scientific reports

Check for updates

OPEN

Identification and detection of DDoS attack on smart home infrastructure using machine learning models

Thejavathy Vengappa Raja, Zoher Ezziane✉, Jun He, Xiaoqi Ma & Asmau Wali-Zubair Kazaure

This study investigates Distributed Denial-of-Service (DDoS) attack detection within smart home environments using both traditional machine learning and deep learning approaches. Real smart home traffic data, collected approximately 11.5 h of normal and attack activity, was used to implement and evaluate two models: k-Nearest Neighbour (k-NN) and an Artificial Neural Network (ANN). The k-NN model achieved an accuracy of 97.13%, while the ANN achieved 81.7% accuracy under the same dataset conditions. Unlike previous studies relying solely on benchmark datasets, this work uses self-collected smart home data to assess model feasibility and real-world deployment potential.

Keywords Smart homes, Distributed denial of service, Machine learning, Deep learning, Cybersecurity

A smart home, also referred to as a Connected Home, Home Automation, or Intelligent Home, consists of interconnected IoT devices designed to automate tasks and adapt to user preferences[1]. These devices increasingly rely on AI-driven functionality to support personalised household needs and lifestyles[2]. Although smart technologies are used across a wide range of sectors, this study focuses specifically on smart home architecture, where devices such as lights, cameras, and appliances operate within a shared network to improve convenience and efficiency[3].

Cyberattacks on such environments typically compromise one or more core security principles: confidentiality, integrity, or availability[4]. DDoS attacks specifically target availability by overwhelming devices or networks with excessive traffic, rendering them inaccessible to legitimate users (Fig. 1). They are commonly classified into traffic/fragmentation, bandwidth/volume, and application-layer attacks[5]. As these attacks evolve, their traffic patterns increasingly resemble legitimate behaviour, making them harder to detect through traditional signature-based or threshold-based mechanisms.

To address these challenges, AI-based detection approaches have become a major focus of cybersecurity research. Several early-phase AI systems have already demonstrated promising performance in detecting and mitigating DDoS activity[6]. However, attackers continually refine their strategies, making adaptive, intelligent detection methods essential. Modern AI techniques including semi-supervised learning, self-supervised learning, and other advanced methods, enable models to learn from large volumes of unlabelled data, identify hidden structures, and adapt to new or evolving threats without requiring explicit manual labelling[7,8].

AI provides several advantages for cybersecurity and DDoS defence, including its ability to learn continuously over time, detect unknown threats, analyse high-volume network traffic, enhance vulnerability management, provide faster and more accurate detection, support behavioural analytics, maintain strong authentication, and automate complex repetitive monitoring tasks[9–15]. These capabilities make AI particularly suitable for defending resource-constrained environments such as smart homes, where traditional security tools may be too static or computationally demanding.

Given these challenges, this work aims to identify the most effective AI method for detecting and mitigating DDoS attacks within smart homes. We propose a feature-selection approach that improves k-Nearest Neighbour (k-NN) performance for IoT-specific DDoS patterns, achieving over 97% detection accuracy and offering strong potential for real-time deployment on smart home gateways or edge devices.

Department of Computer Science, School of Science and Technology, Clifton Campus, Nottingham Trent University, Nottingham NG11 8NS, UK. ✉email: zoheir.ezziane@ntu.ac.uk

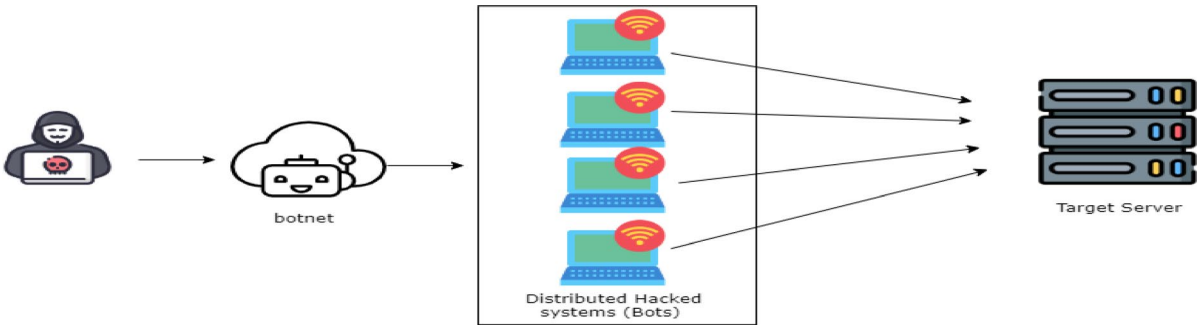


Fig. 1. DDoS attack.

