INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

ORIGINAL ARTICLE

IoT Botnet Detection: A Comparative Performance Analysis of Various ML Models on IoT-23 Dataset

Rintu Das[1]*, Vaskar Deka[1], Gom Taye[2]

1 Department of Information Technology, Gauhati University, Assam, India
2 Department of Computer Science and Engineering, Rajiv Gandhi University, Arunachal Pradesh-791 112, India

OPEN ACCESS

Received: 26/05/2025
Accepted: 24/10/2025
Published: 16/11/2025

Citation: Das R, Deka V, Taye G (2025) IoT Botnet Detection: A Comparative Performance Analysis of Various ML Models on IoT-23 Dataset. Indian Journal of Science and Technology 18(41): 3300-3318. https://doi.org/10.17485/IJST/v18i41.974

* **Corresponding author**.

rintu.das@gmail.com

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: To compare the performance of Machine Learning (ML) models for Intrusion Detection in IoT environment. **Methods:** This study was employed on 7 the IoT-23 dataset, which has 6,046,623 network records from 23 attack and 8 benign scenarios. The models are Random Forest (RF), Decision Tree (DT), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Naïve Bayes (NB), and Logistic Regression (LR). Z-score normalization, Recursive Feature Elimination (RFE), Principal Component Analysis (PCA) and SMOTE balancing were applied to optimize feature selection and handle severe class imbalance (original ratio 13 ≈ 85:15). **Findings:** Among all models, DT, RF, and KNN achieved identical top-tier performance such as, Accuracy = 99.99%, Precision = 99.99%, Recall = 99.99%, and F1-score = 99.99%. RF attained the highest ROC-AUC (0.9999) but required the longest training time (553.08 s). In contrast, DT achieved nearly identical accuracy with the lowest prediction latency (0.0837 s), offering an excellent trade-off between accuracy and real-time deployment. SVM and LR achieved moderate accuracy (64.31%), and NB performed modestly at 72.65%. **Novelty:** The framework, using the entire IoT-23 dataset (≈6 million records) with integrated feature optimization and resampling, demonstrates a latency improvement compared to RF while retaining the same accuracy. The findings shows that Decision Tree (DT) as a lightweight, highly accurate, and real-time solution for IoT botnet detection, suitable for deployment on edge and constrained IoT devices.

Keywords: IoT security; Botnet Detection; IDS; Intrusion Detection System; ML; Botnet; Accuracy; Latency

1 Introduction

The fast growth of IoT devices has made it easier for botnets to launch big DDoS attacks like Mirai and Okiru. Traditional signature-based IDS struggle with zero-day threats, while ML-based models exhibit numerous false positives, imbalances, and latency issues. Earlier studies that used RF, SVM, CNN and GAN-BiLSTM got very good results (about 94 – 99% accuracy), but they often relied on incomplete datasets or were not very efficient in real time.

This study suggests a lightweight, data-driven framework that uses RFE, PCA and SMOTE together to fill in these gaps. It improves feature selection and balances multiclass IoT traffic with the full IoT-23 dataset. Six (6) nos. of classical supervised ML models were analysed and applied on IoT-23 Dataset and a detailed analysis was carried out to explore the best model. DT has been found to be more accurate than more complicated DL methods and can still be used on edge devices. It has a latency of 0.0837 seconds. The suggested method works well for finding IoT botnets in real time because it has high accuracy, low latency, and the ability to grow.

1.1 Research Objectives

The research aims to:

1. Analyze the IoT-23 dataset to identify distinguishing characteristics of benign and malicious IoT traffic.
2. Develop and evaluate multiple classical ML models for multiclass botnet detection.
3. Optimize feature selection and class balance using PCA, RFE and SMOTE to enhance detection accuracy.
4. Compare models on accuracy, false positives and computational efficiency to identify a real-time deployable solution.

1.2 Review of Related Works

Author(s)	Dataset(s)	Algorithm(s)	Accuracy / Key Results	Advantages	Limitations / Remarks
Venu et al. [1]	IoT-23, BoT-IoT	RF, DT, NB, SVM	RF >95%; DT 93–94%; NB 90–92%	RF highly accurate; DT interpretable	NB limited by feature independence; moderate scalability
Garg et al. [2]	BoT-IoT, IoT-23, CIC-DDoS2019	SVM, RF, LGBM, XGBoost	SVM 91–93%; XGB 94.79% (IoT-23), 99.99% (CIC-DDoS2019)	Gradient boosting methods outperform classical ML; faster training	High computational complexity; limited IoT generalization
Farooqi et al. [3]	IoT-23 (12%)	DT, NB, SVM	DT 99.99% (5.2 s), NB 85.04%, SVM 67.97%	DT fastest and most accurate; lightweight design	Used partial dataset only
Nicho et al. [4]	IoT-23	CNN, LSTM	CNN: 93% (binary), 96% (multiclass)	CNN outperforms LSTM in accuracy and efficiency	Deep learning model is computationally heavy
Meng et al. [5]	Mobile IoT traffic	Multiple ML algorithms	High accuracy (not specified)	Lightweight; low device impact	Focused on mobile botnets
Chen et al. [6]	CTU-13	RF, others	RF: 93.6%, FPR: 0.3%	High accuracy and low false alarms	Dataset limited to CTU-13
Hussain et al. [7]	IoT-23, CTU-13	NB, KNN, RF, LR	NB 99.92%, KNN 99.94%, RF 100%, LR 99.91%	Universal feature set; high accuracy	Focused only on Mirai botnet
Segun et al. [8]	BoT-IoT	SMOTE + DL/ML	100% accuracy	Balanced data improves model performance	Overfitting risk; binary classification only
Zainab et al. [9]	BoT-IoT	J48, RF, MLP	0.999 (binary), 0.96 (main), 0.93 (subclass)	SMOTE enhances minority class accuracy	Limited dataset diversity
Maran et al. [10]	Custom IoT dataset	RF, NB, ANN, DT, KNN	RF: 96%	RF most effective	Dataset not standardized
Tarik Himdi et al. [11]	IoT-23	CNN, LSTM, GRU	CNN: 95%	DL models effective for anomaly detection	High computational cost
Mishra et al. [12]	IoT-23, BoT-IoT, UNSW-NB15, ToN_IoT	GAN + BiLSTM Ensemble	IoT-23: 99.08%; BoT-IoT: 99.99%; UNSW-NB15: 99.82%; ToN_IoT: 99.96%	High precision, recall, and F1-score	Complex architecture; heavy processing
Kumari et al. [13]	IoT Intrusion, IoT-23	ML classifiers	99.11% (IoT Intrusion), 99.99% (IoT-23)	Comparative evaluation across datasets	Minimal latency analysis

Continued on next page

Table 1 continued

Alanazi et al. [14]	IoT-23	Ensemble + Feature Selection	Accuracy 99.984%; F1-score 99.983%	Strong generalization and precision	Increased computational cost
Ullah et al. [15]	IoT-23, BoT-IoT, CICIOT2023	CNN–BiLSTM–RF–LR Hybrid (Soft Voting Ensemble)	BoT-IoT: 100%; CICIOT2023: 99.2%; IoT-23: 91.5%	Hybrid adaptive model ensures high scalability	Slight drop in IoT-23 accuracy; complex tuning required
Anshika Sharma et al. [16]	IoT-23	RF, XGB, KNN	RF: 89%; XGB: 88%; KNN: 85%	Practical ML pipeline with feature optimization	Lower accuracy than deep ensembles.
Krzysztoń et al. [17]	IoT-23, CIC-IoT 2023	RF, Autoencoder, Isolation Forest	RF: 99.16–100%; Autoencoder: 98.76–99.27%	Autoencoder promising for unsupervised detection	Isolation Forest unstable across datasets; requires tuning

2 Methodology

The proposed methodology for IoT botnet detection using the IoT-23 dataset. It follows a systematic pipeline for a solution to attain reliable detection performance and practical applicability in real-time IoT environments. The framework consists of the following stages:

The pipeline starts from IoT-23 dataset ingestion → preprocessing → EDA → feature engineering → supervised ML model training → evaluation → comparative performance analysis, ensuring both accuracy and real-time applicability are addressed.

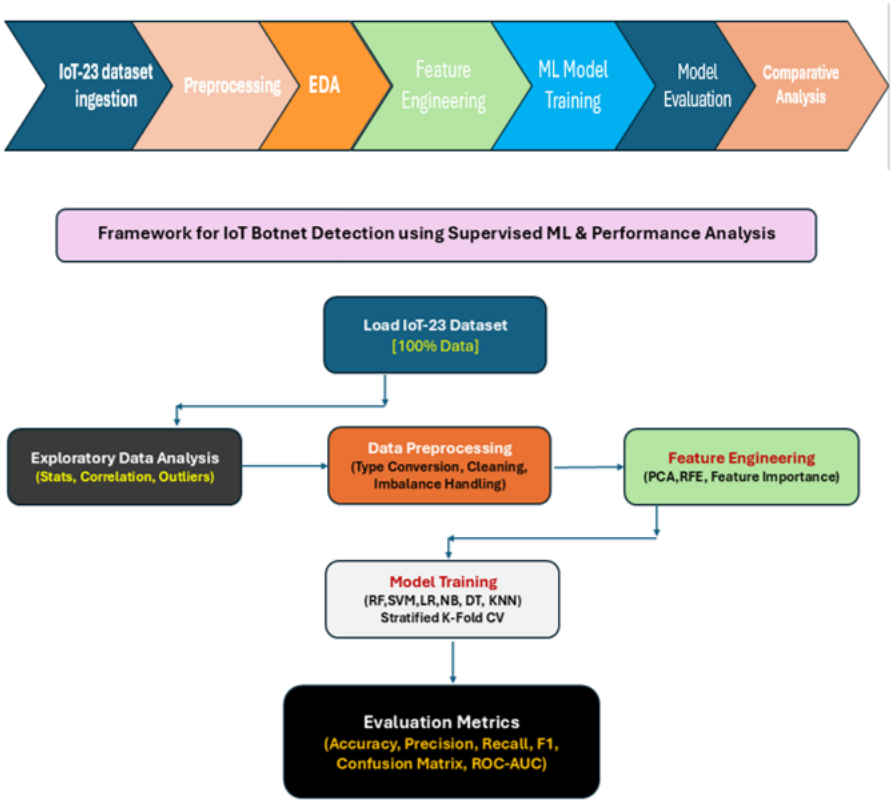

Fig 1. Proposed Framework of ML Performance Analysis on IoT-23 Dataset

2.1 Dataset Study: IoT-23

Stratosphere Laboratory released the IoT-23 dataset in 2020. It is one of the most popular benchmarks for research on how to find IoT botnets [18]. It records the communication patterns of actual IoT devices, including IP cameras, routers, and smart assistants, in both benign and harmful contexts. IoT-23 was made in a controlled lab setting between 2018 and 2019, which makes it possible to reproduce while still being realistic [18], [19]. This is not the case with synthetic datasets. This makes it a great tool for testing ML models and IDS in IoT security.

