International Journal of Basic and Applied Sciences, 14 (3) (2025) 286-297

International Journal of Basic and Applied Sciences

Website: www.sciencepubco.com/index.php/IJBAS
https://doi.org/10.14419/6m5nbj91
Research paper

Development of A Machine Learning-Based Security Module for Detecting Exploit-Type Attacks in IoT Networks

P. Bavithra Matharasi [1] *, M. Chennakesavulu [2], Manjula Prabakaran S. [3], Dheeraj Akula [4], S. Lakshminarasimhan [5], Ravindra Namdeorao Jogekar [6], Naveen Mukkapati [7], Tatiraju V. Rajani Kanth [8]

[1] Department of Computer Science (MCA), Mount Carmel College, Autonomous, Bengaluru, Karnataka 560052, India
[2] Department of Electronics and Communication Engineering, Rajeev Gandhi Memorial College of Engineering and Technology (Autono-mous), Nandyal, Andhra Pradesh 518501, India
[3] Department of Computer Science and Engineering (Data Science)، Madanapalle Institute of Technology & Science, Madanapalle, Andhra Pradesh 517325, India
[4] Department of North America Applications, Oracle Corporation, Austin, Texas 78741, USA
[5] Department of Artificial Intelligence and Data Science, J.J. College of Engineering and Technology, Tiruchirappalli, Tamil Nadu 620009, India
[6] Department of Computer Science and Engineering, Jhulelal Institute of Technology, Nagpur, Maharashtra 441111, India
[7] Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Andhra Pradesh 522502, India
[8] TVR Consulting Services Private Limited, Hyderabad, Telangana 500055, India
**Corresponding author E-mail: p.bavithra.matharasi@mccblr.edu.in*

Received: June 9, 2025, Accepted: July 19, 2025, Published: July 25, 2025

Abstract

This study investigated the effectiveness of a two-layer neural network (NN) model in detecting malicious-based cyberattacks. The model showed strong classification performance with 0.92 precision, 0.965 recall, and F1 score of 0.92. The SMOTE (Synthetic Minority Over-sampling Technique) method was used to solve the class imbalance problem, which greatly improved the prediction ability of the model. In addition, OneHotEncoding was used to convert categorical variables into binary format, which further improved the accuracy of the model. The results demonstrate the potential of deep learning methods, especially NNs, for cybersecurity tasks, demonstrating their ability to accurately detect complex and deep patterns of malicious usage.

Keywords: *Machine Learning; IoT Networks; Security Module; Exploit-Type Attacks.*

1. Introduction

We live in an era where our dependence on technology is increasing, ranging from the optimization of crops such as cannabis through the use of IoT technologies to the adoption of driving assistants and virtual assistants in our homes [1-2]. Therefore, it is essential to be able to identify cyberattacks early, understand where and how security breaches occur, and know how to mitigate them. Cybercriminals often seek to steal or hijack information, which is one of the most valuable assets for companies [3-4]. There are significant cases of recent attacks, such as the one suffered by the Universidad del Bosque, where the integrity of its academic community's data was compromised, generating great uncertainty. Likewise, the Colombian Civil Aeronautics and DANE suffered attacks in 2021 that compromised their internal services, communications, and databases containing sensitive and confidential information [5-7].

The global pandemic forced thousands of workers to adopt a remote work format, which has increased the exposure of personal and business information on personal devices. This change has presented cybercriminals with new opportunities to access and exploit this information. It is essential to improve our detection and response systems to these threats to more quickly and effectively safeguard user information and devices.

It is vitally important to consider cases like that of DANE, where timely detection of suspicious activities could have prevented the theft and/or loss of valuable information [8-10]. Similarly, frequent attacks on government websites during election seasons, aimed at sowing uncertainty and disrupting citizens, as well as capturing and altering election results, are a direct threat to the stability of governments and confidence in their institutions.

2. Literature review

Al-Amiedi et al. (2023) [11] conducted a systematic literature review on attacks and defense mechanisms in RPL-based 6LoWPAN networks, which are widely used in IoT infrastructure. In this study, attacks were classified into three levels: topology-based, resource-based, and traffic-based. Various mitigation strategies were evaluated, including trust-based, cryptographic, and intrusion detection methods. The authors highlighted the need for lightweight, energy-efficient solutions due to the limited processing power of 6LoWPAN devices.
Sodhro et al. (2022) [12] focused on smart authentication mechanisms in 5G-enabled healthcare IoT environments. The paper reviewed machine learning and blockchain-based techniques to secure patient data and communication channels. The authors noted that traditional approaches are inadequate for dynamic and confidential healthcare environments and advocated a hybrid AI approach that can provide robust adaptive security.
Nova (2022) [13] studied cybersecurity issues in smart cities and emphasized the importance of cyberthreat intelligence (CTI). The study examined how threat detection, situational awareness, and automated response systems improve the resilience of cities. Nova believes that combining ACT with smart infrastructure planning can significantly improve cities' ability to respond to and recover from cyber incidents.
Elattar et al. (2024) [14] proposed an explanatory AI model for PDF-based malware detection (PDFMal) using a gradient boosting model. Consider seeking better equity by maintaining transparency in decision-making. This is particularly important in the field of cybersecurity, where explainability is critical to building trust and accountability in AI-powered systems.
Wei et al. (2023) [15] proposed a deep learning framework for IoT edge botnet detection. The model strikes a balance between computational efficiency and accuracy, allowing real-time risk detection in a given situation. The framework supports early detection and response, which is critical for minimizing the impact of botnet attacks.
Salunkhe et al. (2024) [16] explored advanced cryptographic techniques to secure connected medical devices. The study considered homomorphic encryption, quantum-resistant encryption, and weak protocols. The authors highlight the importance of maintaining data privacy and integrity, minimizing computational cost, and using implanted medical devices.
RobEns continues to advance the field of IoT security, bridging the gap between ensemble-based intrusion detection systems (IDS) and adversarial robustness. Its innovation lies in combining a two-layer protection mechanism with ensemble learning to evaluate realistic attacks, achieving superior detection performance even under adversarial attacks and favorable conditions. This makes RobEns a key player in protecting resource-constrained IoT environments [17].
The reviewed literature collectively emphasizes the increasing complexity of IoT cyber threats and the need for mature, vulnerable, and predictive defense mechanisms. Advances in artificial intelligence, encryption, and computer networking and infrastructure (CTI) offer promising solutions, but balancing security and performance remains a challenge, especially in real-time applications such as healthcare and smart cities.

3. Methodology

Figure 1 below presents a diagram showing the methodology used for the development and evaluation of each of the machine learning models.