Although DDoS detection has been widely studied in broader IoT and cloud environments, research specifically targeting smart home infrastructures remains limited. Smart homes contain heterogeneous, low-power devices that require lightweight, adaptive security mechanisms capable of operating under constrained resources. Using real network traffic collected from a functional smart home environment, this study compares the performance of k-NN (as a lightweight machine learning technique) and Artificial Neural Networks (ANNs) (as a deep learning approach).

The primary contribution of this work is an empirical evaluation based on real smart-home traffic, addressing a gap often overlooked in existing literature, and answering a practical question: Which AI paradigm is most effective for securing real smart homes with limited data, limited computational resources, and evolving DDoS threats?

Related work

Vulnerabilities and early approaches in IoT DDoS detection

With the rapid growth of IoT ecosystems, smart homes have become increasingly susceptible to Distributed Denial-of-Service (DDoS) attacks. These attacks exploit weak security configurations and limited computational capacity in devices such as cameras, hubs, and sensors, leading to congestion and service disruption. Traditional mitigation approaches including signature-based detection, thresholding, and rule-based filtering, remain effective for known threats but struggle against modern, multi-vector attacks that mimic normal traffic behaviours.

Early work in the field employed conventional machine learning (ML) models such as Support Vector Machines (SVM), Random Forest (RF), Naïve Bayes (NB), and Decision Trees (DT). While these models achieved high accuracies (often > 95%) on benchmark datasets, they typically relied on handcrafted features, centralized processing, and datasets such as CIC-IDS-2017 or NSL-KDD, which do not reflect the nuanced and heterogeneous traffic patterns of smart home networks. Consequently, there remains a need for detection approaches suited to data-constrained, device-diverse, real-time residential environments.

Shift toward deep learning and hybrid AI models

Recent advances have introduced deep learning and hybrid architectures to improve DDoS detection. Wanda and Hiswati[16] developed the Belief-DDoS model using a Deep Belief Network (DBN), achieving improved detection across mixed attack types. Aguru[17] proposed a modified stacked GRU framework for lightweight multi-vector detection in IoT systems. These works highlight the evolution from static matching techniques toward adaptive, data-driven solutions capable of learning complex nonlinear patterns. However, deep learning approaches typically require large, labelled datasets and high computational power, limiting real-world deployment in small-scale smart home environments[16,17].

Federated, edge, and lightweight learning strategies have emerged in response to these constraints. Shirvani (2024) demonstrated that federated learning improves scalability and privacy by enabling collaborative detection without centralized data sharing. Additional work explores compact neural architectures and hybrid ML systems designed to operate efficiently under constrained memory and processing budgets. These developments strongly motivate the present study's comparison of lightweight ML (k-NN) and compact deep learning (ANN) models on real smart home network traffic[18].

Types of DDoS attacks (condensed summary)

Common DDoS techniques include Smurf, UDP flood, HTTP flood, Tear Drop, Ping of Death, SYN flood, and buffer-overflow-based attacks[19]. Although these classical attacks form the foundation of DDoS research, their modern variants often combine multiple vectors and high-intensity flooding strategies.

Modern and AI-assisted DDoS attacks

Modern DDoS attacks frequently exhibit trends such as increasing volumetric intensity and multi-vector coordination, making them harder to detect and mitigate[20]. Recent years have also seen the emergence of AI-assisted DDoS attacks, where adversaries use machine learning to automate packet generation, feature obfuscation, and evasion strategies. As Grawe[13] and Khalaf et al.[21] note, such attacks blend malicious and legitimate behaviours, rendering signature-based defences insufficient.

Approaches used	DDoS packet detected	Total packets	Accuracy (%)
Modular-based approach[29]	52,000	100,000	52
Rule-based approach[30]	62,654	100,000	62.6
Neighbour intrusion approaches[31]	62,589	100,000	62.5
Collaborative approach[32]	72,000	100,000	72
DoS défense shield[33]	69,000	100,000	69

Table 1. Evolution of DDoS detection performance in literature.

Existing work	Max accuracy (%)	Detection type	Data set	Storage	Latency	Feature count
Owasu et al.[35]	92.7	Classification	Tor dataset	No	No	6
Reza et al.[36]	97.6	Classification	Custom dataset	No	No	13
Xu et al.[37]	87.8	Classification	Custom dataset	No	No	21
Hamza et al.[38]	97.5	DDoS	UNSW	Yes	Yes	20
Doshi et al.[39]	99.8	DDoS	Custom dataset	No	No	11
Yang et al.[40]	99.8	DDoS	KD99	No	No	9
Our proposed solution	97.13	Classification & DDoS	Custom (SIOTLAB) & UNSW dataset	Yes	Yes	6

Table 2. Benchmarking our lightweight model against existing DDoS detection systems.

