with speed, precision, and efficiency makes it a valuable tool for network intrusion detection [62]. The objective function to be minimized in LightGBM, like other gradient boosting methods, is expressed by Equation (30):

$$\text{Objective} = \sum_{i=1}^{n} L\left(y_i, \widehat{y_i^{(t)}}\right) + \lambda \sum_{j=1}^{T} |w_j|^2 \tag{30}$$

where $L\left(y_i, \widehat{y_i^{(t)}}\right)$ is the loss function evaluating the error between the true label, y_i, and prediction $\widehat{y_i^{(t)}}$; $\lambda \sum_{j=1}^{T} |w_j|^2$ is the regularization term on tree weights w_j; and T represents the number of trees, helping control overfitting. The information gain ΔH is defined by Equation (31):

$$\Delta H = H(P) - \left(\frac{|P_L|}{|P|} H(P_L) + \frac{|P_R|}{|P|} H(P_R)\right) \tag{31}$$

where $H(P)$ is the entropy of the parent node, $H(P_L)$ *and* $H(P_R)$ are the entropies of the left and right child nodes, and $|P|, |P_L|,$ *and* $|P_R|$ are the respective sizes of the parent, left, and right nodes.

3.6.7. Histogram-Based Gradient Boosting Classifier (HistGradientBoosting)

HistGradientBoosting is a powerful version of gradient boosting classifiers that perform well with large datasets. It utilizes histograms for summarizing data, which greatly accelerates the training performance and requires less memory [58,61,63]. The combined advantages make HistGradientBoosting an efficient and adaptable choice for addressing challenging network intrusion detection problems. For a dataset with n continuous features, the binning process involves dividing each feature x into k bins, where k is significantly smaller than n. The binned feature x_b can be expressed by Equation (32):

$$x_b = \text{bin}(x) = \sum_{j=1}^{k} j \cdot I\left(x \in \text{bin}_j\right) \tag{32}$$

where I is the indicator function that returns to 1 if x belongs to bin j and which is 0 otherwise. The gradient boosting model then uses these binned values to update the model in each iteration t, following the framework of the objective function given by Equation (33):

$$\mathcal{L} = \sum_{i=1}^{N} L\left(y_i, \widehat{y_i^{(t)}}\right) + \lambda \sum_{j=1}^{T} |w_j|^2 \tag{33}$$

where $\mathcal{L}$ is the total loss; $L\left(y_i, \widehat{y_i^{(t)}}\right)$ represents the loss function for the predicted $\widehat{y_i^{(t)}}$ against the true label y_i; and $\lambda\sum_{j=1}^{T}|w_j|^2$ is the regularization term to prevent overfitting.

The updated step for each tree in the boosting process can be defined using the negative gradient as given by Equation (34):

$$g_i^{(t)} = -\frac{\partial L\left(y_i, \widehat{y_i^{(t)}}\right)}{\partial \widehat{y_i^{(t)}}} \tag{34}$$

In HistGradientBoosting, the histograms are computed for these gradients, $g_i^{(t)}$, which allows for efficient split findings across the bins. The optimal split for a feature at bin j can be calculated, as shown by Equation (35):

$$\begin{aligned}\text{Gain}(j) = \tfrac{1}{N}\left(\sum_{i\in L} g_i^{(t)}\right)^2 \Big/ \left(\sum_{i\in L} h_i^{(t)} + \lambda\right) + \tfrac{1}{N}\left(\sum_{i\in R} g_i^{(t)}\right)^2 \Big/ \left(\sum_{i\in R} h_i^{(t)} + \lambda\right) \\ -\tfrac{1}{N}\left(\sum_{i=1}^{N} g_i^{(t)}\right)^2 \Big/ \left(\sum_{i=1}^{N} h_i^{(t)} + \lambda\right)\end{aligned} \tag{35}$$

where L and R refer to the left and right child nodes of the split; $g_i^{(t)}$ and $h_i^{(t)}$ are the first and second derivatives of the loss function.

3.7. Assessment Metrics

A thorough assessment approach is required when analyzing ML models for NID. The confusion matrix is an important tool in evaluating model performance because it quantifies right and wrong classifications [64,65]. In order to present an extensive analysis of the models' capability to correctly detect intrusions throughout network traffic data, this study used five important metrics: accuracy, precision, recall, F1-score, and AUC-ROC [66]. Table 2 contains these measures, and Equations (36)–(40) demonstrate how they are calculated using formulae [67].

Table 2. Utilized performance metrics and corresponding formulae.

Metrics	Formulae	Equations
Accuracy	$Acc(\%) = \frac{(\text{TP}+\text{TN})}{\text{Overall Instances}} \times 100$	(36)
Precision	$\text{Pre}(\%) = \frac{TP}{\text{Correctly Identified Instances}} \times 100$	(37)
Recall	$Sen\ (\%) = \frac{TP}{\text{Overall Positive Instances}} \times 100$	(38)
F1-Score	$\text{F1}(\%) = \frac{2\times\text{Pre}\times\text{Sen}}{\text{Pre}+\text{Sen}} \times 100$	(39)
AUC-ROC	$\text{AUC-ROC} = \int_0^1 \text{ROC}(t)\, dt$	(40)

4. Results

To identify network intrusion, a large dataset of 25,192 TCP/IP records, each with 41 features, was used in the experiment. To improve dataset quality and usability, these strategies included data cleaning, outlier removal, and label encoding to handle categorical variables with correlation analysis to find links between features and the use of unique and iterative label encoding approaches. After the preprocessing phase, pertinent features were extracted and refined from the dataset using a unique feature engineering technique. The objective of this stage was to maximize the information content of the dataset and enable the machine-learning models to classify data more accurately. The dataset was divided into training and test sets in a 70:30 ratio once it had been appropriately prepared. In particular,

17,634 cases were set aside for training, while 7558 instances were kept for assessing the performance of the model.

For each model, a 10-fold cross-validation technique was used to provide reliable model training and validation. This required splitting the training data into ten groups, using the remaining data for training with each subset as a validation set for model assessment. Furthermore, 100 training epochs were carried out for each fold of the cross-validation procedure in order to reduce overfitting and enhance model generalization. An Acer laptop with 16 GB DDR4 RAM, a 500 GB SSD, and NVIDIA GeForce graphics was used for the experimentation. Smooth processing and effective training were made possible by these hardware specs, even while working with big datasets and carrying out demanding 10-fold cross-validation processes. The main tool for developing, training, and validating models was Python programming, which was used together with the Scikit-learn toolkit. Common measures like accuracy, precision, and recall were used to evaluate the performance and provide a thorough understanding of the predictive power and efficacy of the model.

4.1. Training Result

Using our training dataset, we carefully implemented a 10-fold cross-validation method, training our first classifier as a Bernoulli Naive Bayes (BNB or NB). Figure 7 showed us that the NB classifier had made significant success; recall was 0.96 across all folds, precision was 0.88, and accuracy reached an amazing 0.91. These metrics highlight the model's capacity for precise prediction-making, efficient, positive instance identification, and the extraction of pertinent dataset data points.

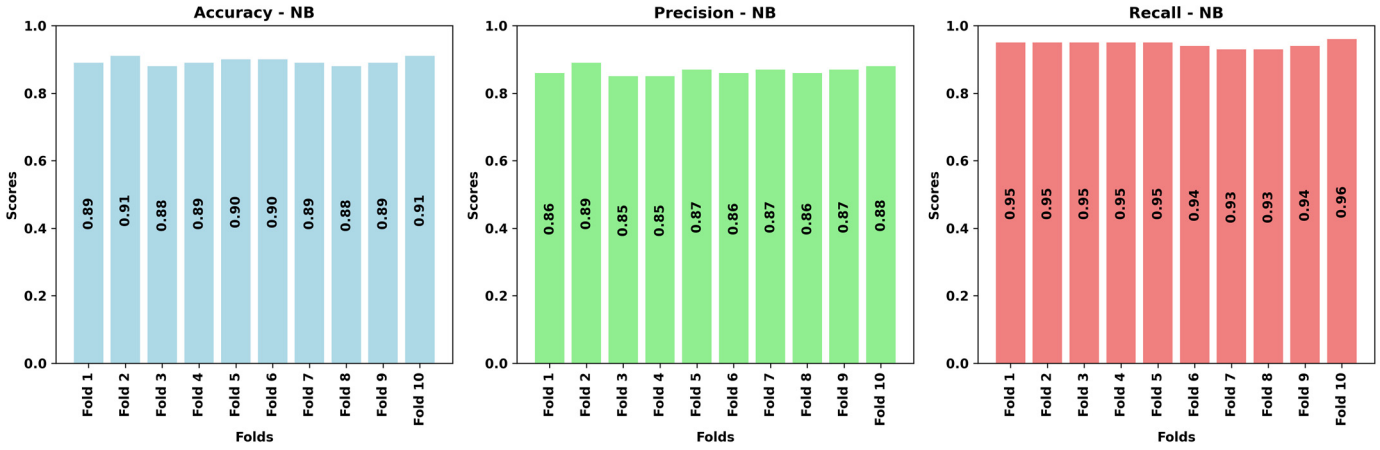


Figure 7. Training performance using 10-fold cross-validation of the NB classifier.