- Initiation: The starting point of the process.
- Understanding the dataset: This refers to becoming familiar with the data, understanding its structure, content, and the relationships between features.
- Data profiling and cleaning: After understanding the data, the next step is to preprocess it. This includes cleaning the data by removing or correcting outliers or missing values, and profiling the data to better understand its characteristics and distributions.
- Feature selection: Selecting the most important features to use to train the model. This may involve removing redundant or irrelevant features to improve the model's efficiency and possibly its performance..

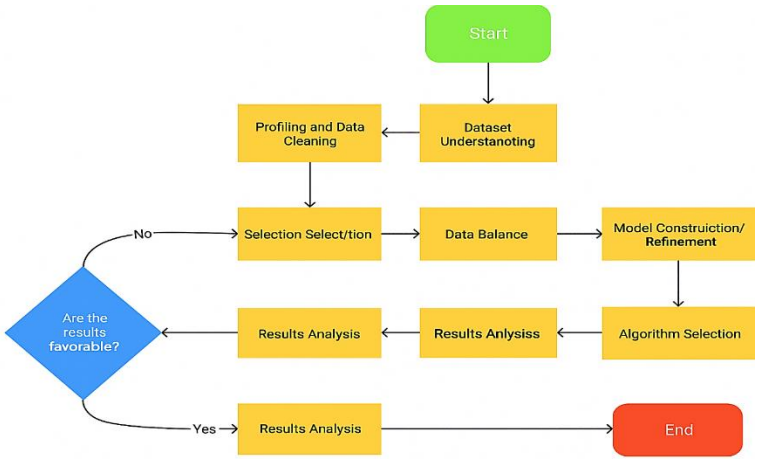

Fig. 1: Methodology for Developing and Evaluating Machine Learning Models.

- Data balancing: If the dataset is unbalanced, techniques are applied to balance the data and avoid bias in the model.
- Model building/refining: Create or improve the machine learning model with the selected features and balanced data.
- Algorithm training: Train the model with the already processed data.
- Results analysis: Evaluate model performance using metrics such as recall, precision, etc. If the results are not satisfactory, return to previous steps to adjust or change the approach.
- Decision: A decision based on the analysis of the results. If they are not favorable, you can return to feature selection to make adjustments.

- Cross-validation and hyperparameter variation: To improve the model, cross-validation is performed, and hyperparameters are adjusted. This helps prevent overfitting and improves the model's generalization. - Algorithm selection: Based on the results obtained, you can decide to change the machine learning algorithm used if you believe it can improve the results.
- End: The process concludes when the results are satisfactory and the model is ready to be deployed.

4. Proposal

4.1. Data preprocessing

The cybersecurity data used in this study was obtained from the publicly available UNSW-NB15 dataset and is appropriate for the Indian context. Using the IXIA PerfectStorm tool, the dataset, originally developed by the University of New South Wales, contains real network traffic as well as traffic representing various types of network attacks, such as fuzzers, analysis tools, backdoors, and denial of service generalC. Each entry in the dataset contains 49 characteristics, including indicators indicating whether the traffic represents an attack and, if so, the specific type of attack.
The entire dataset consists of 2.54 million records. In this project, two subsets were collected to develop the model: a training set with 175,341 records and a test set with 82,322 records. These subsets were predefined for the evaluation of standard training and machine learning, and deep learning algorithms.
This study focuses on developing a model to determine if a given network footprint indicates the use of this type of attack. To achieve this goal, the structure of the existing classification labels in the dataset was reorganized to indicate two outcomes: to indicate the presence or absence of malignancy. This flexibility enables targeted detection and analysis, which is critical to strengthening cybersecurity measures for India's network infrastructure.

Table 1: Training And Test Records for the "No-Exploit" and "Exploit" Categories.

Category	Training Record	Test Record
No-Exploit	141948	71200
Exploit	33393	11132

4.1.1. Data cleaning

Data was imported from the CSV file "UNSW_NB15_training-set.csv" using the pandas library and stored in a data frame named "data."

- A data cleaning process was performed to remove any incorrect or missing data that could affect the analysis. These included:
- Removal of null values.
- Removal of duplicate records.
- A new column was created in the data frame called "Exploits," which was set to 1 if the "attack_cat" column had the value "Exploits," and 0 otherwise. This target variable will be used for subsequent analysis and modeling.
- Relevant features were selected for data analysis and modeling. In this case, a new dataframe called "datos_x" was created containing all the features (columns) from the original "datos" dataframe, except for the columns not relevant to the analysis ("id", "attack_cat", "Exploits", "proto", "label", "response_body_len", "service", and "state").
- The "get_dummies" method from the pandas library was used to transform categorical variables into binary variables, thus facilitating proper interpretation by machine learning algorithms.
- The StandardScaler scaler from the Scikit-learn library was used to normalize the data, ensuring that all features have the same scale and improving the performance of machine learning algorithms.

4.1.2. Feature selection

Feature selection is important for improving model performance, minimizing overfitting, reducing computational costs, and improving interpretability by focusing on the most important variables. This project uses several techniques to achieve its goal: Principal component analysis (PCA) reduces dimensionality by transforming features into uncorrelated principal components, thus capturing most of the variance; Recursive Feature Elimination (RFE) iteratively removes least significant features to retain only influential variables; Non-negative matrix factorization (NMF) decomposes data into non-negative parts to uncover hidden patterns, thus improving understanding. Together, these methods simplify the data and help build efficient and robust models.

4.1.3. Data balancing

The training dataset handled only 15.6% of records from exploit-type attacks, which can negatively impact model performance, as they may have difficulty learning patterns and characteristics of the minority class.
SMOTE: This is an oversampling technique that synthetically generates new examples of the minority class by interpolation between existing examples. This helps increase the number of records from the minority class and balance the dataset. SMOTE improves model generalization and prevents overfitting to the majority class.
Random Undersampling: On the other hand, this technique consists of randomly removing examples from the majority class until the number of records equals that of the minority class. This helps reduce the size of the dataset and balance the classes. Both techniques are important for dealing with imbalanced datasets, as they improve the accuracy and performance of models by ensuring that all classes are adequately represented in the dataset. However, it is important to evaluate the performance of the models after applying these techniques to determine which is the best option for the specific dataset [16].

4.2. Model architectures

4.2.1. K-nearest neighbor model implementation

The KNN model is implemented by importing the sklearn KNeighborsClassifier library and training it with the training data. Figure 2 shows the pseudocode for the model implementation, its training, and how it is evaluated. The model is evaluated with the test data, and the model's accuracy is printed. A confusion matrix is also shown to visualize the model's performance.

```
# KNN Model
knn = KNeighborsClassifier()        # Initialize the K-Nearest Neighbors classifier
knn.fit(X_train, y_train)           # Train the model using training data
y_pred = knn.predict(X_test)        # Predict the output for test data
```

Fig. 2: KNN Implementation.