Table 2. A summary of the IoT-23 dataset

Label	Description	Dataset Rows Count [100%]
PartOfAHorizontal-PortScan	Indicates network traffic consistent with a horizontal port scan across a range of IP addresses.	3389036
Okiru	Associated with the Okiru botnet, it is often involved in DDoS attacks.	1313012
Benign	Represents normal, non-malicious network traffic.	688812
DDoS	Traffic characteristic of a Distributed Denial of Service [20] attack, aiming to overwhelm a target.	638506
C&C	Command and Control traffic, where an attacker control compromised systems.	15286
C&C-HeartBeat	Heartbeat signals from compromised systems to a C&C server.	1332
Attack	A general label indicating some form of malicious activity.	538
C&C-FileDownload	Traffic related to downloading files from a C&C server to a compromised system.	46
C&C-Torii	Associated with the Torii botnet, often used for malware distribution.	30
FileDownload	Generic label for file download activity, could be benign or malicious.	13
C&C-HeartBeat-FileDownload [21]	Combined C&C heartbeat and file download activity.	8
Okiru-Attack	An attack specifically attributed to the Okiru botnet.	3
C&C-Mirai	C&C traffic associated with the Mirai botnet, known for targeting IoT devices. [20]	1
Total		**60,46,623**

The IoT-23 dataset has a lot of different kinds of network traffic and they are grouped into good and bad behavior groups. Here are the main types of attacks, even the ones that aren't harmful: PartOfAHorizontalPortScan is when hackers try to connect to the same port from more than one IP address. DDoS is when a lot of traffic is sent to target systems to make them unavailable. Okiru is a family of botnets that use IoT devices for DDoS attacks and cryptocurrency mining. Benign traffic is when normal IoT operations make connections that are valid and not harmful. Another kind of attack is Command and Control (C&C) communications. In this case, bots (compromised devices) get instructions from a server that is not on their network. C&C-HeartBeat and C&C-HeartBeat-FileDownload are two types of regular file transfers and beaconing between infected devices and command and control (C&C) servers. FileDownload and C&C-FileDownload are two kinds of bad or illegal data transfers. The C&C-Torii and C&C-Mirai classes are famous IoT botnets that spread malware and launch DDoS attacks on a large scale. The Attack label now includes general malicious behaviors that don not fit into any other specific groups. These labels work together to show how IoT networks work as a whole, which enables us to group botnet-related traffic patterns into very small groups.

2.2 Exploratory Data Analysis (EDA)

In this section, a detailed Exploratory Data Analysis (EDA) was conducted to understand feature characteristics and required preprocessing:

1. **Descriptive Statistics**: Data types, distributions, and missing values were examined in the raw dataset.

   As shown in Figure 2, and Initial Data Exploration was carried out and the result (6046623, 22) shows that the dataset has 6,046,623 rows and 22 columns. This dataset is large. It has columns for "ts" (timestamp), "uid" (unique identifier), "id.orig_h" (originating IP), "id.resp_h" (responding IP), "proto" (protocol), "service," "duration," "orig_bytes,"

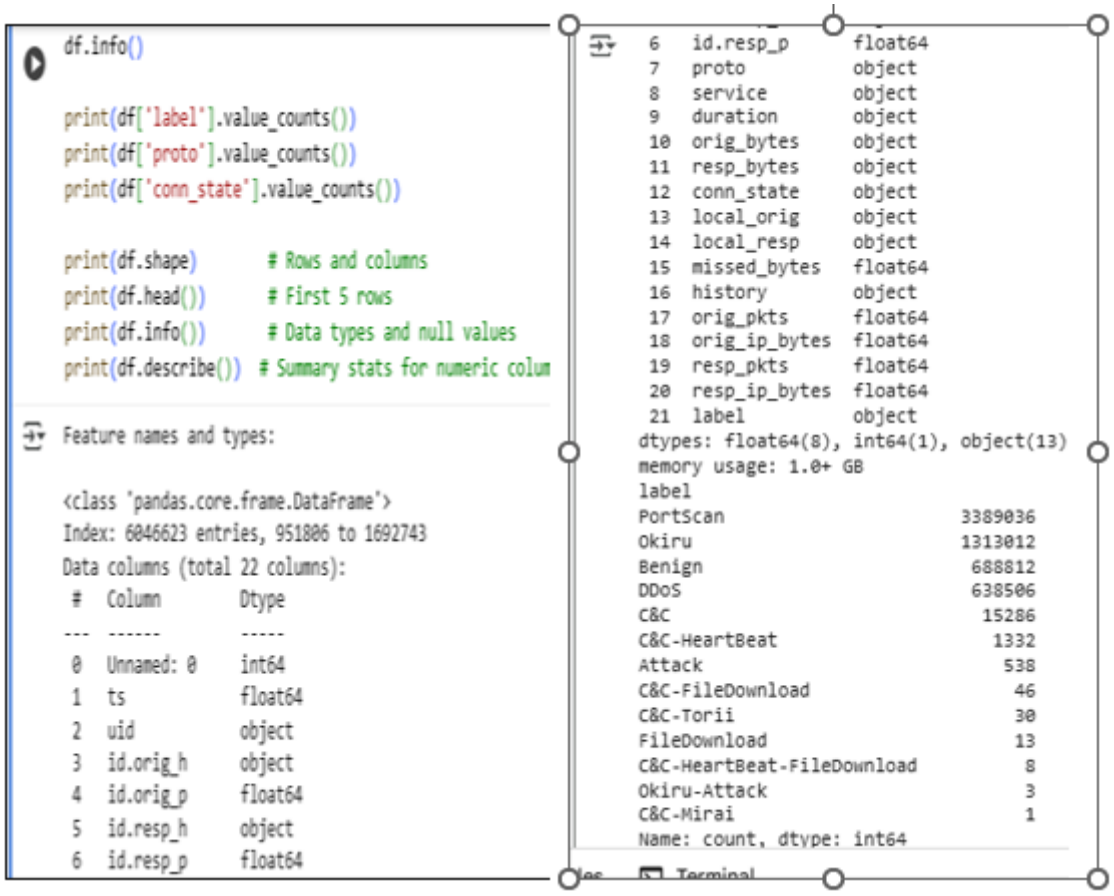

Fig 2. Descriptive Statistics

"resp_bytes," "conn_state" (connection state), "history," and "label." Some columns are of the "object" type,. These are 'uid', 'id.orig_h', 'id.resp_h', 'proto', 'service', 'duration', 'orig_bytes', 'resp_bytes', 'conn_state', 'local_orig', 'local_resp', 'history', and 'label'. The columns 'ts', 'id.orig_p', 'id.resp_p', 'missed_bytes', 'orig_pkts', 'orig_ip_bytes', 'resp_pkts', and 'resp_ip_bytes' are all numbers (either 'float64' or 'int64'). The results show that there are no missing values. It has a lot of different values and possible outliers in columns like 'ts', 'orig_pkts', 'orig_ip_bytes', 'resp_pkts', and 'resp_ip_bytes'. The picture below i.e Figure 2, shows the value counts for all the characteristics in the raw dataset.

The dataset is highly imbalanced. For example, "PartOfAHorizontalPortScan", "Okiru", "Benign" and "DDoS" are the most common labels, while others have very few samples. A lot of scan traffic is likely since it is heavily biased toward 'S0' (Connection attempt seen, no reply). It has a mix of numeric and object data types and there are no missing values.

2. **Correlation analysis:** A heatmap of numeric attributes revealed strong positive correlations between packet counts and corresponding byte counts (orig_pkts–orig_ip_bytes, resp_pkts–resp_ip_bytes) [3].

   From the heatmap of the raw dataset, it is observed that there are strong positive correlations between orig_pkts and orig_ip_bytes, as well as between resp_pkts and resp_ip_bytes. The number of packets is directly related to the total number of bytes sent. There are moderate positive correlations between ts and both id.orig_p and id.resp_p, which suggests that port activity follows a pattern based on time.

   A moderate negative correlation between id.orig_p and id.resp_p, on the other hand, means that certain origin ports likely to talk to some response ports. Also, missed_bytes is weakly positively correlated with resp_pkts and resp_ip_bytes, which suggests that connections with more lost bytes likely to have more response traffic. Most other qualities don't have strong or weak correlations, which suggests that linear relationships aren't very strong.

3. **Categorical analysis**: The dataset doesn't have an even number of protocols; TCP is the most popular, followed by UDP and ICMP. There were no identifiers with a lot of varied values, Such as IP addresses or unique IDs, in the ML modelling.