The second model utilized to train the network intrusion dataset is the Logistic Regression (LR) classifier. The LR classifier's performance measures, which include accuracy, precision, and recall, are visually shown for the training history of the model in Figure 8. With the best precision and best recall scores of 0.95 and 0.97, respectively, the best-recorded accuracy was an astounding 0.96. The lowest measures observed, on the other hand, were 0.94 for recall, 0.92 for precision, and 0.93 for accuracy.

In the third phase of our experiment, we trained the SVC using the training data and a 10-fold cross-validation technique. The training results have been carefully represented by bar graphs, as shown in Figure 9. The SVC model performed quite well throughout all folds, with accuracy values varying from 0.98 to 1.0 and a majority of 0.99. Recall scores ranged from 0.97 to 0.99, demonstrating high consistency. However, despite these commendable achievements, there remains ample scope for further enhancement.

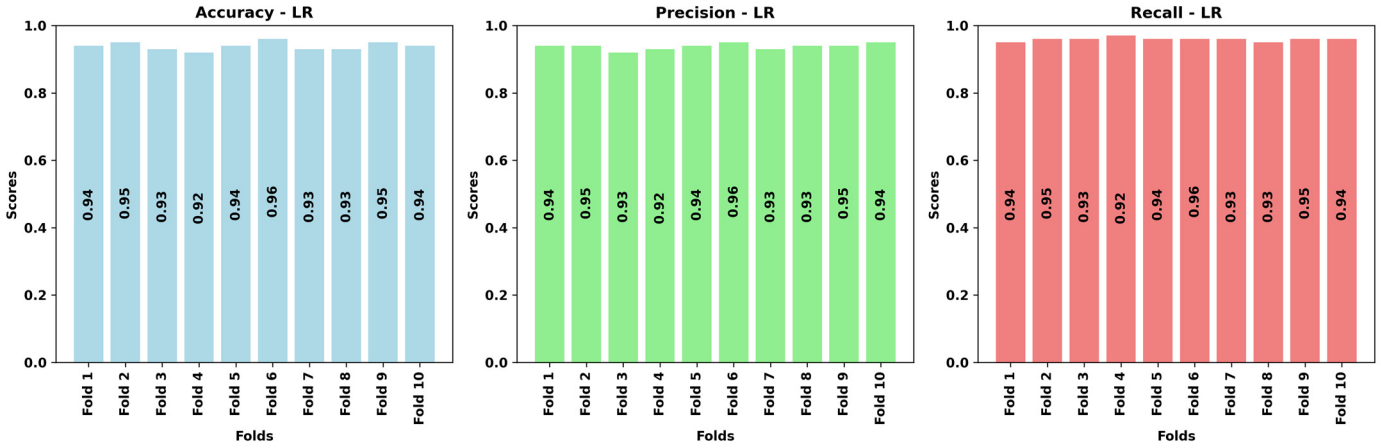


Figure 8. Training performance using 10-fold cross-validation of the LR classifier.

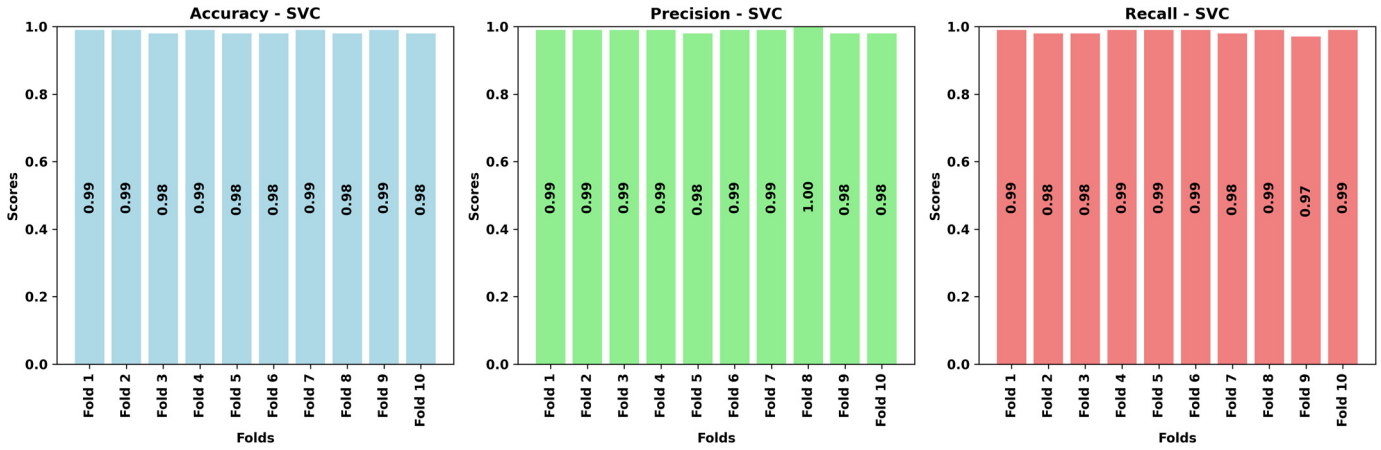


Figure 9. Training performance using 10-fold cross-validation of the SVC classifier.

After training conventional classifiers (NB, LR, and SVC), we experimented with the CatBoostGBM classifier within the Boosting family. CatBoostGBM reached a maximum accuracy of 0.99 using consistent training approaches, while precision and recall reached 0.99 and 1.0, respectively. The lowest accuracy, precision, and recall scores were 0.98, 0.98, and 0.99, respectively (Figure 10). These findings show a minor increase over the SVC classifier, motivating us to investigate other Boosting classifiers for further improvement.

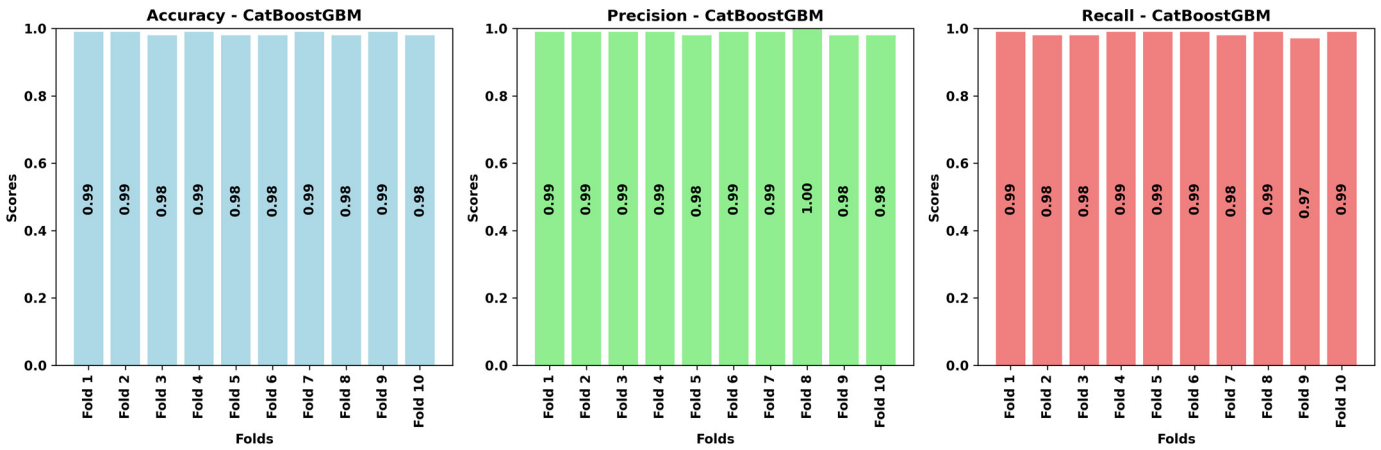


Figure 10. Training performance with 10-fold cross-validation using CatBoost classifier.