To address the class imbalance, the SMOTE method is first used to generate synthetic samples of the minority class. After training, the K Nearest Neighbor (KNN) model is trained using the labeled dataset, and its performance is evaluated. In addition to oversampling, the KNN model is trained and evaluated on the newly generated dataset using the RandomUnderSampler technique.
In addition, dimensionality reduction techniques are investigated to improve model efficiency. Principal Component Analysis (PCA) is applied to the training and test datasets to reduce the number of features by retaining the most important information. Then, the KNN model is retrained using the PCA-transformed data, and its efficiency is measured. Similarly, the data is transformed using Non-Negative Matrix Factorization (NMF), and a new KNN model is trained and evaluated on the processed data. These different preprocessing and transformation methods are intended to improve the model's ability to more accurately classify vulnerability-type attacks.
Once all the algorithm models have been executed, a comprehensive comparison of each one is performed by generating a classification report using the "classification_report" library from "sklearn.metrics." This report provides a detailed view of the precision (Pr), recall (Rc), and F1-score (F1) obtained. It also allows for evaluating the performance of "exploit" attacks and those that do not fall into this category, identified as "non-exploit." The graphs with the results are presented below.
Figure 3 shows that the KNN, KNN with NMF, and KNN with PCA models exhibit higher accuracy compared to the other models evaluated. These models demonstrate a remarkable ability to accurately identify non-exploit attacks. However, it is observed that, in the specific case of "exploit" attacks, their accuracy is limited, reaching a maximum of 74%. Therefore, it is clear that there is still room for improvement in the accurate detection of these types of attacks. As for the other models analyzed, none achieved an accuracy close to this percentage.

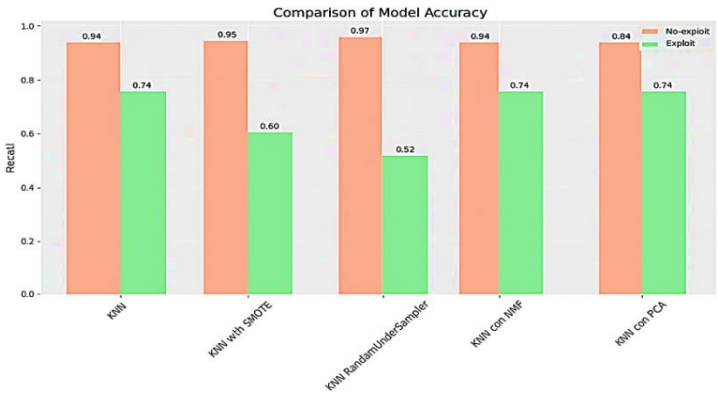

Fig. 3: KNN Accuracy.

As shown in Figure 4, the K-nearest neighbor (KNN) model performs well in identifying the type of malicious attack. Specifically, the model achieves 83% Rc for malicious attacks and 88% Pr for malicious attacks. These metrics show that it should be achieved, but it is not optimal. Compared with other models that perform better in fault detection, the KNN model is still able to detect faults more accurately.
However, its overall implementation is still limited. Despite the high Rc, the overall accuracy of the model is only 52%, which means that the false positive rate is high and the malicious attack instances are incorrectly labeled as malicious attacks. This imbalance drastically reduces the reliability of the model in real-world scenarios. Therefore, the random undersampling of the KNN model performs well in capturing the threats associated with malware attacks, which leads to misclassification of malware attack events, limiting its use as a reliable solution for accurate malware detection.

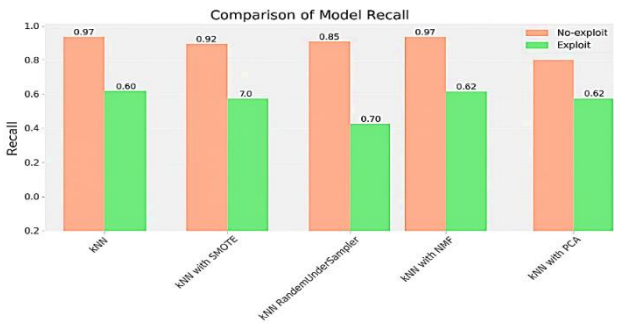

Fig. 4: KNN Recall.

The results highlight the challenges of developing models that can effectively distinguish between different types of cyber attacks, especially vulnerability attacks. In this context, accurate detection requires a careful balance between Pr and Rc, as both metrics provide key information about whether the model can accurately detect threats and reduce misclassifications. These issues highlight the need for further improvements, such as through techniques such as hyperparameter tuning, optimizing feature selection, or using more sophisticated algorithms. These improvements can help improve detection accuracy and reduce false positives, leading to more robust and reliable intrusion detection systems.

4.2.2. Random forest model implementation

The random forest classification model was optimized using GridSearchCV and KFold cross-validation for hyperparameter search. During the key parameter search, we explored multiple settings, including: using the "Gini coefficient" and "entropy" variables as node splitting criteria to measure purity; tree depths of 10, 15, and 20 to observe the growth of each tree; and changing the number of trees in the forest from sa0 to forest. values of 3, 5, and 7, which affect the branching pattern of the trees. The goal of this comprehensive adjustment is to find the best configuration to build a robust and accurate classification model. Figure 5 explains the step-by-step implementation of the model.

```
# Create a KFold object with 10 splits for cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# Define the search space for the hyperparameters we want to tune
search_params = {
    'criterion': ['gini', 'entropy'],       # Criterion for measuring the quality of a split
    'max_depth': [10, 15, 20],              # Maximum depth of the tree
    'n_estimators': [20, 50, 70],           # Number of trees in the forest
    'min_samples_split': [3, 5, 7]          # Minimum number of samples required to split a node
}

# Create an instance of the RandomForestClassifier with a random seed
model_rf = RandomForestClassifier(random_state=42)

# Create a GridSearchCV object to perform an exhaustive search over hyperparameters
# Using cross-validation with the 'recall' metric
best_model_search = GridSearchCV(model_rf, search_params, cv=kf, verbose=10, scoring="recall")

# Fit the model using the training data X_train and y_train
best_fitted_model = best_model_search.fit(X_train, y_train)
```

Fig. 5: Random Forest Implementation.

The code in Figure 5 uses GridSearchCV to perform hyperparameter tuning for a 10-fold cross-validation random forest classifier. It defines a parameter table as a criterion to find the best combination of max_depth, n_estimators, and min_samples_split. The search is guided by the recall score, which is useful for ambiguous classification tasks. Finally, the best model is trained on the training data to achieve better generalization and recall.