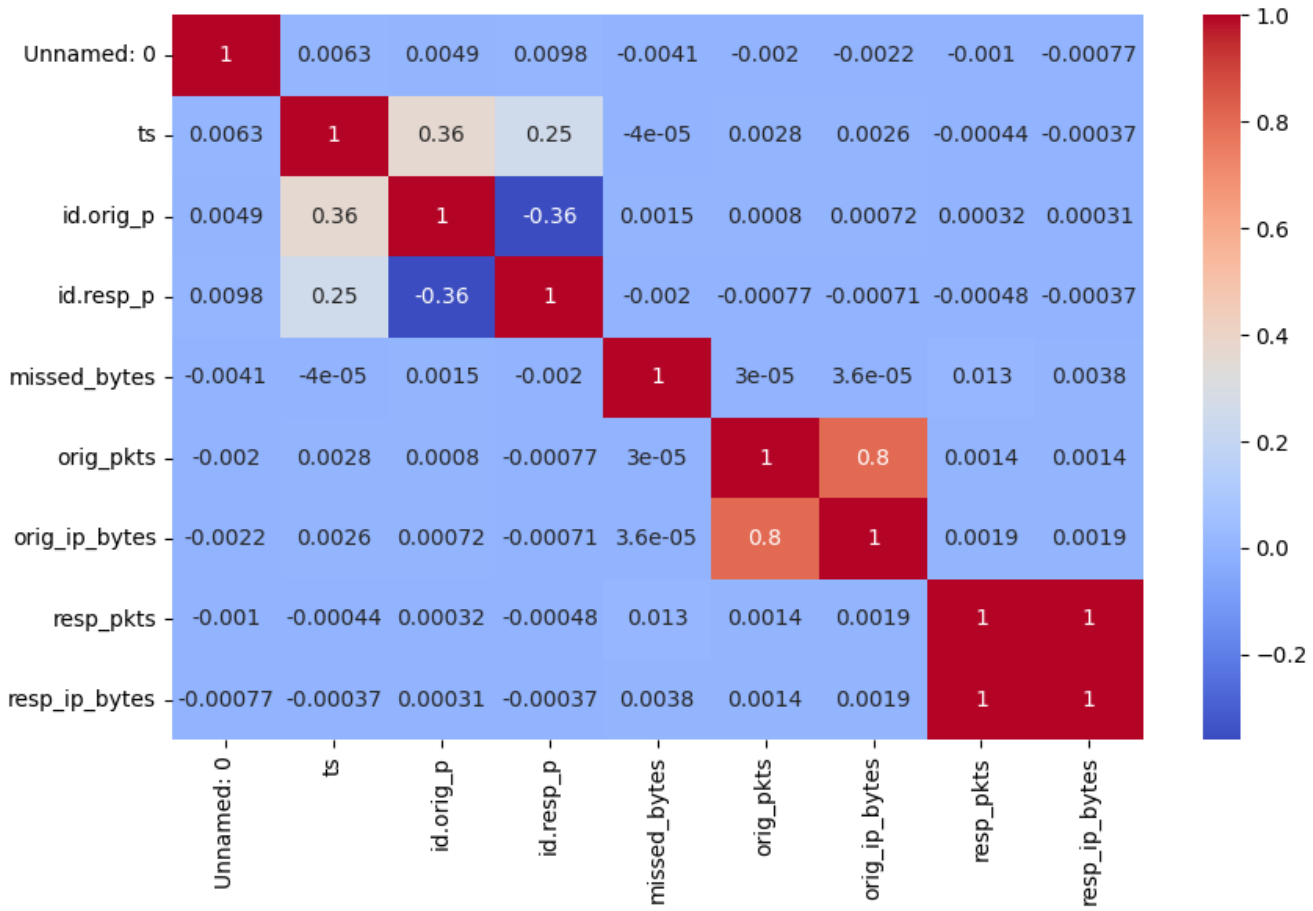

Fig 3. Correlation Heatmap of the IoT -23 Dataset Features

The categorical feature analysis shows that 'uid', 'id.orig_h', and 'id.resp_h' have a lot of unique values, which need to be pre-processed before modelling. Columns like 'proto', 'service', 'conn_state', 'history', and 'label' don't have many unique values, thus they are good category characteristics for classification. Label is especially important because it is the target variable for intrusion detection. 'duration', 'orig_bytes', and 'resp_bytes' are kept as object types, but they really hold numeric data and need to be translated for analysis. The 'local_orig' and 'local_resp' columns each contain only one unique value, which doesn't give us any helpful information, so we can get rid of them. The protocol distribution plot at Figure 4 shows that "tcp" makes up most of the dataset, followed by "udp" and a few "icmp" records. This demonstrates that there is an imbalance between the classes that should be taken into account while training the model.

4. **Outlier detection**: Boxplots identified extreme values in timestamp-based features, confirming the presence of bursts of traffic.

   The box plot of the "ts" (timestamp) feature reveals that most of the data points are close together in a tight interquartile range. This means that most of the timestamps are in a similar time window. The fact that there are spots outside of the whiskers on both ends could mean that there are outliers, or timestamps that are far earlier or later than the rest. These can be infrequent but real events or data problems that need more research before they can be used.

5. **Label Distribution**

Shows the count of each unique label in the 'label' column and provides insight into the distribution of different types of network traffic within the dataset.

The distribution is uneven. 'PartOfAHorizontalPortScan' or 'PortScan', is the most common label and makes up a large part of the samples. The next most common labels are Okiru, Benign, and DDoS, but they don't contain as many cases as

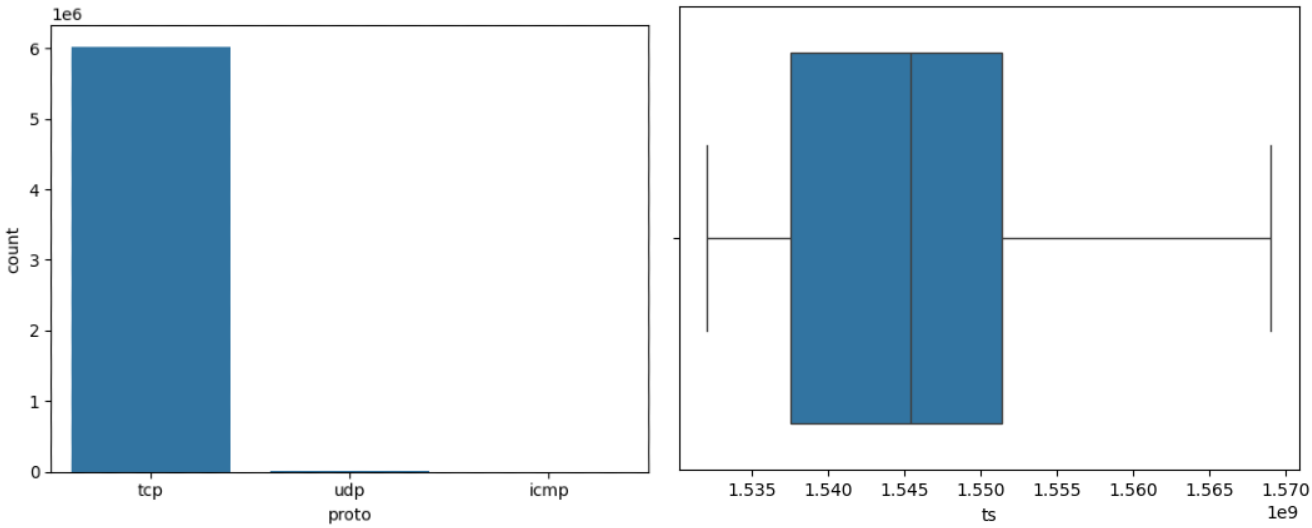

Fig 4. Categorical Analysis & Outlier Detection

Table 3. Label Distribution

Label	Count (Nos.)
PartOfAHorizontalPortScan	3389036
Okiru	1313012
Benign	688812
DDoS	638506
C&C	15286
C&C-HeartBeat	1332
Attack	538
C&C-FileDownload	46
C&C-Torii	30
FileDownload	13
C&C-HeartBeat-FileDownload [21]	8
Okiru-Attack	3
C&C-Mirai	1

'PartOfAHorizontalPortScan'. There are very few examples of the labels [22] 'C&C', 'C&C-HeartBeat', 'Attack', 'C&C-Torii', 'C&C-HeartBeat-FileDownload', and 'FileDownload' [23]. This significant difference in the number of classes makes it hard to train a ML model since models trained on data like this tend to favor the majority class and do poorly on the minority class.

2.3 Data Preprocessing

The raw dataset was processed as under:

1. Type conversion: The columns 'duration', 'orig_bytes' and 'resp_bytes' were converted from strings to numbers.
2. Choosing features: Constant characteristics (like 'local_orig' and 'local_resp') and unique identifiers (like 'uid' and 'IP addresses') were taken out to remove duplicacy and overfitting.
3. Label encoding: The label column was split into multiple groups for supervised learning: benign and other different attacks.
4. Class imbalance: Employed resampling methods like SMOTE oversampling and undersampling to make sure that the class distributions were even for model training.

Handling imbalances:

A hybrid data balancing technique was employed to rectify the significant class imbalance in the dataset. Integration of SMOTE (Synthetic Minority Oversampling Technique) and Random Under-Sampling[24] ensured SMOTE had enough samples to make synthetic data and less common attack types were grouped together into larger groups. For example, "C&C" variants were grouped together and "Okiru-Attack" and "FileDownload" were grouped together into the "Attack" group. Thus, we got rid of very small classes and added samples to minority classes with SMOTE until they had a certain number. Random under-sampling was used to bring the number of samples for large classes like "PartOfAHorizontalPortScan," "Okiru," "Benign," and "DDoS" down to about 100,000 each.