AI offers adaptive detection through supervised and unsupervised learning. Khalaf et al.[21] outline various AI-based DDoS defence strategies. Chidananda, Murthy, and Madhu[19] discuss ANN-based prevention frameworks in cloud environments, proposing a theoretical neural system to analyse resources and filter traffic. Khalaf et al.[21] provide a broader survey encompassing statistical and AI-based mitigation methods. Glăvan et al.[20] and Zhang, Zhang and Yu[22] highlight ANN, Bayes classifiers, and SVMs as prominently used detection techniques.

Hybrid and neural approaches for intrusion detection

Alzahrani and Hong[23] introduced a hybrid system combining BigDL-based ANN anomaly detection with a Suricata signature engine. Their model achieved ≈ 98% detection with nearly zero false positives, illustrating the value of hybrid statistical-AI techniques. Said, Overill and Raszik[24] proposed an ANN-based classifier that detected known attacks with 100% precision and unknown attacks with 95%, reinforcing the adaptability of neural models when sufficient feature diversity is available.

Security concerns in smart home architecture

Smart home devices—anything electrically powered and remotely controlled—introduce unique operational risks[9]. Mantas, Lymberopoulos and Komninos[25] identified two primary categories of threats: internal and external. Despite being more than a decade old, their findings remain relevant: heterogeneous devices, persistent connectivity, and inconsistent protection expose smart homes to unauthorized access, privacy breaches, and cross-device threats. Coboi, Tran, and Son[26] further highlight vulnerabilities in applications and propose techniques incorporating ML and ANN-based models for improving smart home security.

Existing DDoS detection in smart homes

Several studies have directly addressed DDoS detection in smart home contexts. Dalal et al.[27] analysed both DDoS and energy-oriented E-DDoS impacts, while Saxena et al.[28] simulated 24-node smart home networks to evaluate defence mechanisms, reporting improved accuracy compared to modular and rule-based approaches. Table 1 summarises performance trends across modular, rule-based, neighbour-based, and collaborative systems[29–33].

Gordon et al.[34] proposed stateless flow-based features for detecting DDoS traffic, achieving improved accuracy across three ML models: k-NN, kernelized SVMs, and Random Forests.

Lightweight vs. deep learning: benchmarking context

While many studies report high accuracies on large, centralized datasets, these results are often achieved under computationally intensive conditions unsuitable for smart homes. By contrast, the present study emphasises lightweight, edge-deployable models trained on real smart-home data, aiming for practical feasibility rather than pure benchmark performance.

Table 2 compares the proposed approach with prior work[35–40], showing that the k-NN-based method achieves 97.13% accuracy on the SIOTLAB dataset, comparable to or better than many deep learning models, while maintaining low feature count and supporting edge deployment.

Gap in literature and motivation for the present study

This work extends prior studies by Wali et al.[41] and Raja et al.[42], using data collected from ongoing smart home deployments. The literature demonstrates that while many models perform well on benchmark datasets, few

are specifically designed for smart home constraints. Only limited studies have explored compact deep learning approaches like ANN in this context, and none directly compare ANN against lightweight models using real smart home traffic.

Therefore, this research contributes a focused comparative evaluation of k-NN and ANN for DDoS detection in smart homes, addressing the lack of studies using real residential datasets and analysing model suitability under limited data, constrained computation, and heterogeneous device environments.

Methodology

This study adopts the CRoss Industry Standard Process for Data Mining (CRISP-DM) framework as the guiding methodology for data analysis and model development. He[43] presents CRISP-DM as an effective structure for managing machine learning workflows, demonstrating its use through a case study on real datasets.

CRISP–DM

CRISP-DM is a widely used data science life-cycle model comprising six phases:

1. Business Understanding.
2. Data Understanding.
3. Data Preparation.
4. Modelling.
5. Evaluation.
6. Deployment.

Chakure[44] provides a detailed outline of the model, illustrating how it supports structured planning and implementation in ML projects (Fig. 2). An additional phase, Monitoring, is sometimes included after deployment to oversee system performance or conduct quality checks[45]. Not all phases must be applied uniformly; they can be adapted based on project requirements. In this work, CRISP-DM serves as a general blueprint, with the modelling and evaluation tasks forming the core of implementation.

There is also one more phase called 'Monitoring' which is after deployment[45]. It can be included based on the requirements to supervise the functioning of the model or like a quality check. It is not necessary or mandatory to include or go through all the phases of a project, it varies or can be adjusted according to the requirement. So, in that case, this ML research project has used the standard model as a basic outline and implemented it according to the requirement. The goals and objectives are set by understanding the market situation. In the following phases, each one has a few required sub-processes for this research and in the implementation section, a few processes like build and assess are being used.