To address class imbalance in the dataset, we first applied the RandomUnderSampler technique. This approach reduced the number of instances of most classes, creating a more balanced dataset. We then trained the model on this reduced training data and evaluated it on the test set. To evaluate its effectiveness, we created a confusion matrix and a classification calculation to determine the ability of the model to correctly classify the two classes.

In addition to oversampling, we used an oversampling technique, SMOTE, which generated synthetic samples to obtain a balanced oversampling. After applying SMOTE, we used Non-Negative Matrix Factorization (NMF) to reduce the dimensionality of the training and testing datasets. We then trained a new Random Forest model on the transformed data. We evaluated the performance of the model through the classification report, which comprehensively shows the accuracy, Pr, Rc, and F1 after applying SMOTE and NMF simultaneously.

After running all models, the best hyperparameters were determined: criterion='gini', max_depth=20, min_samples_split=3, n_estimators=70. The classification report reports the Pr, Rc, and F1 for the weak and strong models. The results show that random forest and random forest PCA (without SMOTE) have higher accuracy. While the recall scores are generally high (Figure 7), they are not at a level where the best models can be used reliably.

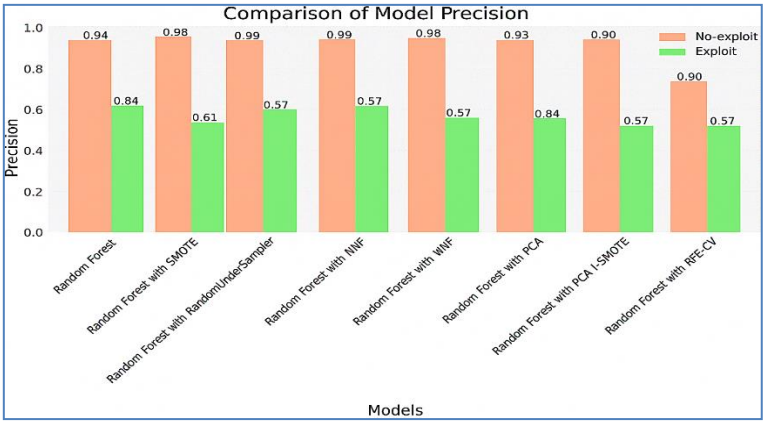

Fig. 6: Accuracy Comparison of the Random Forest Models.

4.2.3. Support vector machine (SVM) model implementation

For the implementation of this model, several possible combinations of hyperparameters were evaluated using the GridSearchCV technique to find the combination that provided the best model performance. The search was performed using KFold cross-validation with 10 partitions to obtain a robust evaluation and avoid model overfitting.
The objective of the search was to maximize the Rc metric during cross-validation, which allowed for the correct identification of as many positive examples as possible. With the best hyperparameters found, the SVC (Support Vector Classifier) model was tuned to obtain an optimal model for the classification task.

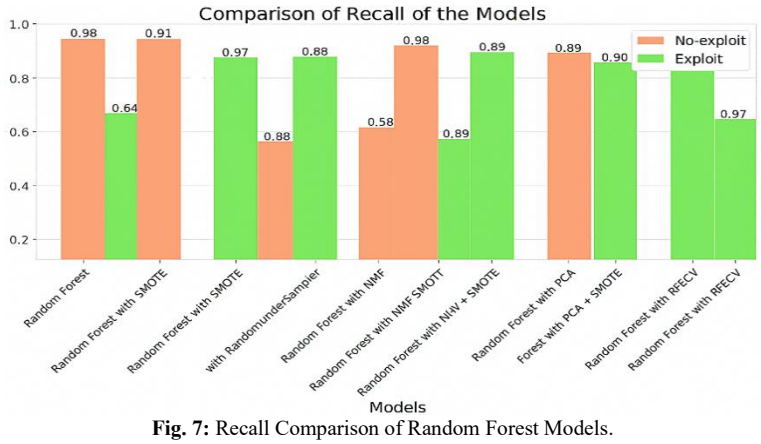

Fig. 7: Recall Comparison of Random Forest Models.

1) Parameter C: This hyperparameter controls the penalty for incorrectly classifying points during the SVM classifier training process. Smaller values of C generate larger margins and allow some points to be misclassified, while larger values of C generate narrower margins and heavily penalize misclassifications. The values tested were [0.1, 1, 10, 100, 1000].
2) Gamma parameter: This hyperparameter affects the shape of the kernel function and can influence the complexity of the SVM model. A small gamma value produces a model with a broader, smoother kernel, which can lead to low bias and high variance. On the other hand, a large gamma value generates a model with a sharper, more complex kernel, which can lead to high bias and low variance. The values tested were [1, 0.01, 0.001, 0.0001].
3) Kernel parameter: This hyperparameter specifies the type of kernel function to use in the SVM model. The two kernels tested are 'rbf' (Radial Basis Function) and 'linear' (linear kernel).

Figure 8 explains the step-by-step process for implementing the model.

```
# Create a KFold object with 10 splits for cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# Define the search space for the hyperparameters we want to tune
search_params = {
    'C': [0.1, 1, 10, 100, 1000],          # Regularization parameter
    'gamma': [1, 0.01, 0.001, 0.0001],     # Kernel coefficient
    'kernel': ['rbf', 'linear']            # Type of kernel to use
}

# Create an instance of the SVM (Support Vector Classifier) with a random seed
model_svm = svm.SVC(random_state=42)

# Create a GridSearchCV object to perform exhaustive search over hyperparameters
# Using the kf cross-validation object and the 'recall' metric
best_model_search = GridSearchCV(model_svm, search_params, cv=kf, verbose=10, scoring="recall")

# Fit the model using the training set X_train and y_train
best_fitted_model = best_model_search.fit(X_train, y_train)
```

Fig. 8: SVM Implementation.

To address class imbalance, the Random Undersampling technique (RandomUnderSampler) was used. The model is trained with the training data and evaluated with the test data. A confusion matrix and a classification report are generated to assess the model's performance.
Additionally, an oversampling process is performed using the SMOTE technique. After applying SMOTE, the non-negative matrix factorization (NMF) technique is applied to the training and test data. Another Random Forest model is trained with the transformed data, and its performance is evaluated. A classification report is generated to assess the model's performance after applying SMOTE and NMF.
Once all the algorithm models were executed, the results obtained with each one were analyzed, and the classification_report function from the sklearn library was run. The following results were obtained for the Pr and Rc of each model:
Figure 9 shows that the models with the best Pr were the SVM and SVM with NMF models. They obtained the same Pr score for detecting non-exploit attacks, and SVM with NMF was only 0.01 better at detecting exploit-type attacks.
Regarding the RC results, Figure 10 shows that most of the algorithms obtained high scores for detecting non-exploit attacks. However, four algorithms stood out with similar scores: SVM with SMOTE, SVM with random undersampler, SVM with NMF and SMOTE, and SVM with RFECV.