As a result of the balancing, the dataset had even label distribution, which made rare attack classes better represented and made big ones less dominant. The new correlation heatmap showed that constant columns like local_orig and local_resp only had NaN correlations, which proved that they could be removed. This balanced dataset was then used to train models to find less common sorts of IoT attacks more quickly.

```
X_train, X_test, y_train, y_test, scaler

Loading data...

Preprocessing data...

Selecting features...

Original Label Distribution:
label
PartOfAHorizontalPortScan     339334
Okiru                         131065
Benign                         68589
DDoS                           63927
C&C                             1532
C&C-HeartBeat                    139
Attack                            63
C&C-Torii                          4
C&C-FileDownload                   4
C&C-HeartBeat-FileDownload         2
FileDownload                       2
C&C-Mirai                          1
Name: count, dtype: int64
```

```
Other Attack                       551
Name: count, dtype: int64

Balancing training data...
Balancing strategy: Oversampling ['C&C', 'Ot
Warning: Non-numeric columns found in X befo

Balanced Training Label Distribution:
PartOfAHorizontalPortScan    2711229
Benign                        100000
C&C                           100000
DDoS                          100000
Okiru                         100000
Other Attack                  100000
Name: count, dtype: int64

Test Label Distribution (before balancing):
PartOfAHorizontalPortScan    677807
Okiru                        262603
Benign                       137763
DDoS                         127701
C&C                            3341
Other Attack                    110
Name: count, dtype: int64
```

Fig 5. Preprocessing & Balancing

The balancing process fixed the dataset's severe class imbalance, making it easier to train classification models. The correlation analysis on the balanced data at Figure 6, shows that the relationships between the numeric features stayed mostly the same.

Correlation Heatmap Analysis after Balancing:

2.4 Feature Engineering

These feature engineering methods were used on the data set:

Columns like duration, orig_bytes, and resp_bytes were changed to numbers, and changed any missing values to 0. Converted nominal category names like proto, service, conn_state, local_orig, local_resp, and history into numbers. A custom function was used for changing the IP address fields (id.orig_h and id.resp_h) from strings in IPv4/IPv6 format to numbers by converting them to integers or hashing them for formats that aren't standard. Used a RandomForestClassifier and RFE to find the most important features by removing the least important ones one by one. ts (timestamp) keeps track of patterns of attacks that happen over time.

Network Identifiers like id.orig_h, id.orig_p, and id.resp_p show the ports and hosts that are the source and target. Connection attributes like conn_state and history show how often connections work or don't work.

The orig_pkts, orig_ip_bytes, and resp_pkts are numbers that show how many packets and bytes are coming from regular or attack traffic. PCA was used on RFE-selected features to lower dimensionality and multicollinearity while keeping most of the data's variance. This made the model more efficient.

In general, this technique turned raw network data into a small, useful set of temporal, behavioral, and structural properties that are necessary for telling the difference between good and bad network activity.

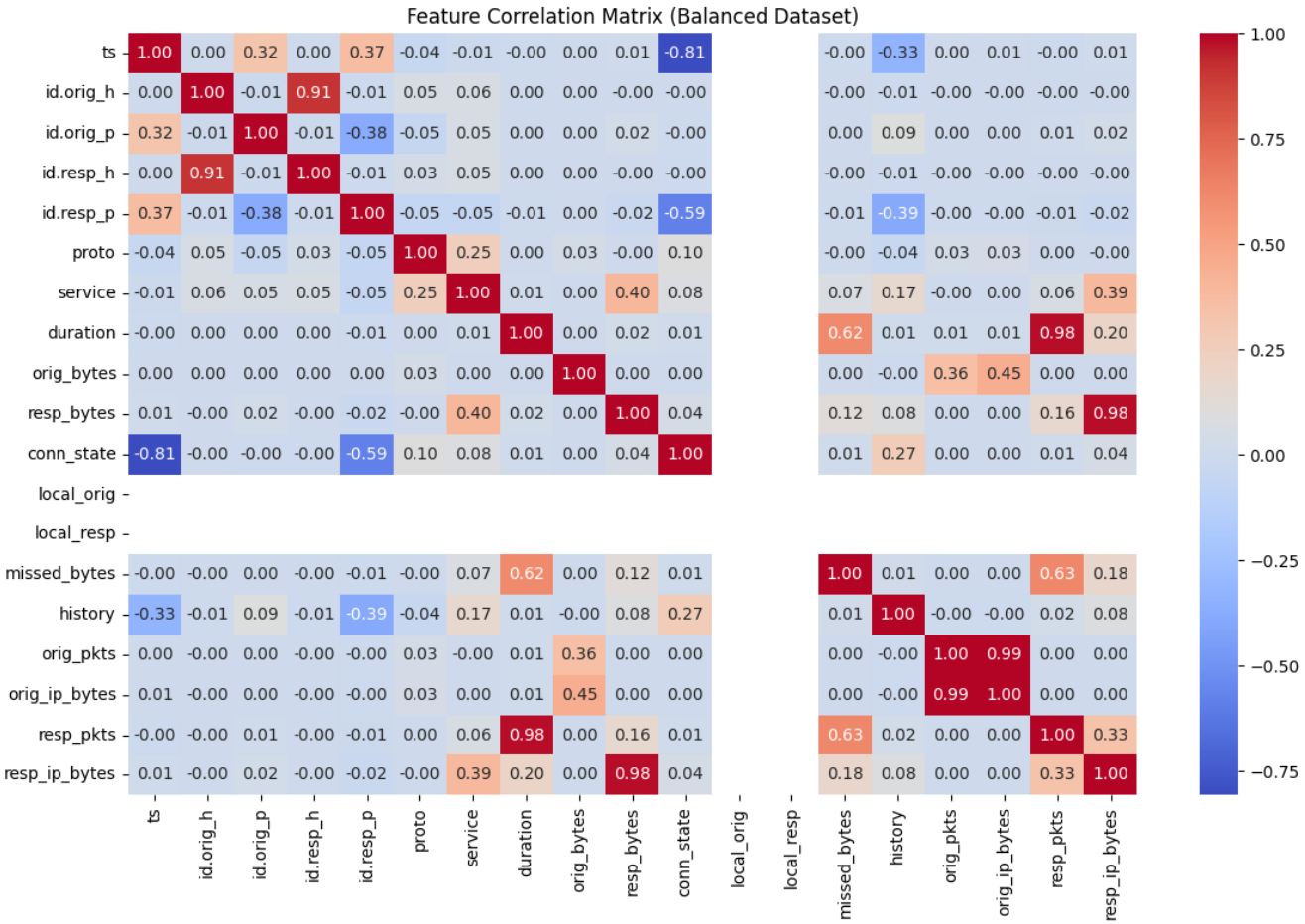

Fig 6. Correlation Heatmap Analysis after Balancing

2.5 Classical Supervised ML Model Development – Model Training

Several ML classifiers were implemented to evaluate detection performance:

1. Random Forest (RF)
2. Support Vector Machine (SVM)
3. Logistic Regression (LR)
4. Naïve Bayes (NB)
5. Decision Tree (DT)
6. K-Nearest Neighbors (KNN)

The models were trained using Stratified K-Fold Cross-Validation to maintain balanced class distributions across folds.

2.6 Model Evaluation

Model performance was assessed using multiple evaluation metrics relevant to intrusion detection:

- Accuracy: Overall correctness of classification.
- Precision, Recall, and F1-score: To account for class imbalance and minimize false alarms.
- Confusion Matrix: For detailed error analysis across classes.
- ROC Curve and AUC: To evaluate the trade-off between true positive and false positive rates.

Performance of the ML models was evaluated using the following evaluation metrics [25] referred as under: -

Accuracy: This is the proportion of all records that are correctly classified as either positive or negative.

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

Precision The ratio of correctly categorized positive records to all negatively classified records, either correctly classified as positive or mistakenly labelled as negative.

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

Recall Rate : The ratio of accurately categorized positive records to all negatively classified records, either correctly labelled as positive or mistakenly classed as negative.

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

F-Measure: It provides a more accurate assessment of the performance of an unbalanced dataset and is the harmonic mean of precision and recall.

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

False Positive Rate: The percentage of negative records that are accurately classified is indicated.

$$\text{False Positive Rate} = \frac{FP}{FP+TN} \tag{5}$$

Where:

- True Positives, or TPs, show how many positive cases were correctly recognized.
- TN (True Negative) shows how many negative cases were successfully detected.
- False Positive, or FP, is a measure of how often negative cases are mistakenly labelled as positive.
- False Negative, or FN, is a measure of how often positive cases are mistakenly labelled as negative.

Special emphasis was placed on reducing **false negatives**, as undetected attacks pose the greatest threat to IoT security.

2.7 Experimental Environment Setup:

The study used Python with libraries like TensorFlow and Scikit-learn. The experiments were performed on Google Colab with the following configuration:

Table 4. Environment set up configuration and specification

Hardware / Software	Specifications
Operating System	Linux, Release: 6.1.85+, Version: #1
CPU	Physical Cores: 1, Total Cores: 2, Frequency: 2200.13Mhz
Machine & Processor	x86_64
Memory	12.67 GB
Development environment	Python Version: 3.11.12, Google Colab
Matplotlib	Version: 3.10.0
NumPy	Version: 2.0.2
Pandas	Version: 2.2.2
Scikit-learn	Version: 1.6.1
Keras	Version: 3.8.0
TensorFlow	Version: 2.18.0

3 Results and Discussion

Multiple supervised ML models were trained & tested using real-world datasets like IoT-23 samples to see how well they worked. To find botnets in IoT networks, people utilize models like SVM, KNN, DT, RF, Logistic Regression, and NB. These algorithms learn from data that has been marked as either bad or good traffic. The supervised ML methods employed in this work are as follows:

- **Support Vector Machines (SVM):**

SVM [26] helps find botnet activity by using a hyperplane to separate data into different groups. It is helpful when working with data that has a lot of dimensions and network traffic patterns that are hard to understand. It has been used to find IoT botnets, which are a big danger to the security of IoT networks [27], [28], [29], [30].

- **Decision Trees (DT):**

DTs [26] put communication packets into a tree-like structure to make them easier to interpret. They are flexible ML models that may be used for both classification and regression tasks, such as finding IoT botnets. The algorithm splits the input data into smaller groups based on decision criteria over and over again until each group only has one class or value [31], [32].

- **Random Forest (RF):**

RF [26] is a strong ML which has potential for finding IoT botnet assaults. It is quite good at figuring out if network traffic is real or fake by looking at things that show how IoT devices act and how they talk to one other [33]. This algorithm can find patterns that are signs of Botnet attacks and is a good way to find and stop them [34]. RF uses more than one decision tree to make predictions more accurate.