In the fifth experimentation phase, we trained the LightGBM classifier using parameters similar to previous experiments. The training history for the 10-fold cross-validation utilizing the LightGBM model is illustrated in Figure 11. Across all ten folds, we achieved accuracy and recall values of 1.0, indicating perfect classification performance. Additionally, the precision values also reached 1.0 in eight out of the ten folds, with only fold one and fold seven achieving precision values of 0.99.

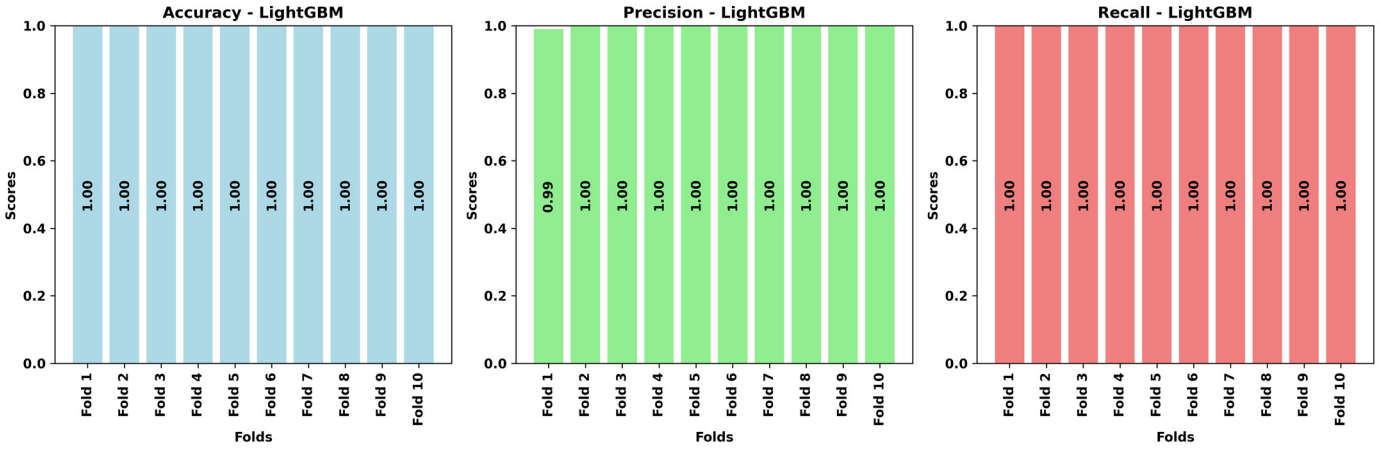


Figure 11. Training performance with 10-fold cross-validation using LightGBM classifier.

Similarly, in the sixth experiment, we employed the HistGradientBoosting classifier on the same dataset and parameters, as depicted in Figure 12. Remarkably, both boosting classifiers, LightGBM and HistGradientBoosting, exhibited outstanding training performance, with negligible variations observed.

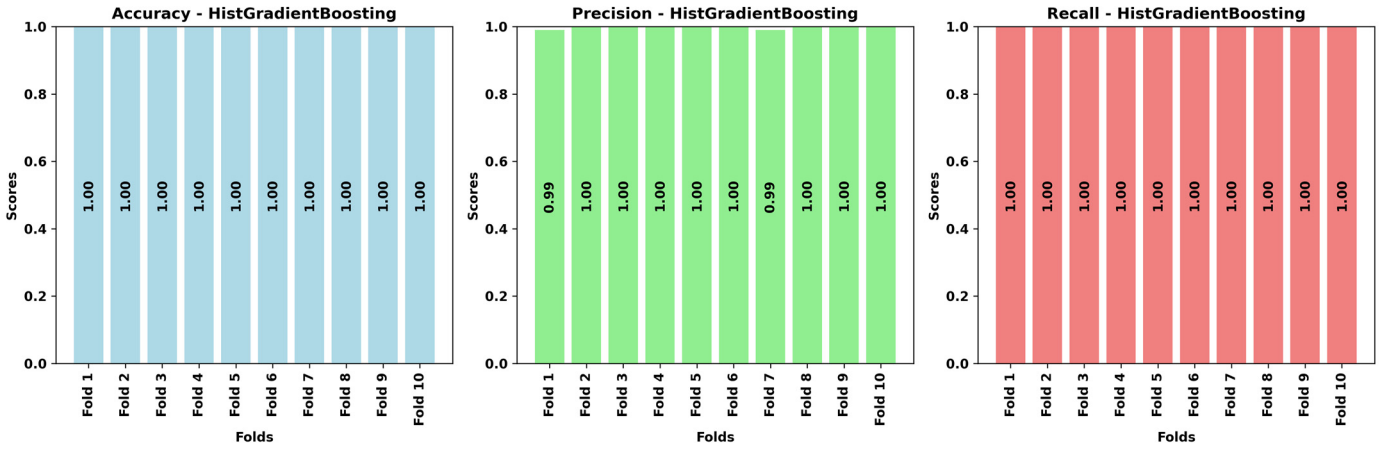


Figure 12. Training performance with 10-fold cross-validation using HistGradientBoosting classifier.

In the final experimentation phase, we utilized the XGBoost classifier, which is another member of the boosting family classifiers. The training history, encompassing accuracy, precision, and recall metrics, is meticulously presented and visualized in Figure 13. Our observations reveal that the XGBoost classifier achieved near-perfect performance across all ten-fold validations for accuracy, precision, and recall metrics. Remarkably, the classifier attained almost perfect scores, with only one instance in fold one's precision history recording a value of 0.99. This remarkable consistency across all metrics underscores the effectiveness of XGBoost in our application for predicting network intrusion detection using machine-learning techniques.

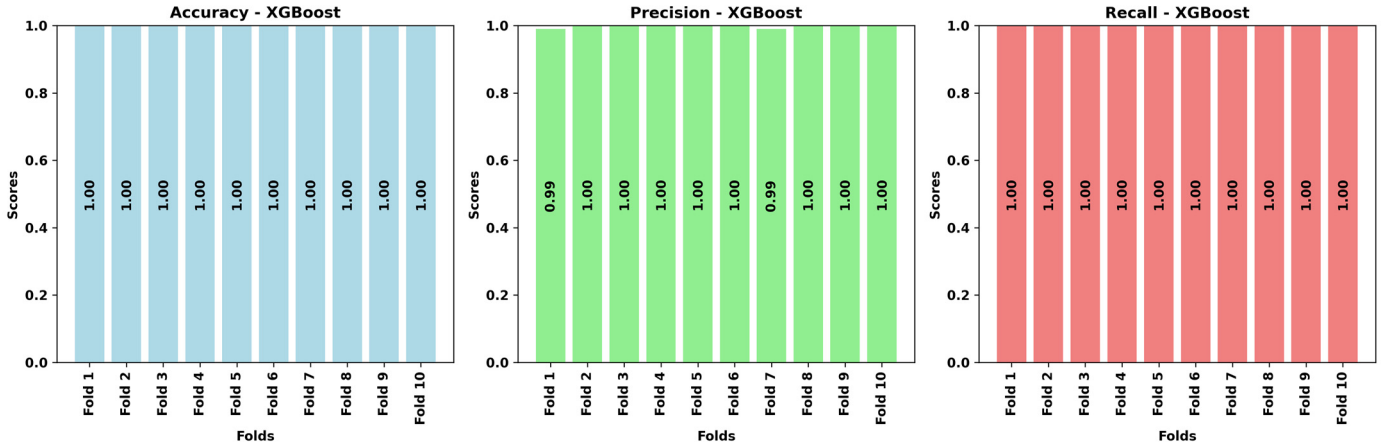


Figure 13. Training performance with 10-fold cross-validation using XGBoost classifier.

4.2. Testing Result

As we move forward, the next crucial step involves validating the performance of these trained models by testing them on independent test data. The test outcome was shown in the form of a confusion matrix on 30% of the test data (7558), including both categories, normal and anomaly. We determined the values of various metrics, including accuracy, precision, recall, and F1 values, along with the standard deviation of each of these metrics, using data from the confusion matrix.

The confusion matrix, as shown in Figure 14a, highlights the capacity of the NB classifier to distinguish between normal network traffic and anomaly. Notably, 3862 anomalies were correctly classified, indicating that it accurately detected a sizable proportion of occurrences. Furthermore, 2978 examples of regular network traffic were correctly classified, demonstrating the classifier's ability to identify common network behavior. The NB classifier did, however, also show instances of incorrect classifications, labeling 180 anomalies as regular traffic and 538 instances of normal network traffic as anomalies.