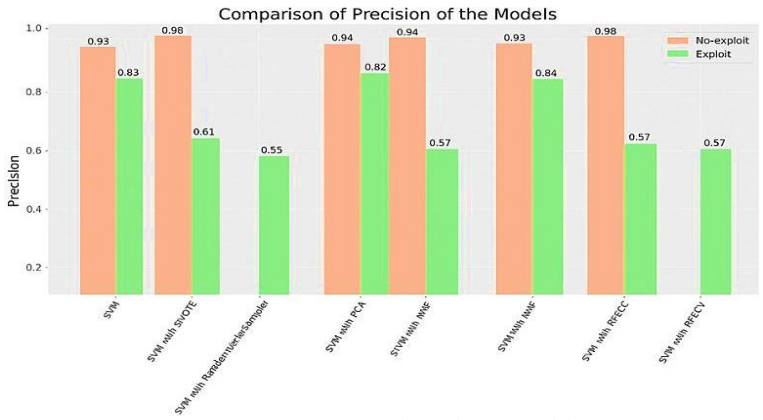

Fig. 9: Accuracy Comparison of SVM Models.

4.2.4. Naive Bayes Model Implementation

A multinomial naive Bayes model is constructed using scikit-learn's MultinomialNB, assuming that the given categorical features are independent of each other. The model is trained on X_train and Y_train and tested on X_test to generate predictions (Y_pred). Its performance is evaluated using accuracy_score, which measures the percentage of correct predictions, indicating the effectiveness of the model.

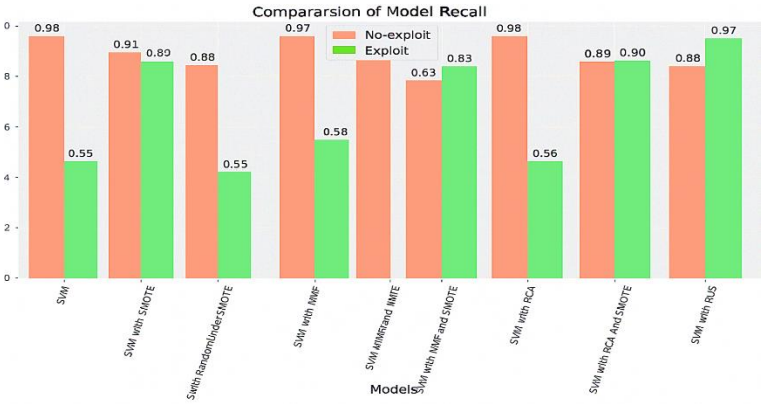

Fig. 10: Recall Comparison of Random Forest Models.

The confusion matrix evaluates the performance of the model created using the confusion_matrix function based on the number of true/false positive and negative samples. Using the Multinomial Naive Bayes model (shown in Figure 11), SMOTE allows using the training data to make good predictions, improves precision and recall, and improves the model's ability to distinguish between categories.

```
# Initialize the Naive Bayes model
# MultinomialNB is suitable for classification with discrete features
model = MultinomialNB()

# Train the model using the training dataset
model.fit(X_train, y_train)

# Make predictions using the trained model and the test dataset
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
# Accuracy is defined as the proportion of correct predictions out of all predictions made
accuracy = accuracy_score(y_test, y_pred)

# Calculate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)

# Display the evaluation results of the model
print("Accuracy: ", accuracy)
print("Confusion Matrix:\n", confusion)
```

Fig. 11: Implementation of the Naive Bayes Model.

It is worth mentioning that, although the training time of the multinomial Naive Bayes algorithm was considerable, the benefit of having a balanced dataset has proven to be essential for obtaining more accurate and robust results in the classification of instances in the context of the problem addressed. The combination of the SMOTE technique with the Naive Bayes model has provided an effective and efficient solution for the analysis of imbalanced data, and the results obtained support the usefulness of this approach for improving the performance of classification models in situations where inequality between classes is a significant challenge.

Once all the algorithm models have been executed, a comprehensive comparison of each one is performed by generating a classification report. This report provides a detailed view of the Pr, Rc, and F1 obtained. It also allows for evaluating performance for exploit-type

attacks and those that do not fall into this category, identified as non-exploit attacks. The graphs with the results obtained for each model are presented below:

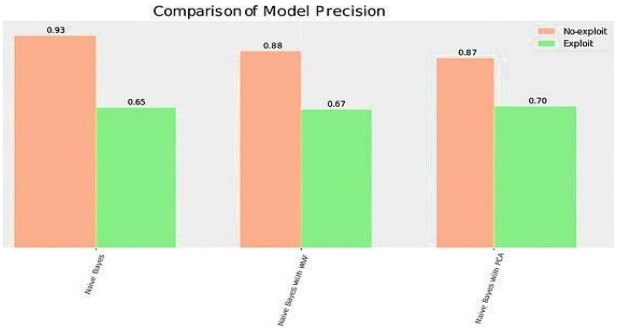

Fig. 12: Naive Bayes Model Accuracy.

Figure 12 compares the accuracy of detecting "non-use" and "use" events using the naive Bayes model and its improved version using NMF and PCA. The accuracy of the naive Bayes model is 0.93 for "non-exploitation" and 0.65 for "exploitation." Using NMF, the accuracy for "non-use" is 0.88 and for "use" is 0.67. Using PCA, the accuracy for "non-exploitation" is 0.87 and for "exploitation" is 0.70. The naive Bayes model has high accuracy in classifying non-exploitation events and moderate accuracy in classifying exploitation events. Applications of NMF and PCA with naive Bayes models show a tendency for the accuracy of "non-use" events to decrease, while the classification of "use" events gradually improves. As can be seen in Figure 13, the results obtained during the algorithm evaluation were unfavorable. The model showed poor performance in the Rc metric, specifically in the identification of exploit-type attacks. Rc is a critical metric, as it measures the model's ability to correctly capture positive cases. A low Rc value indicates that the algorithm was unable to adequately identify most exploit-type attacks, which could have significant consequences for the system's security and performance. For this reason, the data presented in Figure 12, which reflects the algorithm's accuracy, will not be considered. This is because accuracy alone does not provide a complete assessment of the model's performance, and in this case, other metrics such as Rc are more relevant for identifying exploit-type attacks.