- **Naive Bayes (NB):**

This probabilistic classifier is a quick and easy way to find botnets. It often works with other algorithms, such as SVM and RF. People often employ NB [10] for large datasets. This algorithm can analyze quickly and effortlessly. It has high expectations for the features to have their own worth. Problem with this approach is that it assumes that all the features are separate from each other. It will also presume that values not in the training data are none, which means it will treat them as having zero probability. So, this approach might not work with this IoT-23 dataset.

- **Logistic Regression (LR):**

This method [26] examines the characteristics of a network packet to guess how likely it is that it is a botnet attack. This method for finding botnets early on is easy to use and works. It is preferred for its simplicity, efficiency, and clarity [35].

- **K-Nearest Neighbor (KNN):**

KNN [26] is a ML method that is often used for classification and regression problems [36]. This algorithm uses the similarity principle to group instances based on how similar their attributes [26] are to those of known instances. In KNN, the class or value of an instance is based on the k nearest neighbors. KNN has been used for classification tasks based on network traffic data in the context of IoT botnet detection [37]. The effectiveness of KNN in botnet identification is influenced by the selection of features, the k value, and the dataset size [38], [39]. For the KNN model, the number of neighbors (k) was set to the default value used by sklearn.neighbors.KNeighborsClassifier(), which is k=5.

3.1 Performance Metrics Comparison:

The figures below show how the accuracy and execution time of several supervised ML algorithms. RF (99.99%) and DT (99.99%) had the best accuracies, showing that tree-based models are quite good at finding patterns in the balanced dataset that was reduced (RFE+PCA). KNN (99.99%) also did almost as well, which shows that distance-based classifiers operate effectively on data that is well-balanced and has been feature-engineered. NB (72.65%) had a rather poor accuracy, which was much lower than RF/DT/KNN. SVM (64.31%) and LR (64.31%) had rather low accuracy on the complete dataset, which was much lower than RF/DT/KNN.

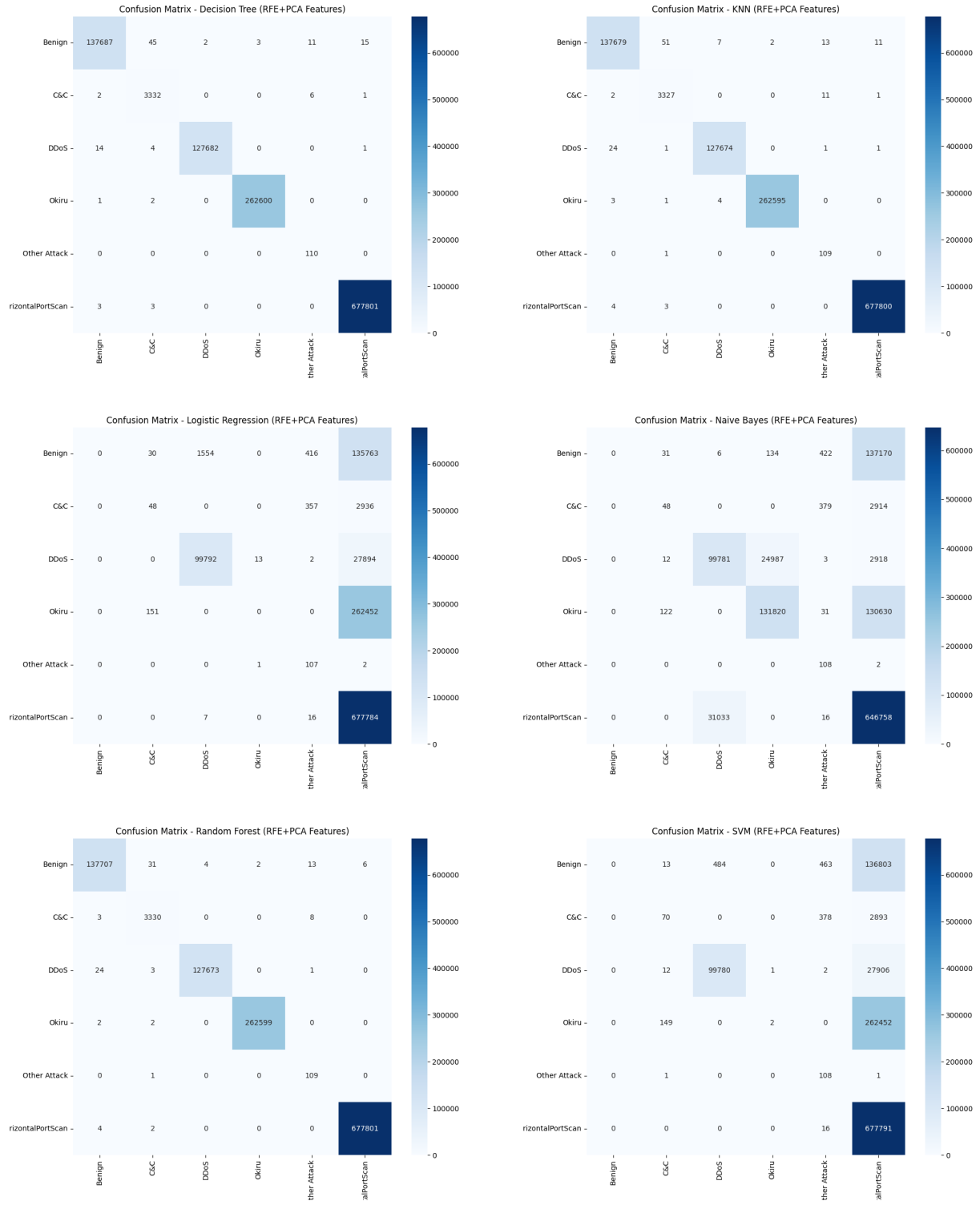

Fig 7. Confusion Matrix of All Models

Table 5. Comparative Performance Metrics

#	Model	Test Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	train-ing_time (s)	predic-tion_time (s)	ROC
1	SVM	64.31%	71.82%	64.31%	51.81%	1.1E-05	0.11363	0.8683
2	KNN	**99.99%**	99.99%	99.99%	99.99%	12.1361	**9.04781**	0.9998
3	DT	**99.99%**	99.99%	99.99%	99.99%	29.8437	**0.08371**	0.9997
4	RF	**99.99%**	99.99%	99.99%	99.99%	553.08	**1.38471**	0.9999
5	NB	72.65%	65.74%	72.65%	67.17%	2.28541	0.34929	0.9286
6	LR	64.31%	44.78%	64.31%	51.78%	20.5098	0.04934	0.869

#	Model	macro_ sensitivity	macro_ specificity	Mean Absolute Error (MAE)	Matthews Correlation Coefficient (MCC)	Kappa
1	RF	0.53619	0.99998	0.00074	0.99954	0.99954
2	DT	0.51048	0.99997	0.00088	0.99949	0.99949
3	NB	0.32588	0.95717	1.55512	0.55827	0.50702
4	LR	0.1943	0.93769	1.79816	0.3935	0.25366
5	SVM	0.19627	0.93765	1.79959	0.39346	0.25319
6	KNN	0.53745	0.99997	0.00099	0.99945	0.99945