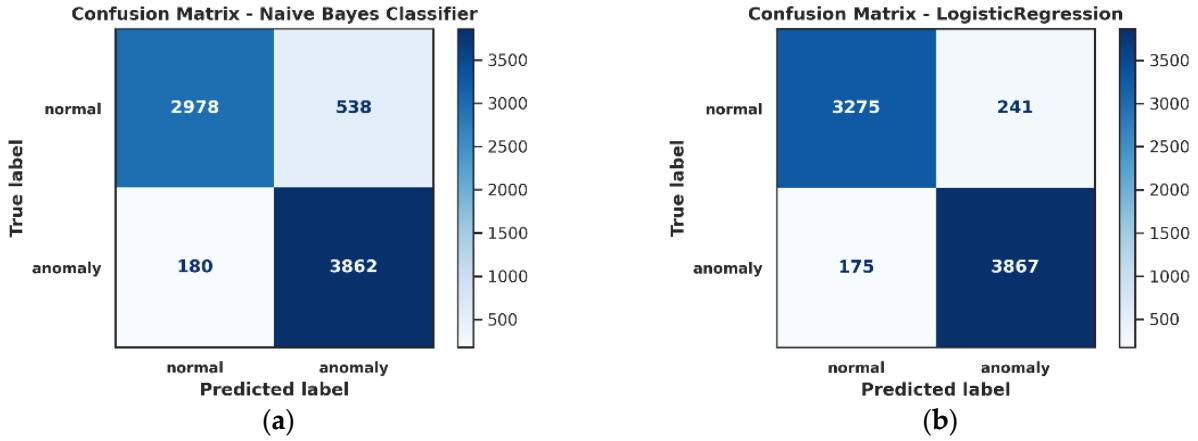


Figure 14. Confusion matrix for testing results: (**a**) NB classifier and (**b**) LR classifier.

The robustness of the Logistic Regression classifier in differentiating between regular and anomalous network traffic within the NIDS test dataset is presented in Figure 14b in the form of a confusion matrix. The LR classifier accurately identified 3867 anomalies and 3275 instances of normal network traffic; however, it incorrectly categorized 241 usual traffic cases as anomalies and 175 anomalies as usual traffic.

The confusion matrix in Figure 15a illustrates the SVC's assessment of the test dataset for NIDS applications. Notably, when compared to earlier models such as Logistic Regression (LR) and Naive Bayes (NB), the SVC demonstrated noteworthy performance.

The SVC's ability to correctly distinguish 3985 instances of anomalies and 3498 cases of regular network traffic was shown by the confusion matrix. The SVC performed well overall despite a few small misclassifications: 18 instances of regular traffic were incorrectly classified as anomalies, whereas 57 anomalies were mistakenly recognized as normal traffic. Compared to the NB and LR models, SVC performed better.

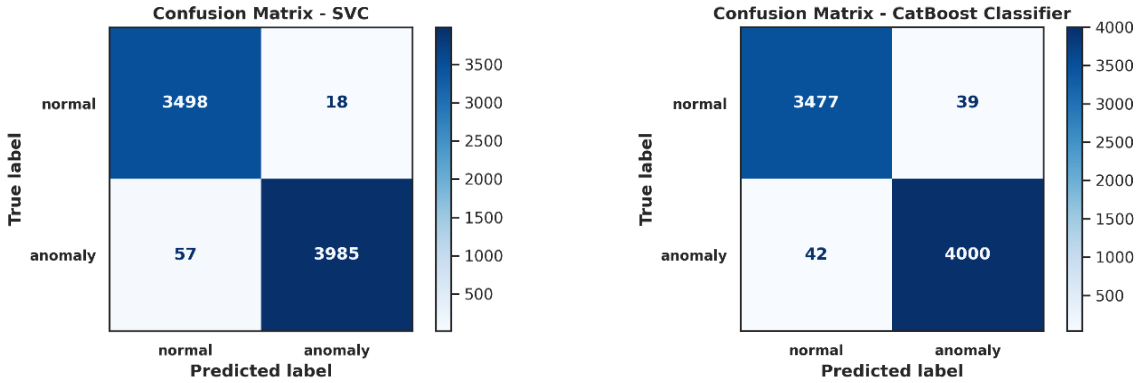


Figure 15. Confusion matrix for testing results: (**a**) SVC classifier and (**b**) CatBoost classifier.

Promising results were obtained from the evaluation of the CatBoostGBM classifier on the test dataset for NIDS applications, as shown by the confusion matrix shown in Figure 15b. In total, 3477 cases of typical network traffic and 4000 cases of abnormalities were correctly categorized by the classifier. Nonetheless, it identified 42 anomalies as regular traffic and 39 instances of normal traffic as anomalies, pointing to a small degree of inaccuracy in the dataset. When the CatBoostGBM classifier is compared to earlier models, such as SVC, NB, and LR, it performs similarly to SVC but better than NB and LR.

The assessment of the LightGBM classifier on the test dataset for NIDS applications demonstrates its promising performance, as seen in the confusion matrix of Figure 16a. In total, 3487 cases of typical network traffic and 4026 instances of abnormalities were correctly categorized by the classifier. However, it misclassified 16 anomalies as regular traffic and 29 instances of normal traffic as anomalies, indicating a low degree of misclassification. When comparing the performance of the LightGBM classifier to earlier models, such as SVC, NB, and LR, it showed better results.

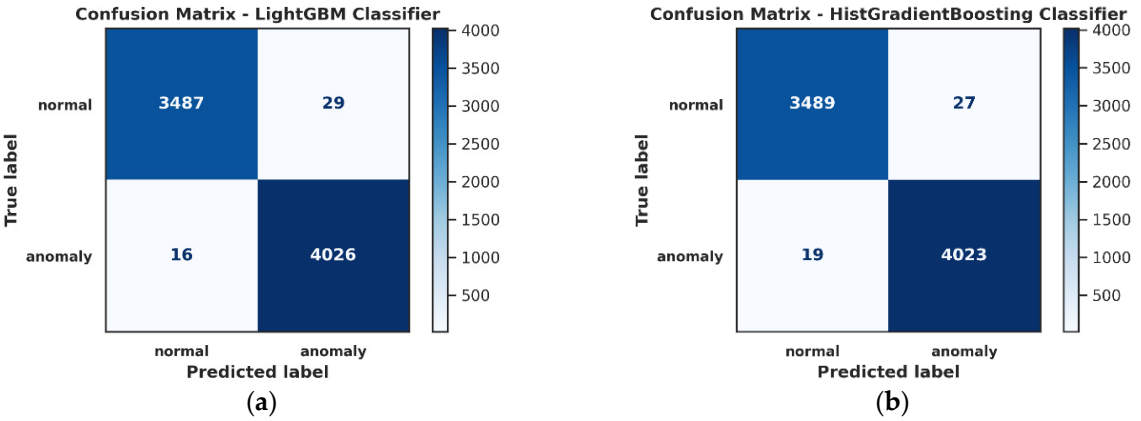


Figure 16. Confusion matrix for testing results: (**a**) LightGBM classifier and (**b**) HistGradientBoosing classifier.

The efficacy of the HistGradientBoosting classifier in NIDS applications is demonstrated by its performance evaluation, which is represented by the confusion matrix shown in Figure 16b. With just 27 instances of normal traffic being misclassified as anomalies and

19 anomalies misclassified as normal traffic, the classifier properly classified 4023 anomaly cases and 3489 instances of regular network traffic, showing a small degree of misclassification. When the HistGradientBoosting classifier is compared against other boosting classifiers like CatBoost and LightGBM, it performs competitively.

The XGBoost classifier was used for the final model testing, and the confusion matrix in Figure 17 presents the results. The XGBoost classifier's confusion matrix demonstrates how effective it is in NIDS applications. In total, 3496 cases of typical network traffic and 4027 occurrences of abnormalities were correctly classified by the classifier. It misclassified a few things: twenty regular traffic cases were incorrectly labeled as anomalies, while fifteen anomalies were mistakenly classified as normal traffic. The XGBoost classifier performs better when compared to other boosting classifiers like CatBoost, LightGBM, and HistGradientBoosting.

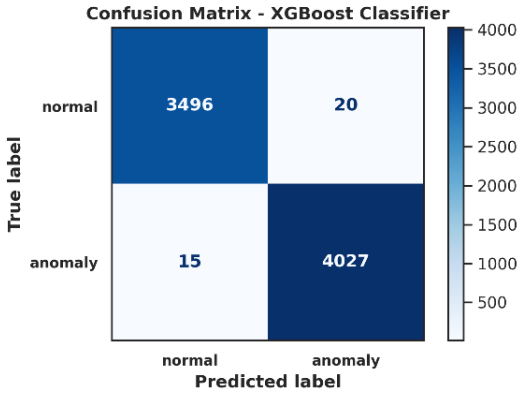


Figure 17. Confusion matrix for testing results with XGBoost classifier.