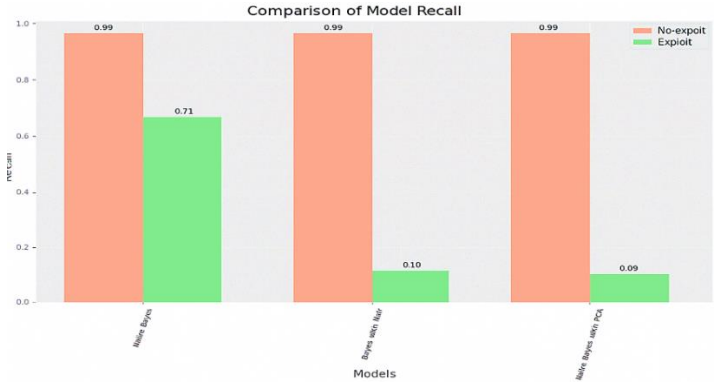

Fig. 13: Recall of the Naive Bayes Model.

4.2.5. Neural network model implementation

To handle class imbalance, a binary classification model was built using Keras with the SMOTE technique to generate synthetic minority samples. The model architecture includes two hidden layers (64 and 32 neurons) with ReLU activation and a sigmoid-activated output layer. It uses binary cross-entropy loss, the Adam optimizer, and recall as the main metric. Trained for 10 epochs with a batch size of 32, the model setup is illustrated in Figure 14.

```
# Step 1: Apply SMOTE to the dataset
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Print class distribution before and after applying SMOTE
print("Class distribution before SMOTE:", y_train.sum() / len(y_train))
print("Class distribution after SMOTE:", y_resampled.sum() / len(y_resampled))

# Step 2: Build the model
model = Sequential()
model.add(Dense(64, input_dim=X_resampled.shape[1], activation='relu'))  # Hidden Layer 1
model.add(Dense(32, activation='relu'))                                  # Hidden Layer 2
model.add(Dense(1, activation='sigmoid'))                                # Output Layer (1 neuron

# Step 3: Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=[keras.metrics.Recall()])

# Step 4: Train the model
model.fit(X_resampled, y_resampled, epochs=10, batch_size=32)

# Step 5: Make predictions
y_pred = model.predict(X_resampled)

# Round predictions to 0 or 1
y_pred = np.round(y_pred)

# Evaluate the results
accuracy = accuracy_score(y_resampled, y_pred)
precision = precision_score(y_resampled, y_pred)
recall = recall_score(y_resampled, y_pred)
f1 = f1_score(y_resampled, y_pred)

# Print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Fig. 14: NN Implementation.

The NN demonstrated outstanding performance in the classification task, exhibiting an effective ability to distinguish between attacks and non-attacks with a remarkable success score. Compared to other classification models used in this study, the NN stood out by obtaining the highest scores in terms of Pr, Rc, and F1. To further improve model quality, as shown in Figure 14 (step 1), the SMOTE oversampling technique was implemented during data preprocessing. The use of SMOTE was instrumental in counteracting class imbalance and, consequently, significantly contributed to the excellent results obtained by the NN. The combination of the network architecture, the application of SMOTE, and the use of appropriate evaluation metrics has resulted in a robust and reliable classification model that shows promise for applications in attack detection and other challenging classification scenarios. A table with the results is presented below:

Table 2: Neural Network Results Report.

	Pr	Rc	F1
0	0.96	0.87	0.91
1	0.88	0.96	0.92

The neural network model achieved excellent performance with improved class balance by SMOTE, with an overall accuracy of 92%. The precision was 0.96 (class 0) and 0.88 (class 1), the recall was 0.87 and 0.96, and the F1 score was 0.91 and 0.92. The results show stable and consistent classification performance between the two classes.

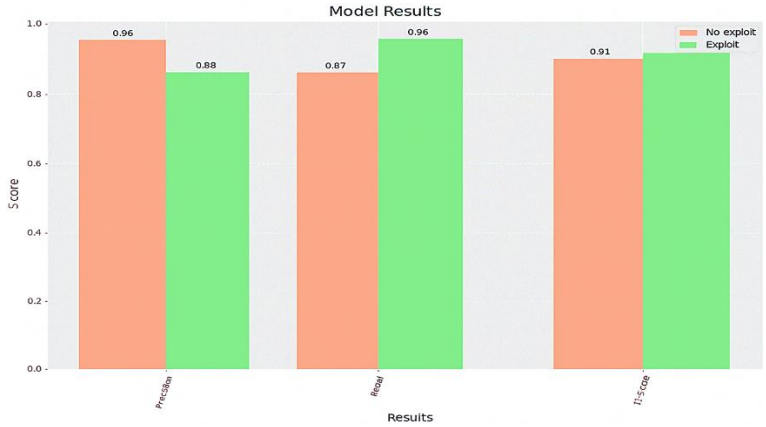

Fig. 15: Results of the Neural Network Models.

The stability and consistency performance metrics for precision, recall, and F1 score are shown in Figure 15. The confusion matrix in Figure 16 shows that there are 49,639 true negatives, 55,169 true positives, 1,839 false negatives, and 7,369 false positives, indicating that the classification is reliable, but the performance varies greatly.

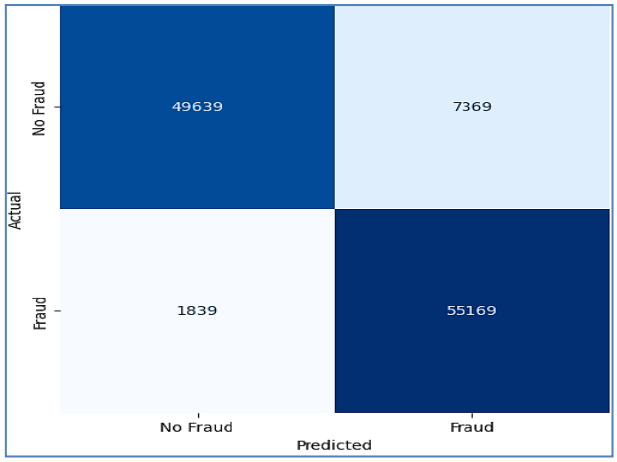

Fig. 16: Confusion Matrix of the Neural Network.

5. Analysis of the best model

The proposed neural network (NN) model performs well in terms of recall and precision. The success is due to a well-designed architecture consisting of two layers with 64 and 32 neurons to extract nonlinear features using ReLU activation function and a single output neuron using Sigmoid activation function for binary classification.

An important step in the preprocessing was the use of the SMOTE method to solve the class imbalance problem. This method generates low-class synthetic examples by interpolating between existing examples, which improves the class representativeness and generalization ability of the model. As a result, the NN can better distinguish between different classes without being biased towards the dominant class.

The estimation results of the model were excellent. The classification accuracies for class 0 and class 1 are 0.96 and 0.88, respectively, indicating high accuracy in predicting positive and negative samples. The RC is also good, 0.87 and 0.96 for Class 0 and Class 1, respectively, indicating the ability of the model to identify true positive samples and reduce false positive and false negative samples. The F1 value further indicates the balanced performance of the model, 0.91 and 0.92 for Class 0 and Class 1, respectively. These results confirm that NNs can maintain a trade-off between Pr and Rc.

Overall, the model achieved an accuracy of 92%, indicating its robustness in performing two classification tasks. Macro-based average metrics also indicate the strong generalization ability of the model. The structure of this configuration is described in more detail, and the results are shown in Fig. 11, which further supports the conclusions regarding the optimal settings and performance of the NN.
Although the SVM and Random Forest models, both using Random Undersampling, achieved a Rc of 0.97, which is 0.01 higher than that obtained by the NN, they did not perform well in terms of Pr. This indicates that although these models were successful in correctly identifying most positive cases, they also had a high rate of false positives, which negatively impacts the model's overall accuracy.
This behavior is concerning, as Pr is a crucial metric for evaluating a model's ability to accurately classify positive and negative cases.
Due to these shortcomings in the Pr metric, both the SVM and Random Forest models cannot be considered candidates to be the best model for this classification task. Despite their high Rc, their lack of Pr makes them less reliable and unsuitable options for applications requiring high classification accuracy.