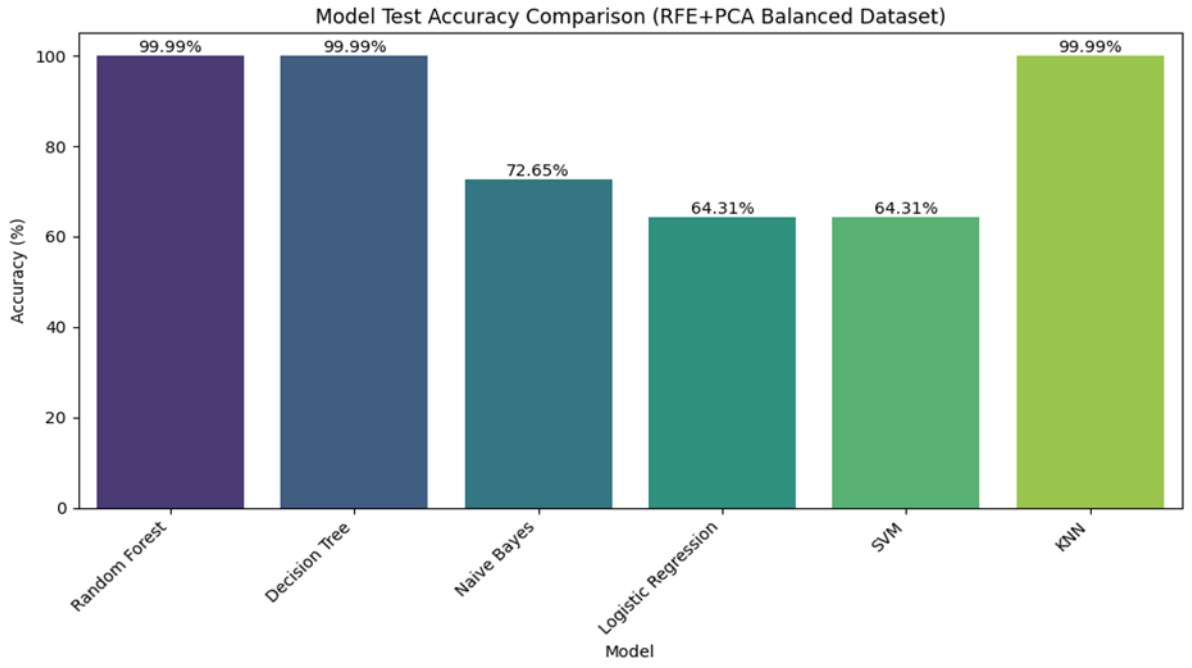

Fig 8. Model Accuracy Comparison

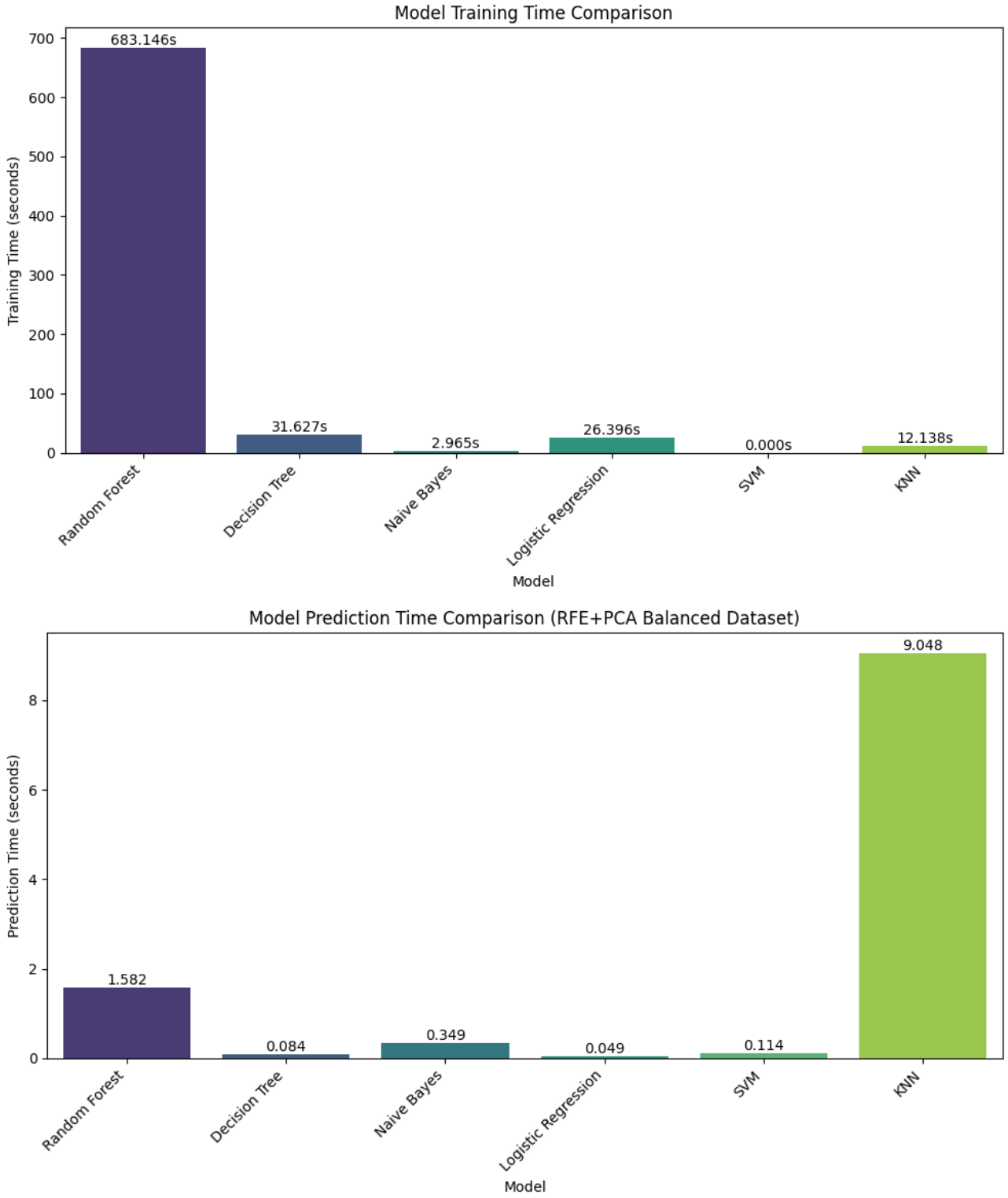

Fig 9. Model Training Time vs Prediction Time

3.2 Training & Prediction Times (RFE+PCA Balanced Dataset)

The amount of time it takes to train each model is a significant aspect, especially when working with big datasets or when thinking about how long it would take to retrain models when new data comes in. The training times seen in this experiment (after balancing the data and engineering the features) are as follows:

RF took the longest to train. For DT, the training time was much less than that of RF. In KNN, the training time was average. In the case of LR, it took a reasonably short time to train. Logistic. NB is one of the models that trained the fastest. SVM (LinearSVC) took very short training time.

Prediction time or inference time is an important measure for real-time intrusion IDS since it tells us how quickly the model can sort through new network traffic. Logistic Regression and SVM (LinearSVC) models had the shortest times for making predictions. The forecast time for DT was relatively quick, though a little slower than the linear models. It took a moderate amount of time to make a forecast in the case of NB. KNN model took the longest to make predictions of all the ones we looked at.

3.3 F1 Scores (RFE+PCA Balanced Dataset)

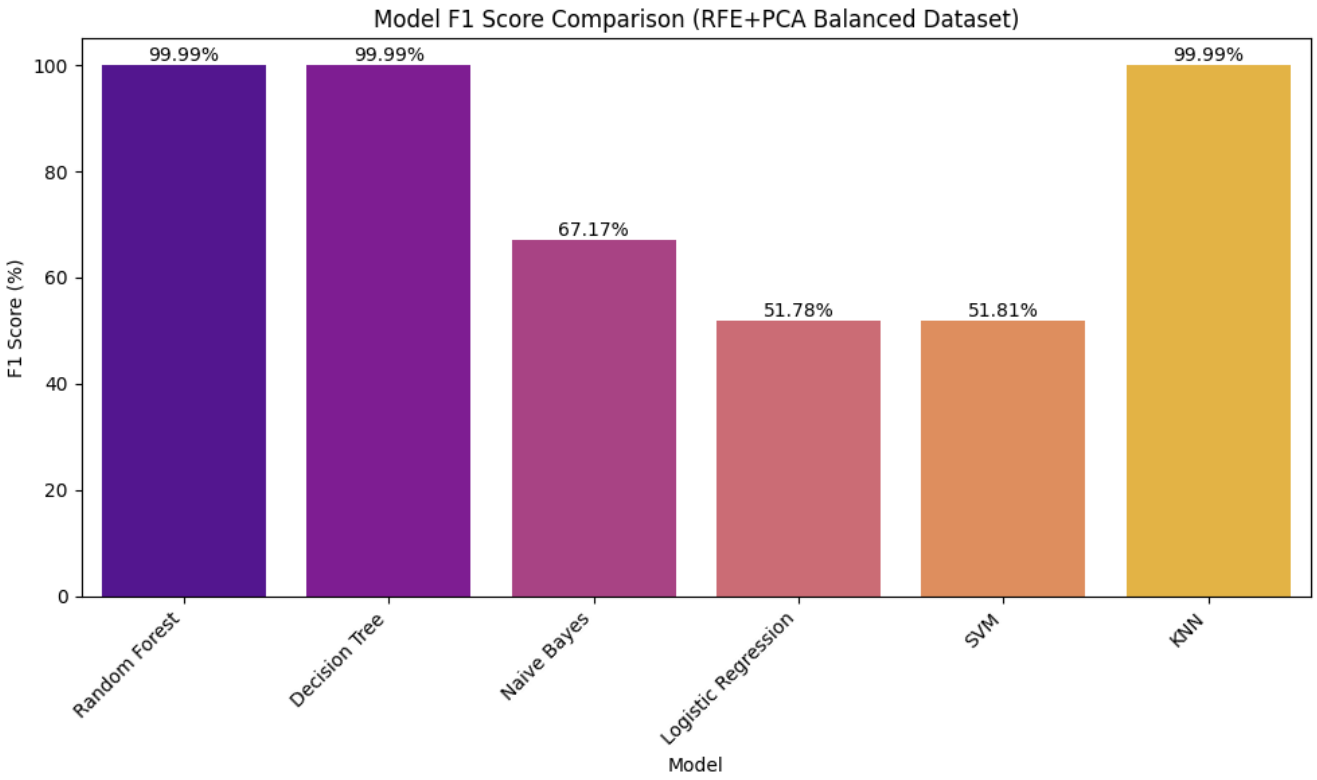

Fig 10. F1 Score Comparison

The F1-score is a good way to check how well a model gets both recall and precision right. This is important for datasets that are not evenly split. Let us look at the Balanced Dataset with PCA and RFE:

RF: Very good precision and recall, which means it can find different types of attacks very well. In case of DT, works very well after balancing the data, just like RF. For KNN, the F1-score is almost the same; it works best with data that is balanced and voting based on neighbors.

Logistic Regression doesn't work well with non-linear patterns, which makes the trade-off between precision and recall worse. SVM (LinearSVC) works like Logistic Regression, but because it is linear, it is hard to use with complicated data.

RF, DT, and KNN have the highest F1 scores for finding intrusions. These models are the best at getting the right answers and remembering them. Tree-based models are better for this job than NB, Logistic Regression, and Linear SVC.

3.4 ROC Curve Comparison

The plot above displays the macro-average ROC curve for each model that was tested on the test dataset. The X-axis (False Positive Rate - FPR) shows the percentage of negative cases that are wrongly labeled as positive. A lower FPR is desirable. Y-axis (True Positive Rate - TPR) or Recall: This shows how many positive cases are correctly identified as positive. A greater TPR is better.

The Perfect Curve: A perfect model would have a curve that goes straight up from (0,0) to (0,1) and then straight across to (1,1). This means that the TPR is 1 and the FPR is 0, which means that the categorization is perfect. The Random Guess Line: The dashed black line from (0,0) to (1,1) shows a random guess classifier. If a model did no better than random chance, its ROC curve would be close to this line, and its AUC would be 0.50. AUC (Area Under the Curve. A model is usually better if it has a higher AUC.