Table 3 provides an exhaustive evaluation of the NIDS's classifier performance, including accuracy, precision, recall, F1-score, and standard deviation (SD), as calculated using the confusion matrix. The BNB classifier achieves 90.50% accuracy, 90.12% precision, 91.04% recall, and 90.37% F1-score, exhibiting reliability in spite of some prediction uncertainty. In comparison, the LR classifier improved significantly with metrics with a score of 94.5 and an incredibly low SD, indicating its ability to effectively capture complex datasets.

The SVC achieved performance levels above 99.00% in accuracy, precision, recall, and F1-score, making it an exceptional performer. SVC is a great option for NIDS applications because of its exceptional performance and low SD values, which demonstrate its durability and dependability in differentiating between regular network traffic and unusual activity. The results show that the boosting classifiers, CatBoostGBM, LightGBM, and HistGradientBoosting, perform exceptionally well, with the F1-score above 98.90% and accuracy, precision, and recall exceeding 98.90%. These classifiers show little variation in their predictions, which suggests that they can manage complicated datasets well and produce accurate classifications in practical situations. But XGBoost is the best classifier; it outperforms the others with scores higher than 99.50% for accuracy, precision, recall, and F1-score. Because of its small standard deviation values, which demonstrate how consistently and steadily XGBoost makes predictions on various datasets, it continues to be a powerful tool for classification jobs.

Table 3. NIDS detection performance on test datasets across different models and metrics.

Model Validation	Mean Metric (%)		SD
NB	accuracy	90.50	±0.0082
	precision	90.12	±0.0091
	recall	91.04	±0.007
	F1-Score	90.37	±0.0011
LR	accuracy	94.50	±0.0076
	precision	94.41	±0.0085
	recall	94.53	±0.009
	F1-Score	94.46	±0.0004
SVC	accuracy	99.01	±0.0024
	precision	99.04	±0.003
	recall	98.97	±0.0054
	F1-Score	99.00	±0.0021
CatBoostGBM	accuracy	98.93	±0.0029
	precision	98.93	±0.0047
	recall	98.92	±0.003
	F1-Score	98.92	±0.0008
LightGBM	accuracy	99.40	±0.0006
	precision	99.39	±0.0012
	recall	99.41	±0.0008
	F1-Score	99.40	±0.0006
HistGradientBoosting	accuracy	99.39	±0.0008
	precision	99.38	±0.0017
	recall	99.40	±0.001
	F1-Score	99.39	±0.0015
XGBoost	accuracy	99.54	±0.0007
	precision	99.53	±0.0013
	recall	99.54	±0.001
	F1-Score	99.53	±0.0014

In addition to accuracy, precision, recall, and F1-score, we also used the AUC-ROC (Area Under the Receiver Operating Characteristic Curve) statistics to enhance our assessment of the model's capability. As shown in Figure 18, the AUC-ROC analysis demonstrated that SVC, XGBoost, LightGBM, CatBoost, HistGradientBoosting, and AdaBoost all had near to perfect AUC values of 0.999, indicating perfect classification. Logistic Regression was performed in a similar manner with an AUC of 0.987; however, Naive Bayes had a slightly less favorable AUC of 0.953, indicating higher misclassification. For network-based intrusion detection, this demonstrates the boosting models' and SVC's higher performance.

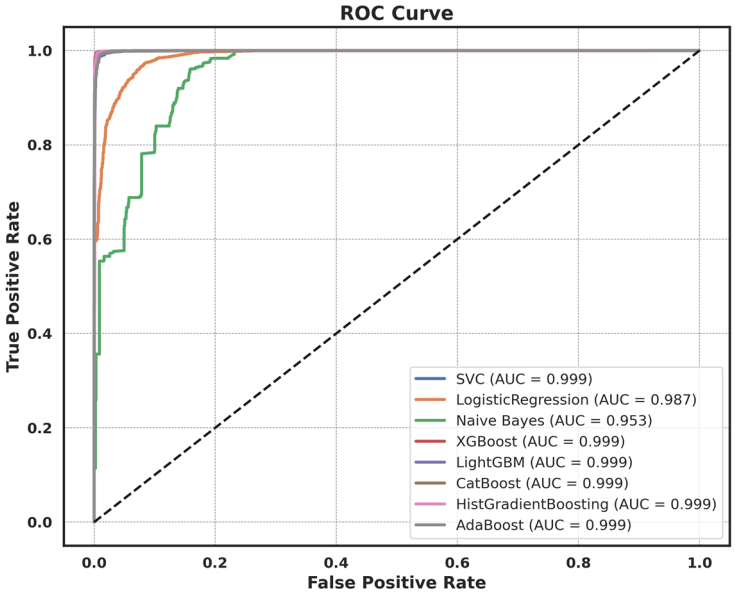


Figure 18. ROC-AUC curves comparing the performance of utilized models.

5. Discussion

To validate the methodology, it is prudent to compare and corroborate the outcomes with recently employed techniques for the NIDS. In the preceding section (Section 3), we conducted a comparative analysis of the results obtained using seven types of classifiers to validate our methodologies. However, to further solidify our findings, we compared our approach with state-of-the-art methods. In order to establish a reliable comparison, we chose a set of methodologies that encompass a variety of sophisticated approaches utilized in the current literature, with a particular emphasis on works published from 2018 to 2024. These techniques were chosen for inclusion in the list due to their popularity and effectiveness in network intrusion detection, as shown by the metrics supplied. In our comparison study, for instance, we used approaches, such as the Deep Neural Network (DNN) and gradient boosting tree (GBT) from Ref [9], which also showed exceptional performance, along with Dugat-LSTM [13], which obtained great accuracy on the TON-IOT and NSL-KDD datasets. These methods span a wide range of approaches, from complex ensemble and deep-learning models to conventional classifiers. By comparing our presented methodologies with these techniques, we provided a comprehensive analysis and discussion of the findings, enhancing the credibility and reliability of our approach.

In order to provide a baseline against which to measure future improvements in performance, we also analyzed baseline methodologies. Because of their established roles in intrusion detection, baseline techniques, including Naive Bayes (NB), Logistic Regression (LR), and Support Vector Classifier (SVC), were chosen as a benchmark for gauging advancements with more complex models. We could demonstrate the relative improvements and efficacy of our suggested technique in comparison to known standards using this approach. In addition to baseline approaches, we compared our proposed method to several advanced techniques, such as the ensemble methods and deep-learning models described in [12,16,24]. In light of current state-of-the-art solutions, this thorough comparison serves to confirm the efficacy of our strategy. Our research, as stated in Table 4, illustrates how our method performs relative to these sophisticated techniques, thus providing a thorough view of its competitive status.

Table 4. Comparative analysis of the proposed technique with state-of-the-art methods for NIDS detection.

Author/Year	Techniques	Datasets	K-Fold Cross-Validation	Metrics (%)
[13]	Dugat-LSTM	TON-IOT, NSL-KDD	×	Acc: 98.76 (TON-IOT) Acc: 99.65% (NSL-KDD)
[19]	SVM, NB, DT, RF	UNSW-NB15	×	Acc: 97.49 Sen: 93.53 Spe: 97.75
[18]	RNN (GRU), MLP	KDD 99, NSL-KDD	10-fold	Acc (KDD): 99.84 Acc (NSL-KDD): 99.24
[12]	NDNN	KDD99, NSL-KDD	×	Acc: 99.90
[9]	DNN, Ensemble, RF, GBT	UNSW NB15, CICIDS2017	5-fold	Acc (DNN): 99.16 Acc (GBT): 99.99
[17]	caXGB, gcForest, GoogLeNetNP	UNSW-NB15 CICIDS2017	×	Acc (UNSW): 99.93 Acc (CICIDS2017): 95.79
[16]	NID-Shield, CAPPER, ML	UNSW-NB15, NSL-KDD	10-fold	Acc (UNSW): 99.89 Acc (KDD): 99.90
[25]	DLA, SDHT, TL, Bootstrapping	UNSW-NB15	×	Acc: 95.60
[11]	SAE + RF	CICIDS2017	×	Acc: 99.90 Sen: 99.90 Spe: 99.89 F1: 99.89
[10]	CNN, BiLSTM, AM, C5.0 DT	KDD CUP 99	×	Acc: 95.50
[24]	Ensemble, SVC, ExtraTree	KDDCupp99	×	Acc: 99.90
[15]	Stacking	CICIDS2017 dataset, CIPMAIDS2023-1	×	F1: 98.24
[23]	PCA + NIDS	NSL-KDD	×	Acc: 97.38
Proposed (2024)	NB, LR, SVC, CatBoost, LightBoost, HistGradientBoosting, **XG Boost**	Private NIDS dataset	**10-fold**	**Acc: 99.54 ± 0.0007** **Pre: 99.53 ± 0.0013** **Rec: 99.54 ± 0.001** **F1: 99.53 ± 0.0014**

The findings from the table highlight that the authors [9] achieved the highest performance in detecting network intrusion. They utilized a range of techniques, including DNN, Ensemble, RF, and GBT, and applied these to datasets such as UNSW NB15 and CICIDS2017. Through 5-fold cross-validation, they reported accuracy rates of 99.99% for GBT and 99.16% for DNN. However, it is important to note that this study solely relied on 5-fold cross-validation and presented accuracy metrics, which may not be the sole criteria to validate model performance. A more thorough evaluation of the model's efficacy in identifying network intrusion might be obtained with the use of additional evaluation metrics and validation methods.