5.1. Testing neural network on IoT datasets and adversarial scenarios

To assess the generality and robustness of the proposed neural network (NN) model on the UNSW-NB15 dataset, additional tests were performed using two IoT-specific cybersecurity datasets—TON_IoT and BoT-IoT—and FaGsnarient Mestarient using SiSMF. The TON_IoT dataset, created by the UNSW-NB15 and research team, includes telemetry and network traffic data from IoT sensors, with attack categories such as DoS, Backdoor, and Injection. Preprocessing included "exploiting" and "non-exploiting," normalizing, and using the SMOTE technique to overcome class imbalance. The model achieved high performance with an overall accuracy of 89.4%, but the accuracy decreased slightly as the attack types became more comprehensive. The BoT-IoT dataset, which contains classes such as Recon, Theft, and DDoS, caused more problems due to severe class imbalance and noise. After feature adaptation and SMOTE, the NN achieved an overall accuracy of 85.1%, with a clear decrease in precision but retained recall, indicating a strong detection ability without compromise on false positives.
To evaluate the collision strength, an FGSM attack with perturbation parameter $\varepsilon = 0.1$ was used on the UNSW-NB15 test set. The accuracy of the model decreased from 92% to 71.4%, and the precision and recall decreased drastically for both the "exploitation" and "non-exploitation" classes. This shows that the NN model is vulnerable to adversarial attacks, leading to incorrect classifications of human-perceived disturbances. Table 3 summarizes the performance of the NN model in different scenarios:

Table 3: Performance Comparison of NN Model on IoT and Adversarial Data.

Scenario	Class	Precision	Recall	F1-Score	Accuracy
TON_IoT	0	0.91	0.88	0.89	89.4%
	1	0.85	0.93	0.89	
BoT-IoT	0	0.87	0.80	0.83	85.1%
	1	0.78	0.91	0.84	
FGSM ($\varepsilon = 0.1$)	0	0.75	0.67	0.70	71.4%
	1	0.64	0.78	0.70	

These results show that, while neural network models have excellent classification performance in various IoT scenarios, they also highlight the need for techniques to improve susceptibility to adversarial perturbations, such as adversarial learning, dropout, or ensemble methods. For use in secure environments, it is necessary to improve the robustness of models to such inputs.
Table 4 is a comparative table and summarizes prior ML-based IoT IDS approaches vs. your advancements:

Table 4: Comparison between Prior ML-Based IDS and Proposed Neural Network-Based IDS

Aspect	Prior ML-Based IoT IDS	Proposed NN-Based IDS (This Work)
Algorithms Used	KNN, SVM, Random Forest, Naive Bayes	Deep Neural Network with SMOTE balancing
Handling of Class Imbalance	Often ignored or addressed with basic sampling	SMOTE (oversampling) and RandomUndersampling applied systematically
Feature Engineering	Limited preprocessing; often manual feature selection	Automated selection with PCA, NMF, RFE
Categorical Handling	Minimal or inconsistent treatment	One-hot encoding using get_dummies
Normalization	Often missing or inconsistent	Standardized using StandardScaler
Generalization to Other Datasets	Rarely tested on external IoT datasets	Validated on TON_IoT and BoT-IoT with strong performance
Performance on Exploit Class	Low recall and precision	High recall (0.96) and F1-score (0.92) for exploit detection
Adversarial Robustness	Not tested	Evaluated using FGSM ($\varepsilon = 0.1$); performance drop analyzed
Model Accuracy	Moderate (60–85%), often with high false positives	High (92%) with a strong balance between Pr and Rc
Scalability and Adaptability	Limited due to shallow models	High, due to neural network architecture and modular preprocessing

5.2. Adversarial robustness

Although the proposed neural network (NN) model is effective in classifying exploitative and non-exploitative attacks, its robustness against adversarial evasion attacks remains a critical concern. Defensive attacks, such as the Fast Gradient Signal Method (FGSM), implicitly use input features to mislead the classifier without changing the underlying semantics of the network traffic. In this study, FGSM was used with a perturbation value $\varepsilon = 0.1$ to evaluate the robustness of the model. The NN model, which initially achieved 92% accuracy on the clean test set, dropped to 71.4% accuracy in the adversarial cases. Precision and recall decreased significantly, especially for the minority (exploitation) class, indicating that the model is vulnerable to adversarial manipulations. This vulnerability poses a risk in the real world, where attackers can intentionally make inputs to evade detection. Future work to address this limitation includes adversarial learning, defensive distillation, and anomaly detection architectures to improve model reliability against such sneaky attacks. Developing reliable IDS solutions for dynamic and threatening IoT environments requires the inclusion of adversarial robustness as a key design requirement.

6. Comparative summary of prior ML-based IoT IDS and current advancements

Below is Table 5 that summarizes prior ML-based IoT security solutions, their key limitations, and how this work advances the field:

Table 5: Summary of Prior ML-Based IoT Security Solutions and Advancements in this Work

Approach	Limitations of Prior Works	Advancements in This Work
K-Nearest Neighbors (KNN)	Struggles with high-dimensional data, poor exploit-class detection, sensitive to imbalance	Dimensionality reduction with PCA/NMF, class balancing via SMOTE, and undersampling
Support Vector Machine (SVM)	High recall but low precision, poor scalability with large datasets	GridSearchCV tuning, tested on multiple datasets, recall–precision balance analyzed
Random Forest (RF)	Tends to overfit imbalanced data, lacks adversarial robustness	Hyperparameter tuning, class balancing, and dimensionality reduction for improved generality
Naive Bayes (NB)	Strong for non-exploit class, but poor recall for exploit attacks	Combined with SMOTE and PCA/NMF to improve minority-class detection
Traditional ML pipelines	Limited preprocessing, poor handling of categorical/imbalanced data	Full pipeline: data cleaning, one-hot encoding, normalization, and intelligent feature selection