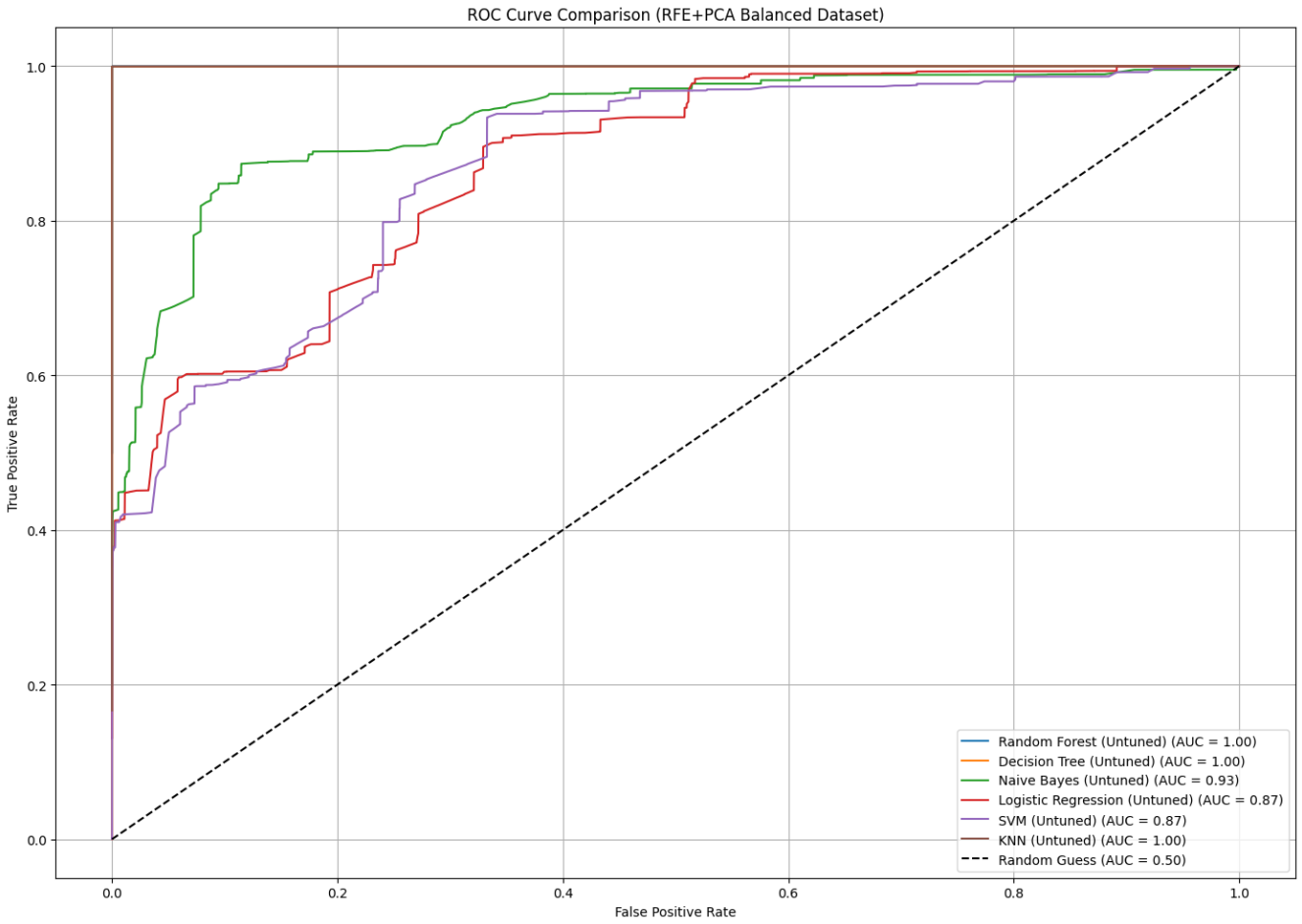

Fig 11. ROC Curves of all Models

Observations from the plotted ROC curves:
Models that work well (AUC near to 1.00):

- RF: With an AUC of 0.9999, it does very well.
- DT: It also has an AUC of 0.9997, which is quite good.
- KNN: Has an AUC of 0.9998, which means it works quite well.

Models that don't work as well (AUC much less than 1.00):

- Logistic Regression: With an AUC of 0.8690, it does far better than random chance but not as good as the tree-based models, KNN, or SVM.

- NB: This model has the lowest AUC of 0.9286, which means it did the worst job.
- SVM: Has an AUC of 0.8683, which is good but not quite as good as the best tree-based models and KNN.

The ROC curve comparison clearly shows that the RF, DT, and KNN models did better on this dataset, even without tweaking the hyperparameters. The SVM, Logistic Regression, and NB models are better than guessing at random, but they don't work as well for this multi-class intrusion detection job with the present data and preprocessing. The high AUC ratings (near to 1.00) for RF, DT, and KNN show that these models can tell the different classes very well [40]. The comparative assessment of classical supervised ML models on the IoT-23 dataset has revealed significant variation in accuracy and latency, as well as in balanced performance indicators, including F1-score, True Positive Rate (TPR/Recall), and False Positive Rate (FPR). We can compare the models and find the best one for this intrusion detection task by looking at their overall performance on different metrics (Accuracy, Precision, Recall, F1-score, MAE, MCC, Kappa), training time, and prediction time [41].

Random Forest (RF) consistently achieved near-perfect results across all metrics (Accuracy, F1-score, MCC, Kappa, Sensitivity, Specificity), demonstrating excellent classification capability. Decision Tree (DT) worked almost exactly like RF, finding both positive and negative cases correctly. KNN got similar scores, proving that it works well on balanced data. Naïve Bayes (NB), Logistic Regression (LR), and SVM (LinearSVC), on the other hand, had much lower accuracies and F1-scores and higher MAE, which means they didn't generalize as well.

SVM and NB were the fastest to train, LR and DT were in the middle, and RF and KNN were the slowest. LR and SVM were the fastest at predicting latency, DT and NB were average, and KNN and RF were the slowest.

DT is the best option because it has the best balance of accuracy and efficiency. It has a high detection precision and recall (F1 $\approx$ 0.9999) and a low prediction delay (0.0837 s). So, DT is the best model for finding IoT botnets in real time because it strikes the best balance between speed and accuracy for edge-based deployments.

Comparison of the work with other works published by different authors are as under:

Table 6. Comparison of Existing Works vs. Present Work

Sl. No	Author(s)	Dataset(s)	Algorithm(s)	Multiclass Accuracy
01	Nicho et al. [4]	IoT-23	CNN	96%
02	Mishra et al. [12]	IoT-23, BoT-IoT, UNSW-NB15, ToN_IoT	GAN + BiLSTM Ensemble	99.08%
03	Kumari et al. [13]	IoT-23	ML Classifiers	99.99%
05	Alanazi et al. [14]	IoT-23	Ensemble + Feature Selection	99.984%
06	Tarik Himdi et. al. [11]	IoT-23	CNN, LSTM, and GRU	95%
07	Farooqi et al. [3]	IoT-23 [12% of dataset]	ML Classifiers	DT 99.99% detection time 5.2 seconds
08	Maran et al. [10]	Dataset Generated from IoT Devices	ML Classifiers	RF 96%
09	Venu et al. [1]	IoT-23	ML Classifiers	DT 93%-94%
10	Ullah et al. [15]	IoT-23, BoT-IoT, CICIOT2023	CNN–BiLSTM–RF–LR Hybrid (Soft Voting Ensemble)	BoT-IoT: 100%; CICIOT2023: 99.2%; IoT-23: 91.5%
11	Anshika Sharma et al. [16]	IoT-23	RF, XGB, KNN	RF: 89%; XGB: 88%; KNN: 85%
12	Krzysztoń et al. [17]	IoT-23, CIC-IoT 2023	RF, Autoencoder, Isolation Forest	RF: 99.16–100%; Autoencoder: 98.76–99.27%
13	Ullah et al. [15]	IoT-23, BoT-IoT, CICIOT2023	CNN–BiLSTM–RF–LR Hybrid (Soft Voting Ensemble)	BoT-IoT: 100%; CICIOT2023: 99.2%; IoT-23: 91.5%
14	This Work	IoT-23 full dataset	PCA+RFE Feature Selection, SMOTE & Classical ML classifiers	**99.99%** [prediction time of DT **0.083** seconds]

The work in this article describes a dual-stage feature optimization pipeline that uses PCA to reduce the number of dimensions and RFE to choose the best features. This pipeline makes sure that there is as little redundancy as possible and as much discriminative power as possible. Also, adding SMOTE for synthetic minority oversampling fixes the problem of class imbalance, which is widespread in IoT traffic data.

The proposed framework achieves high multiclass accuracy (99.99%) and significantly reduces prediction latency (DT: 0.0837 seconds) by applying preprocessing improvements on classical ML classifiers. This combination of accuracy, stretch of identification, and low latency makes the framework a new, lightweight and deployable way to find IoT botnets in real time.

4 Conclusion

This study conducted a detailed comparison of six classical ML algorithms on the IoT-23 dataset (6.04 million samples, 23 scenarios). The results revealed that the DT classifier achieved Accuracy = 99.99%, Precision = 99.99%, Recall = 99.99%, and F1-score = 99.99% with a prediction latency of only 0.0837 s, outperforming other models. RF achieved the same accuracy and a ROC-AUC of 0.9999, it required 553.08 s to train, making it less practical for real-time detection. KNN matched DT's accuracy but incurred a prediction delay over 9 s, unsuitable for latency-sensitive IoT contexts. In contrast, SVM (64.31%), NB (72.65%), and LR (64.31%) displayed limited generalization capabilities.

Comparatively, the proposed framework has shown better results than previous studies by utilizing 100% of the IoT-23 dataset, using PCA + RFE feature selection and SMOTE balancing. The result is a 0.91% – 6.99% accuracy improvement and a faster detection latency than deep learning counterparts.

In summary, the DT model provides an optimal balance between detection accuracy and computational efficiency, making it well-suited for real-time intrusion detection in resource-constrained IoT networks. Future work may extend this framework toward hybrid ML–DL or context-aware architectures, targeting adaptive learning and energy-efficient edge deployment for next-generation IoT security.