Similarly, the research in Ref [12] showed that the NDNN classifier could detect intrusions with a 99.90% accuracy rate. They used KDD99 and NSL-KDD as the two datasets for their analysis. It is important to note, nevertheless, that they did not use multiple assessment metrics or 10-fold cross-validation in their investigation. These extra validation methods and indicators may offer a more thorough assessment of the classifier's effectiveness in identifying intrusions.

On the other hand, the study by [11] achieved remarkable results utilizing the SAE + RF techniques on the CICIDS2017 dataset, including an accuracy of 99.90%, a sensitivity of 99.90%, a specificity of 99.89%, and an F1-score of 99.89%. They did not use a cross-validation training strategy, which is noteworthy even if their performance was excellent on

all measures. The robustness of their conclusions may be increased using cross-validation, which can also shed light on how well their model generalizes to other datasets or datasets.

A 99.90% accuracy rate for network intrusion detection was found in different research in Ref [24]. KDDCup99 was the dataset they used to test Ensemble, SVC, and ExtraTree classifiers. Like the previously noted research, the authors just provided accuracy as the evaluation metric and trained the models without using a cross-validation technique. Although attaining high accuracy is noteworthy, the lack of cross-validation and dependence just on accuracy measurements can make it more difficult to evaluate their models' effectiveness and generalizability thoroughly.

The research carried out by [9,16,18] is noteworthy for its employment of a fold cross-validation methodology in model training. It is noteworthy that they were able to obtain classification accuracy levels of 99.7%. It is important to note, nevertheless, that this research did not include a variety of indicators to support the efficacy of their models. Presenting alternative assessment measures might offer a more thorough picture of how well the models perform and resilience across various datasets or circumstances, even though accuracy is still a crucial criterion. Thus, a wider set of assessment measures might be included in future studies to help confirm the efficacy of intrusion detection algorithms.

In fact, when compared to other investigations, the study in Ref [10] showed the lowest accuracy at 95.50%. On the KDD CUP 99 dataset, they used CNN, BiLSTM, AM, and C5.0 DT models. The lack of a cross-validation technique and the methods used may have contributed to the decreased accuracy noted in their study, even though they used a variety of deep-learning models. The effectiveness of their models may have been impacted by a number of variables, such as feature engineering methodologies, pre-processing strategies, and dataset selection. Investigating different approaches and streamlining preprocessing stages might be beneficial for future research aimed at increasing the precision of deep-learning-based intrusion detection systems. It is true that the UNSW-NB15 and NSL-KDD databases are often used in a variety of research projects. Because of their extensive availability and global recognition, these datasets are popular choices in intrusion detection research. Because datasets like UNSW-NB15 and NSL-KDD provide a wide variety of network traffic data, which enables reliable model training and assessment, researchers frequently favor them.

However, the CIPMAIDS2023-1 dataset, as used by [15], appears to be less extensively used than UNSW-NB15 and NSL-KDD. There might be various causes for this. The CIPMAIDS2023-1 dataset may be relatively fresh or unknown among the intrusion detection research community. In addition, it may be less accessible or extensive than the other datasets. Furthermore, researchers may be more likely to employ existing datasets with a track record in the area.

To evaluate a private NIDS dataset, we used a range of machine-learning models in our proposed technique, including BNB, LR, SVC, CatBoost, LightBoost, HistGradient-Boosting, and XGBoost. With great care and attention to detail, we were able to obtain these impressive metric values using 10-fold cross-validation: 99.54 ± 0.0007 for accuracy, 99.53 ± 0.0013 for precision, 99.54 ± 0.001 for recall, and 99.53 ± 0.0014 for F1-score. A significant innovation in our approach is the combination of advanced feature engineering techniques—such as random forest-based feature selection (utilizing the Gini coefficient), unique feature scaling, and feature extraction—that optimize the input data for better model performance. The combination of the XGBoost algorithm's strength with this customized feature engineering procedure yielded an intrusion detection method that was more reliable and effective than previous approaches. With excellent levels of accuracy, precision, recall, AUC-ROC, and F1-score on the private NIDS dataset, our proposed methodology showed promising results. A strong and comprehensive intrusion detection technique is highlighted by the thoughtful integration of many classifiers. This comprehensive approach has the potential to significantly improve performance in real-world circumstances, hence increasing the efficacy of intrusion detection systems.

There are certain limitations to our study that should be acknowledged. For starters, we did not use various datasets, which may have offered more information about the generalizability and resilience of our methods in other data environments. Furthermore, our technique was not evaluated on a real-time network intrusion detection system, which might provide useful information on its practical applicability and performance in actual scenarios.

We introduced an innovative approach to network-based intrusion detection by integrating a carefully crafted feature engineering process with the XGBoost algorithm, which is a leading boosting classifier. This approach represents a significant advancement in the field due to the unique combination of our feature engineering methods and the XGBoost algorithm, which has outperformed current algorithms in efficacy for detecting network intrusions.

6. Conclusions

In this study, we employed an innovative approach for network-based intrusion detection by integrating a carefully crafted feature engineering approach with the XGBoost algorithm, which is a leading boosting classifier. This approach represents a significant advancement in the field due to the unique combination of our feature engineering approach and XGBoost algorithm, which outperformed many ML classifiers, including boosting algorithms. We collected 25,192 TCP/IP-based data samples for our dataset, which were mainly divided into two categories: normal and anomalous. To ensure data accuracy and eliminate irrelevant details, we employed various preprocessing techniques, including cleaning, outlier removal, label encoding, correlation analysis, custom label encoding, and iterative label encoding. Following data preprocessing, we applied a unique feature engineering approach to 41 extracted features, incorporating novel feature scaling and random forest-based feature selection techniques. We trained seven types of machine-learning models, including three conventional ML models (BN, LR, and SVC) and four boosting models (CatBoost, LightBoost, HistGradientBoosting, and XGBoost). Each model underwent training on 70% of the dataset using 10-fold cross-validation to optimize training efficacy. Subsequently, we evaluated model performance using four key metrics: accuracy, F1-score, recall, AUC-ROC, and precision. Testing was conducted on the remaining 30% of the dataset, which was not utilized during the training process. The XGBoost classifier, in particular, performed best out of all the models, according to our study, with mean metric values of more than 99.54 for accuracy (Acc), 99.53 for precision (Pre), 99.54 for recall (Rec), and an F1-score of 99.53. We also showcased the ROC curve for model testing, showing that all boosting classifiers achieved an AUC of one. To validate our methodology, we compared our model with state-of-the-art methods. The results highlighted our approach's effectiveness and robustness in the domain of intrusion detection.

In future studies, we plan to improve our technique by including additional and real-time datasets that cover a wide range of vulnerabilities, therefore enriching our understanding of network-based attacks. Concurrently, we intend to incorporate deep-learning models alongside standard classifiers to improve the performance, efficiency, and computational efficiency of our intrusion detection system. Furthermore, we intend to build real-time monitoring tools to allow for rapid detection and reactions to network breaches as they occur. Implementing these upgrades aims to strengthen cybersecurity measures and effectively combat new threats in dynamic network settings.

Author Contributions: Methodology, H.M.R. and S.A.; Conceptualization, H.M.R. and J.Y.; visualization, H.M.R.; writing—original draft preparation, H.M.R., J.Y. and S.A.; validation, J.Y. and S.A.; writing—review and editing, H.M.R., J.Y. and S.A.; supervision, J.Y. and S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no particular support from any public, commercial, or nonprofit source.