Cross-dataset testing	Rarely tested on external IoT datasets	Evaluated on TON_IoT and BoT-IoT datasets with strong generalization performance
Adversarial robustness	Not evaluated	FGSM-based adversarial testing performed; future defense strategies identified
Deployment readiness	Models are mostly offline, lacking real-time or energy-efficient implementation	Framework outlined for real-time, scalable, and low-power IoT deployment

7. Future work

Future work will focus on increasing the size, energy efficiency, and real-time performance of the Intrusion Detection System (IDS) model. To support large-scale IoT environments, the system must be configured for distributed or edge deployment to efficiently process large amounts of data. Improving energy efficiency is important, especially for resource-constrained IoT devices; this can be achieved using modern compression techniques such as merging and quantization. In addition, the current offline model needs to be transformed into a real-time detection system that includes real-time traffic analysis, low-latency summary, and automated alert integration. These improvements help make an intrusion detection system (IDS) practical, lightweight, and responsive for real-world IoT applications.

8. Conclusions

The two-layer NN model achieved the best results in vulnerability-based attack detection. The model exhibited strong performance metrics, achieving a Pr of 0.92, a Rc of 0.965, and an F1 of 0.92. The introduction of the SMOTE oversampling technique effectively overcame the class imbalance problem, thereby improving the prediction ability of the model. In addition, the conversion of categorical variables into binary format through OneHotEncoding further improved the overall performance of the model. These findings demonstrate the effectiveness of deep learning methods in cybersecurity applications, especially in identifying vulnerability-based threats. The ability of NNs to learn and recognize complex patterns in data is a key factor in achieving high- Pr identification of such attacks.

References

[1] Alosaimi, S., & Almutairi, S. M. (2023). An intrusion detection system using BoT-IoT. *Applied Sciences*, 13(9), 5427. https://doi.org/10.3390/app13095427.

[2] Damaševičius, R., Bacanin, N., & Misra, S. (2023). From sensors to safety: Internet of Emergency Services (IoES) for emergency response and disaster management. *Journal of Sensor and Actuator Networks*, 12, 41. https://doi.org/10.3390/jsan12030041.

[3] Yogesh, K. M., Stephan, T., Bharath, M. B., Gad, I., Arpitha, S., & Prakash, M. M. (2023). Characterization of darknet traffic detection using time domain features. In *Proceedings of the International Conference on Computer Vision and Internet of Things (ICCVIoT'T)* (pp. 233–237). https://doi.org/10.1049/icp.2023.2881.

[4] Alotaibi, B. (2023). A survey on industrial Internet of Things security: Requirements, attacks, AI-based solutions, and edge computing opportunities. *Sensors*, 23, 7470. https://doi.org/10.3390/s23177470.

[5] Srinivasan, S., & Deepalakshmi, P. (2023). Enhancing the security in cyberworld by detecting the botnets using ensemble classification based machine learning. *Measurement: Sensors*, 25, 100624. https://doi.org/10.1016/j.measen.2022.100624.

[6] Ju, Z., Zhang, H., Li, X., Chen, X., Han, J., & Yang, M. (2022). A survey on attack detection and resilience for connected and automated vehicles: From vehicle dynamics and control perspective. *IEEE Transactions on Intelligent Vehicles*, 7, 815–837. https://doi.org/10.1109/TIV.2022.3186897.

[7] Bahashwan, A. A., Anbar, M., Manickam, S., Al-Amiedy, T. A., Aladaileh, M. A., & Hasbullah, I. H. (2023). A systematic literature review on machine learning and deep learning approaches for detecting DDoS attacks in software-defined networking. *Sensors*, 23(9), 4441. https://doi.org/10.3390/s23094441.

[8] Alam, S., Bhatia, S., Shuaib, M., Khubrani, M. M., Alfayez, F., Malibari, A. A., & Ahmad, S. (2023). An overview of blockchain and IoT integration for secure and reliable health records monitoring. *Sustainability*, 15, 5660. https://doi.org/10.3390/su15075660.

[9] Rihan, S. D. A., Anbar, M., & Alabsi, B. A. (2023). Approach for detecting attacks on IoT networks based on ensemble feature selection and deep learning models. *Sensors*, 23(17), 7342. https://doi.org/10.3390/s23177342.

[10] Suleski, T., Ahmed, M., Yang, W., & Wang, E. (2023). A review of multi-factor authentication in the Internet of Healthcare Things. *Digital Health*, 9, https://doi.org/10.1177/20552076231177144.

[11] Al-Amiedy, T. A., Anbar, M., Belaton, B., Bahashwan, A. A., Hasbullah, I. H., Aladaileh, M. A., & Mukhaini, G. A. (2023). A systematic literature review on attacks defense mechanisms in RPL-based 6LoWPAN of Internet of Things. *Internet of Things*, 22, 100741. https://doi.org/10.1016/j.iot.2023.100741.

[12] Sodhro, A. H., Awad, A. I., van de Beek, J., & Nikolakopoulos, G. (2022). Intelligent authentication of 5G healthcare devices: A survey. *Internet of Things*, 20, 100610. https://doi.org/10.1016/j.iot.2022.100610.

[13] Nova, K. (2022). Security and resilience in sustainable smart cities through cyber threat intelligence. *International Journal of Information and Cybersecurity*, 6, 21–42.

[14] Elattar, M., Younes, A., Gad, I., & Elkabani, I. (2024). Explainable AI model for PDFMal detection based on gradient boosting model. *Neural Computing and Applications*, 36(34), 21607–21622. https://doi.org/10.1007/s00521-024-10314-y.

[15] Wei, C., Xie, G., & Diao, Z. (2023). A lightweight deep learning framework for botnet detecting at the IoT edge. *Computers & Security*, 129, 103195. https://doi.org/10.1016/j.cose.2023.103195.
[16] Salunkhe, V., Tangudu, A., Mokkapati, C., Goel, D. P., & Aggarwal, A. (2024). Advanced encryption techniques in healthcare IoT: Securing patient data in connected medical devices. *Modern Dynamics in Mathematics and Progressions*, 1, 224–247. https://doi.org/10.36676/mdmp.v1.i2.22.
[17] Alkadi, S., Al-Ahmadi, S., & Ben Ismail, M. M. (2024). RobEns: Robust Ensemble Adversarial Machine Learning Framework for Securing IoT Traffic. *Sensors*, 24, 2626. https://doi.org/10.3390/s24082626.