5 Data availability statement

Aposemat IoT-23: A labeled dataset with malicious and benign IoT network traffic. This dataset was created as part of the Avast AIC laboratory with the funding of Avast Software[18]. Dataset used in this work has been downloaded from https://www.kaggle.com/.

No new data were generated.

6 Details of ethical clearance

This research employs the publicly accessible IoT-23 dataset, comprising anonymized and non-identifiable network traffic data, thereby negating the necessity for formal ethical approval. The dataset was used in accordance with open-use academic licensing rules, and all experimental procedures followed ethical research practices to the letter, making sure that privacy and data protection laws were followed.

References

1) Venu N, AK A, ASR A. Botnet Attacks Detection in Internet of Things Using Machine Learning. *NeuroQuantology*. 2022. https://doi.org/10.14704/nq.2022.20.4.NQ22298.
2) Garg S, Kumar SRPV. Identification of Internet of Things (Iot) Attacks Using Gradient Boosting: A Cross Dataset Approach. *TELEMATIQUE*. 2022. Available from: https://www.provinciajournal.com/index.php/telematique/article/view/965.
3) Farooqi AH, RA R, SK S. ML-Driven Lightweight Botnet Detection System for IoT-Networks. *International Journal of Innovations in Science and Technology*. 2024. Available from: https://journal.50sea.com/index.php/IJIST/article/download/1107/1643/10941.
4) Nicho M, Cusack B, McDermott CD, Girija S. Assessing IoT intrusion detection computational costs when using a convolutional neural network. *Information Security Journal: A Global Perspective*. 2025;34:471–491. https://doi.org/10.1080/19393555.2025.2496327.
5) Meng X, Spanoudakis G. MBotCS: A Mobile Botnet Detection System Based on Machine Learning. 2016. http://dx.doi.org/10.1007/978-3-319-31811-0_17.
6) Chen R, Niu W, Zhang X, Zhuo Z, Lv F. An Effective Conversation-Based Botnet Detection Method. *Math Probl Eng*. 2017. https://doi.org/10.1155/2017/4934082.
7) Hussain F, Abbas SG, Fayyaz UU, Shah GA, Toqeer A, Ali A. Towards a Universal Features Set for IoT Botnet Attacks Detection. IEEE. 2020. https://doi.org/10.1109/INMIC50486.2020.9318106.
8) Popoola SI, Adebisi B, Ande R, Hammoudeh M, Anoh K, Atayero AA. SMOTE-DRNN: A Deep Learning Algorithm for Botnet Detection in the Internet-of-Things Networks. *Sensors*. 2021;21:2985. https://doi.org/10.3390/s21092985.
9) Alothman Z, Alkasassbeh M, Baddar SAH. An efficient approach to detect IoT botnet attacks using machine learning. *Journal of High Speed Networks*. 2020;26:241–254. https://doi.org/10.3233/JHS-200641.
10) Maran P, Yap TTV, Chin JJ, Ng H, Goh VT, Kuek TY. Comparison of Machine Learning Models for IoT Malware Classification. Dordrecht. Atlantis Press International BV. 2022. https://doi.org/10.2991/978-94-6463-094-7_3.
11) Himdi TT, Ishaque M. DEEP LEARNING-ENHANCED ANOMALY DETECTION FOR IOT SECURITY IN SMART CITIES. *ARPN Journal of Engineering and Applied Sciences*. 2024. https://doi.org/10.59018/032456.

12) Mishra AK, Paliwal S, Srivastava G. Anomaly detection using deep convolutional generative adversarial networks in the internet of things. *ISA Trans*. 2024;145:493–504. https://doi.org/10.1016/j.isatra.2023.12.005.
13) Kumari P, Mangat V, Singh A. Comparative Analysis of State-of-the-Art Attack Detection Models. IEEE. 2023. https://doi.org/10.1109/ICCCNT56998.2023.10306428.
14) Alanazi M, Aljuhani A. Anomaly Detection for Internet of Things Cyberattacks. *Computers, Materials & Continua*. 2022;72:261–279. https://doi.org/10.32604/cmc.2022.024496.
15) Ullah S, Wu J, Lin Z, et al. Comparative analysis of deep learning and traditional methods for IoT botnet detection using a multi-model framework across diverse datasets. *Sci Rep*. 2025;15:31072. https://doi.org/10.1038/s41598-025-16553-w.
16) Sharma A, Babbar H. Understanding IoT-23 Dataset: A Benchmark for IoT Security Analysis. Bangalore, India. 2024. https://doi.org/10.1109/CONIT61985.2024.10627334.
17) Krzysztoń E, Rojek I, Mikołajewski D. A Comparative Analysis of Anomaly Detection Methods in IoT Networks: An Experimental Study. *Applied Sciences*. 2024;14(24):11545. https://doi.org/10.3390/app142411545.
18) Garcia APMJES. IoT-23: A labeled dataset with malicious and benign IoT network traffic. 2020. http://doi.org/10.5281/zenodo.4743746.
19) Miettinen M, Marchal S, Hafeez I, Asokan N, Sadeghi AR, Tarkoma S. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. IEEE. 2017. https://doi.org/10.1109/ICDCS.2017.284.
20) Jain A, Bagoria R, Arora P. An intelligent zero-day attack detection system using unsupervised machine learning for enhancing cyber security. *Knowl Based Syst*. 2025;324:113833. https://doi.org/10.1016/j.knosys.2025.113833.
21) Alrefaei A, Ilyas M. Using Machine Learning Multiclass Classification Technique to Detect IoT Attacks in Real Time. *Sensors*. 2024;24:4516. https://doi.org/10.3390/s24144516.
22) Ahmed N, Zahran B, Ayyoub B, Alzoubaidi AR, Ngadi MA. Intrusion Detection System Using Hybrid Machine Learning Classifiers and Optimum Feature Selection in Internet of Things (IoT). *IRECAP*. 2024;14(2). https://doi.org/10.15866/irecap.v14i2.24944.
23) Teja SR, Janardhana DR. Enhancing Cybersecurity Through Machine Learning-Based Classification of IoT Network Traffic. Kalaburagi, India. 2023. https://doi.org/10.1109/ICIICS59993.2023.10421234.
24) Elreedy D, Atiya AF, Kamalov F. A theoretical distribution analysis of synthetic minority oversampling technique (SMOTE) for imbalanced learning. *Mach Learn*. 2024;113:4903–4923. https://doi.org/10.1007/s10994-022-06296-4.
25) Hodo E, Bellekens X, Iorkyase E, Hamilton A, Tachtatzis C, Atkinson R. Machine Learning Approach for Detection of nonTor Traffic. *Journal of Cyber Security and Mobility*. 2017. https://doi.org/10.1145/3098954.3106068.
26) Nazir A, He J, Zhu N, et al. Advancing IoT security: A systematic review of machine learning approaches for the detection of IoT botnets. *Journal of King Saud University - Computer and Information Sciences*. 2023;35:101820. https://doi.org/10.1016/j.jksuci.2023.101820.
27) Atzori L, Iera A, Morabito G. The Internet of Things: A survey. *Computer Networks*. 2010;54:2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010.
28) Pan Y, Zhang J. Parallel Programming on Cloud Computing Platforms. *Media Watch*. 2012. https://www.researchgate.net/publication/235776354_Parallel_Programming_on
29) Malhotra R, Jain A. Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality. *Journal of Information Processing Systems*. 2012;8:241–262. https://doi.org/10.3745/JIPS.2012.8.2.241.
30) Arowolo MO, Ogundokun RO, Misra S, Oluranti J, Kadri AF. K-Nearest Neighbour Algorithm for Classification of IoT-Based Edge Computing Device. 2022. https://doi.org/10.1007/978-3-030-80821-1_8.
31) Raghavendra M, Chen Z. Detecting IoT Botnets on IoT Edge Devices. IEEE. 2022. https://doi.org/10.1109/ICCWorkshops53468.2022.9814555.
32) Medghaghet S, Sahraoui S. Efficient Machine Learning Technique for Early Detection of IoT Botnets. 2022. https://doi.org/10.1007/978-3-030-90618-4_4.
33) Alazzam A, Alkhatib M, Shaalan K. Artificial Intelligence Chatbots: A Survey of Classical versus Deep Machine Learning Techniques. *Information Sciences Letters*. 2023;12:1217. https://doi.org/10.18576/isl/120437.
34) Chunduri H, Kumar TG, Charan PVS. A Multi Class Classification for Detection of IoT Botnet Malware. .
35) Bapat R, Mandya A, Liu X, et al. Identifying malicious botnet traffic using logistic regression. IEEE. 2018. https://doi.org/10.1109/SIEDS.2018.8374749.
36) Xiong L, Yao Y. Study on an adaptive thermal comfort model with K-nearest-neighbors (KNN) algorithm. *Build Environ*. 2021;202:108026. https://doi.org/10.1016/j.buildenv.2021.108026.
37) Isnain AR, Supriyanto J, Kharisma MP. Implementation of K-Nearest Neighbor (K-NN) Algorithm For Public Sentiment Analysis of Online Learning. *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*. 2021;15:121. https://doi.org/10.22146/ijccs.65176.
38) Kumar A, Lim TJ. A Secure Contained Testbed For Analyzing IoT Botnet Malware. 2019. https://doi.org/10.48550/arXiv.1906.07175.
39) Quek YT, Woo WL, Thillainathan L. IoT Load Classification and Anomaly Warning in ELV DC Picogrids Using Hierarchical Extended ${k}$-Nearest Neighbors. *IEEE Internet Things J*. 2020;7:863–873. https://doi.org/10.1109/JIOT.2019.2945425.
40) Wahib HA, Abdullah MZ, Mohammad AS. Enhancing intrusion detection system for software-defined networks based on machine learning. *Ingénierie des Systèmes d'Information*. 2025;30(4):1015–1025. https://doi.org/10.18280/isi.300418.
41) Singh NJ, Singh KR, Hoque N, et al. Massive IoT network traffic analysis using ML and DL methods: an empirical evaluation. *J Supercomput*. 2025;81:1107. https://doi.org/10.1007/s11227-025-07575-2.