Data Availability Statement: The dataset used in this article is publicly available and may be obtained for free at https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection (accessed on 7 February 2024).

Conflicts of Interest: The authors declare that there are no conflicts of interest with this manuscript.

References

1. Rithesh, K. Anomaly-Based NIDS Using Artificial Neural Networks Optimised with Cuckoo Search Optimizer. In *Emerging Research in Electronics, Computer Science and Technology*; Springer: Singapore, 2019; pp. 23–35.
2. Sivasankari., N.; Kamalakkannan., S. Building NIDS for IoT Network Using Ensemble Approach. In Proceedings of the 2022 7th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 22–24 June 2022; pp. 328–333.
3. Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J. Survey of Intrusion Detection Systems: Techniques, Datasets and Challenges. *Cybersecurity* **2019**, *2*, 20. [CrossRef]
4. Sowmya, T.; Mary Anita, E.A. A Comprehensive Review of AI Based Intrusion Detection System. *Meas. Sens.* **2023**, *28*, 100827. [CrossRef]
5. Dini, P.; Elhanashi, A.; Begni, A.; Saponara, S.; Zheng, Q.; Gasmi, K. Overview on Intrusion Detection Systems Design Exploiting Machine Learning for Networking Cybersecurity. *Appl. Sci.* **2023**, *13*, 7507. [CrossRef]
6. Liu, Q.; Hagenmeyer, V.; Keller, H.B. A Review of Rule Learning-Based Intrusion Detection Systems and Their Prospects in Smart Grids. *IEEE Access* **2021**, *9*, 57542–57564. [CrossRef]
7. Vanin, P.; Newe, T.; Dhirani, L.L.; O'Connell, E.; O'Shea, D.; Lee, B.; Rao, M. A Study of Network Intrusion Detection Systems Using Artificial Intelligence/Machine Learning. *Appl. Sci.* **2022**, *12*, 11752. [CrossRef]
8. Gupta, S.; Sharmila; Rai, H.M. IoT-Based Automatic Irrigation System Using Robotic Vehicle. In *Information Management and Machine Intelligence. ICIMMI 2019. Algorithms for Intelligent Systems*; Goyal, D., Bălaş, V.E., Mukherjee, A., de Albuquerque, V.H.C., Gupta, A.K., Eds.; Springer: Singapore, 2021; pp. 669–677, ISBN 978-981-15-4936-6.
9. Faker, O.; Dogdu, E. Intrusion Detection Using Big Data and Deep Learning Techniques. In Proceedings of the ACMSE 2019—Proceedings of the 2019 ACM Southeast Conference, Kennesaw, GA, USA, 18–20 April 2019; Association for Computing Machinery, Inc.: New York, NY, USA, 2019; pp. 86–93.
10. Gao, J. Network Intrusion Detection Method Combining CNN and BiLSTM in Cloud Computing Environment. *Comput. Intell. Neurosci.* **2022**, *2022*, 7272479. [CrossRef]
11. Zhang, C.; Chen, Y.; Meng, Y.; Ruan, F.; Chen, R.; Li, Y.; Yang, Y. A Novel Framework Design of Network Intrusion Detection Based on Machine Learning Techniques. *Secur. Commun. Netw.* **2021**, *2021*, 6610675. [CrossRef]
12. Jia, Y.; Wang, M.; Wang, Y. Network Intrusion Detection Algorithm Based on Deep Neural Network. *IET Inf. Secur.* **2019**, *13*, 48–53. [CrossRef]
13. Devendiran, R.; Turukmane, A. V Dugat-LSTM: Deep Learning Based Network Intrusion Detection System Using Chaotic Optimization Strategy. *Expert Syst. Appl.* **2024**, *245*, 123027. [CrossRef]
14. Ren, K.; Zeng, Y.; Cao, Z.; Zhang, Y. ID-RDRL: A Deep Reinforcement Learning-Based Feature Selection Intrusion Detection Model. *Sci. Rep.* **2022**, *12*, 15370. [CrossRef]
15. Ali, M.; ul Haque, M.; Durad, M.H.; Usman, A.; Mohsin, S.M.; Mujlid, H.; Maple, C. Effective Network Intrusion Detection Using Stacking-Based Ensemble Approach. *Int. J. Inf. Secur.* **2023**, *22*, 1781–1798. [CrossRef]
16. Rincy N, T.; Gupta, R. Design and Development of an Efficient Network Intrusion Detection System Using Machine Learning Techniques. *Wirel. Commun. Mob. Comput.* **2021**, *2021*, 9974270. [CrossRef]
17. Zhang, X.; Chen, J.; Zhou, Y.; Han, L.; Lin, J. A Multiple-Layer Representation Learning Model for Network-Based Attack Detection. *IEEE Access* **2019**, *7*, 91992–92008. [CrossRef]
18. Xu, C.; Shen, J.; Du, X.; Zhang, F. An Intrusion Detection System Using a Deep Neural Network with Gated Recurrent Units. *IEEE Access* **2018**, *6*, 48697–48707. [CrossRef]
19. Belouch, M.; El Hadaj, S.; Idhammad, M. Performance Evaluation of Intrusion Detection Based on Machine Learning Using Apache Spark. *Procedia Comput. Sci.* **2018**, *127*, 1–6. [CrossRef]
20. Carta, S.; Podda, A.S.; Recupero, D.R.; Saia, R. A Local Feature Engineering Strategy to Improve Network Anomaly Detection. *Future Internet* **2020**, *12*, 177. [CrossRef]
21. Swarna Priya, R.M.; Maddikunta, P.K.R.; Parimala, M.; Koppu, S.; Gadekallu, T.R.; Chowdhary, C.L.; Alazab, M. An Effective Feature Engineering for DNN Using Hybrid PCA-GWO for Intrusion Detection in IoMT Architecture. *Comput. Commun.* **2020**, *160*, 139–149. [CrossRef]
22. Saia, R.; Carta, S.; Recupero, D.; Fenu, G.; Stanciu, M. A Discretized Extended Feature Space (DEFS) Model to Improve the Anomaly Detection Performance in Network Intrusion Detection Systems. In Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Vienna, Austria, 17–19 September 2019; SCITEPRESS—Science and Technology Publications: Setubal, Portugal, 2019; pp. 322–329.
23. Babu, B.S.; Reddy, G.A.; Goud, D.K.; Naveen, K.; Reddy, K.S.T. Network Intrusion Detection Using Machine Learning Algorithms. In Proceedings of the 2023 3rd International Conference on Smart Data Intelligence, ICSMDI 2023, Trichy, India, 30–31 March 2023; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2023; pp. 367–371.

24. Bhati, N.S.; Khari, M. A New Ensemble Based Approach for Intrusion Detection System Using Voting. *J. Intell. Fuzzy Syst.* **2022**, *42*, 969–979. [CrossRef]
25. Ashiku, L.; Dagli, C. Network Intrusion Detection System Using Deep Learning. *Procedia Comput. Sci.* **2021**, *185*, 239–247. [CrossRef]
26. Anderson, J.P. *Computer Security Threat Monitoring and Surveillance*; ManageEngine Endpoint Central: Washington, MD, USA, 1980.
27. Denning, D.E. An Intrusion-Detection Model. *IEEE Trans. Softw. Eng.* **1987**, *SE-13*, 222–232. [CrossRef]
28. Roesch, M. Snort—Lightweight Intrusion Detection for Networks. In Proceedings of the 13th USENIX Conference on System Administration, Seatlle, WA, USA, 7–12 November 1999; USENIX Association: Boston, MA, USA, 1999; pp. 229–238.
29. Axelsson, S. *Intrusion Detection Systems: A Survey and Taxonomy*; ResearchGate: Berlin, Germany, 2000.
30. Bhuyan, M.H.; Bhattacharyya, D.K.; Kalita, J.K. Network Anomaly Detection: Methods, Systems and Tools. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 303–336. [CrossRef]
31. Buczak, A.L.; Guven, E. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1153–1176. [CrossRef]
32. Sommer, R.; Paxson, V. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 16–19 May 2010; pp. 305–316.
33. Bensaoud, A.; Kalita, J.; Bensaoud, M. A Survey of Malware Detection Using Deep Learning. *Mach. Learn. Appl.* **2024**, *16*, 100546. [CrossRef]
34. Musa, T.H.A.; Bouras, A. Anomaly Detection: A Survey. In *Lecture Notes in Networks and Systems*; ACM: New York, NY, USA, 2022; pp. 391–401.
35. San Carlos, C. Surge in Cybercrime: Check Point 2023 Mid-Year Security Report Reveals 48 Ransomware Groups Have Breached over 2200 Victims. Available online: https://www.checkpoint.com/press-releases/surge-in-cybercrime-check-point-2023-mid-year-security-report-reveals-8-spike-in-global-cyberattacks/ (accessed on 16 September 2024).
36. Network Intrusion Detection (NIDS) Dataset. Available online: https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection (accessed on 12 September 2023).
37. Shiomoto, K. Network Intrusion Detection System Based on an Adversarial Auto-Encoder with Few Labeled Training Samples. *J. Netw. Syst. Manag.* **2023**, *31*, 5. [CrossRef]
38. Senthilnathan, S. Usefulness of Correlation Analysis. *SSRN Electron. J.* **2019**. [CrossRef]
39. Miot, H.A. Correlation Analysis in Clinical and Experimental Studies. *J. Vasc. Bras.* **2018**, *17*, 275–279. [CrossRef] [PubMed]
40. Hancock, J.T.; Khoshgoftaar, T.M. Survey on Categorical Data for Neural Networks. *J. Big Data* **2020**, *7*, 28. [CrossRef]
41. Shah, D.; Xue, Z.Y.; Aamodt, T.M. Label Encoding for Regression Networks. *arXiv* **2022**, arXiv:2212.01927.
42. Wang, X.; Zhang, X.; Gong, H.; Jiang, J.; Rai, H.M. A Flight Control Method for Unmanned Aerial Vehicles Based on Vibration Suppression. *IET Collab. Intell. Manuf.* **2021**, *3*, 252–261. [CrossRef]
43. Yang, T.; Song, J.; Li, L. A Deep Learning Model Integrating SK-TPCNN and Random Forests for Brain Tumor Segmentation in MRI. *Biocybern. Biomed. Eng.* **2019**, *39*, 613–623. [CrossRef]
44. Disha, R.A.; Waheed, S. Performance Analysis of Machine Learning Models for Intrusion Detection System Using Gini Impurity-Based Weighted Random Forest (GIWRF) Feature Selection Technique. *Cybersecurity* **2022**, *5*, 1. [CrossRef]
45. Yuan, Y.; Wu, L.; Zhang, X. Gini-Impurity Index Analysis. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 3154–3169. [CrossRef]
46. Moqurrab, S.A.; Rai, H.M.; Yoo, J. HRIDM: Hybrid Residual/Inception-Based Deeper Model for Arrhythmia Detection from Large Sets of 12-Lead ECG Recordings. *Algorithms* **2024**, *17*, 364. [CrossRef]
47. Rai, H.M.; Yoo, J.; Dashkevych, S. GAN-SkipNet: A Solution for Data Imbalance in Cardiac Arrhythmia Detection Using Electrocardiogram Signals from a Benchmark Dataset. *Mathematics* **2024**, *12*, 2693. [CrossRef]
48. Hildebrand, B.; Ghimire, A.; Amsaad, F.; Razaque, A.; Mohanty, S.P. Quantum Communication Networks: Design, Reliability, and Security. *IEEE Potentials* **2024**, 2–8. [CrossRef]
49. Zhi, J.; Sun, J.; Wang, Z.; Ding, W. Support Vector Machine Classifier for Prediction of the Metastasis of Colorectal Cancer. *Int. J. Mol. Med.* **2018**, *41*, 1419–1426. [CrossRef] [PubMed]
50. Nguyen, M.T.; Shahzad, A.; Van Nguyen, B.; Kim, K. Diagnosis of Shockable Rhythms for Automated External Defibrillators Using a Reliable Support Vector Machine Classifier. *Biomed. Signal Process Control* **2018**, *44*, 258–269. [CrossRef]
51. Maalouf, M. Logistic Regression in Data Analysis: An Overview. *Int. J. Data Anal. Tech. Strateg.* **2011**, *3*, 281. [CrossRef]
52. Yang, Z.; Li, D. Application of Logistic Regression with Filter in Data Classification. In Proceedings of the 2019 Chinese Control Conference (CCC), Guangzhou, China, 27–30 July 2019; pp. 3755–3759.
53. Jha, K.; Pasbola, M.; Rai, H.M.; Amanzholova, S. Utilizing Smartwatches and Deep Learning Models for Enhanced Avalanche Victim Identification, Localization, and Efficient Recovery Strategies: An In-Depth Study. In Proceedings of the 5th International Conference on Information Management & Machine Intelligence, Jaipur, India, 23–25 November 2023; ACM: New York, NY, USA, 2023; pp. 1–5.
54. Salau, A.O.; Assegie, T.A.; Akindadelo, A.T.; Eneh, J.N. Evaluation of Bernoulli Naive Bayes Model for Detection of Distributed Denial of Service Attacks. *Bull. Electr. Eng. Inform.* **2023**, *12*, 1203–1208. [CrossRef]
55. Panigrahi, R.; Kuanar, S.K.; Kumar, L. Application of Naïve Bayes Classifiers for Refactoring Prediction at the Method Level. In Proceedings of the 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), Gunupur, India, 13–14 March 2020; pp. 1–6.

56. Goyal, Y.; Rai, H.M.; Aggarwal, M.; Saxena, K.; Amanzholova, S. Revolutionizing Skin Cancer Detection: A Comprehensive Review of Deep Learning Methods. In Proceedings of the 5th International Conference on Information Management & Machine Intelligence, Jaipur, India, 23–25 November 2023; ACM: New York, NY, USA, 2023; pp. 1–6.
57. Kim, H.J.; Lim, J.S. Study on a Biometric Authentication Model Based on ECG Using a Fuzzy Neural Network. *IOP Conf. Ser. Mater. Sci. Eng.* **2018**, *317*, 012030. [CrossRef]
58. Almufti, S.M. EXtreme Gradient Boosting Algorithm with Machine Learning: A Review. *Acad. J. Nawroz Univ.* **2023**, *12*, 320–334. [CrossRef]
59. Vo, D.M.; Nguyen, N.Q.; Lee, S.W. Classification of Breast Cancer Histology Images Using Incremental Boosting Convolution Networks. *Inf. Sci.* **2019**, *482*, 123–138. [CrossRef]
60. Hancock, J.T.; Khoshgoftaar, T.M. CatBoost for Big Data: An Interdisciplinary Review. *J. Big Data* **2020**, *7*, 94. [CrossRef] [PubMed]
61. Huang, B.; Wang, C. Research on Data Analysis of Efficient Innovation and Entrepreneurship Practice Teaching Based on LightGBM Classification Algorithm. *Int. J. Comput. Intell. Syst.* **2023**, *16*, 145. [CrossRef]
62. Khan, M.B.S.; Atta-Ur-Rahman; Nawaz, M.S.; Ahmed, R.; Khan, M.A.; Mosavi, A. Intelligent Breast Cancer Diagnostic System Empowered by Deep Extreme Gradient Descent Optimization. *Math. Biosci. Eng.* **2022**, *19*, 7978–8002. [CrossRef]
63. Nhat-Duc, H.; Van-Duc, T. Comparison of Histogram-Based Gradient Boosting Classification Machine, Random Forest, and Deep Convolutional Neural Network for Pavement Raveling Severity Classification. *Autom. Constr.* **2023**, *148*, 104767. [CrossRef]
64. Doreswamy; Hemanth, K.S. Performance Evaluation of Predictive Engineering Materials Data Sets. *Artif. Intell. Syst. Mach. Learn.* **2011**, *3*, 1–8.
65. Ebrahimian, S.; Singh, R.; Netaji, A.; Madhusudhan, K.S.; Homayounieh, F.; Primak, A.; Lades, F.; Saini, S.; Kalra, M.K.; Sharma, S. Characterization of Benign and Malignant Pancreatic Lesions with DECT Quantitative Metrics and Radiomics. *Acad. Radiol.* **2022**, *29*, 705–713. [CrossRef]
66. Mehmood, M.; Abbasi, S.H.; Aurangzeb, K.; Majeed, M.F.; Anwar, M.S.; Alhussein, M. A Classifier Model for Prostate Cancer Diagnosis Using CNNs and Transfer Learning with Multi-Parametric MRI. *Front. Oncol.* **2023**, *13*, 1225490. [CrossRef]
67. Ma, S.; Chen, J.; Zhang, Y.; Shrivastava, A.; Mohan, H. Cloud Based Resource Scheduling Methodology for Data-Intensive Smart Cities and Industrial Applications. *Scalable Comput. Pract. Exp.* **2021**, *22*, 227–235. [CrossRef]