Data analysis process

Data understanding (collect initial data)

The dataset used in this research consists of smart home network traffic collected under two conditions: benign traffic and DDoS attack traffic[46]. Wireshark was used to capture packets, saved both as .pcap files and exported .csv files. The benign traffic is stored in NormalData.csv, while DDoS-induced traffic is stored in DDoSData.csv (Fig. 3).

The smart home environment includes a bulb, plug, and motion sensor (Fig. 4), representing a minimal but realistic IoT setup. To generate attack traffic, the Low Orbit Ion Cannon (LOIC) tool was used to flood the smart home hub's static IP address. Wireshark successfully detected the resulting SYN Flood traffic, which forms the attack dataset (Fig. 5).

Before applying machine learning algorithms, Exploratory Data Analysis (EDA) was performed to inspect statistical properties, identify feature distributions, and assess initial dataset quality. Wali et al.[46] provide a summary of the dataset attributes and their relevance for traffic classification.

Dataset quality During conversion from .pcap to .csv, no missing values were introduced. However, the raw feature values differ significantly from conventional benchmark datasets (e.g., NSL-KDD), requiring formatting and feature transformation during preprocessing.

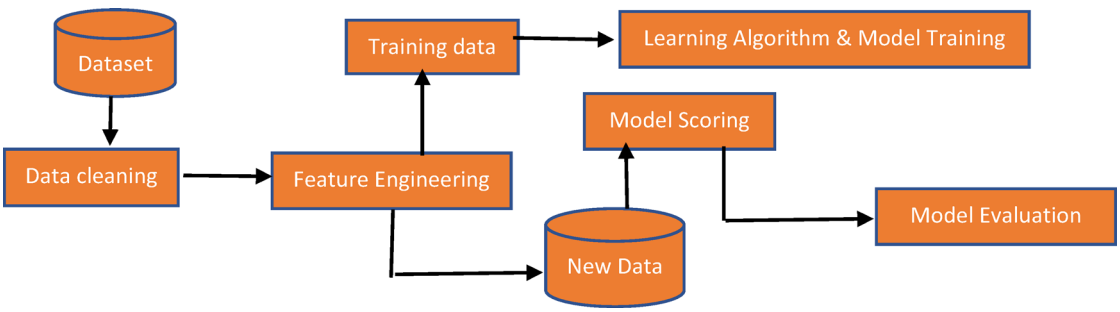


Fig. 2. General ML process.

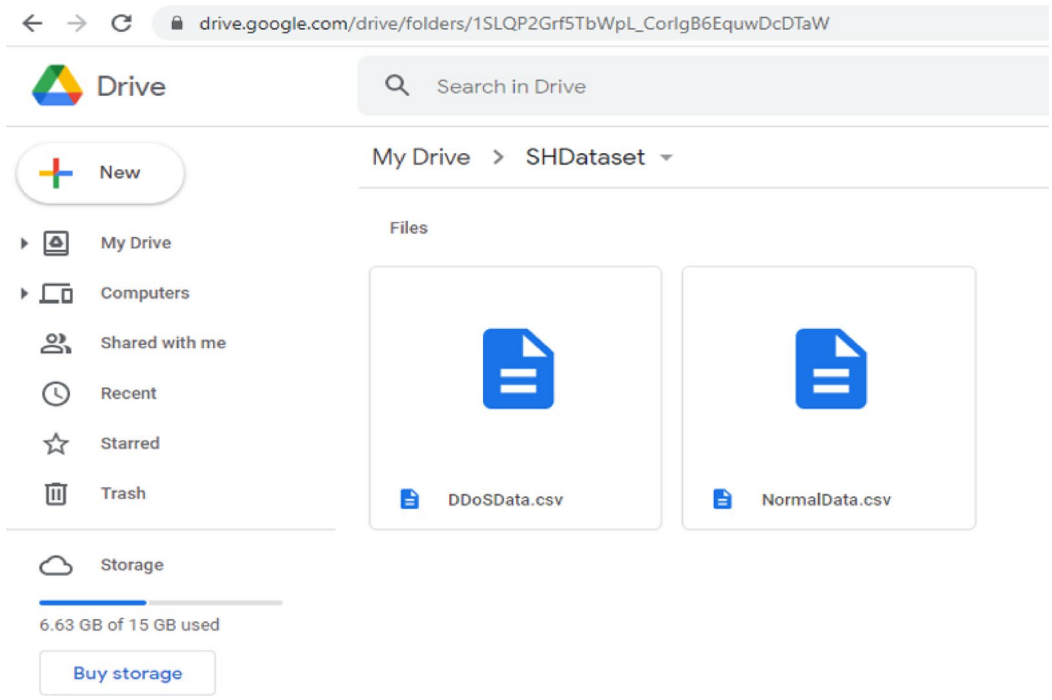


Fig. 3. Smart home datasets in google drive.

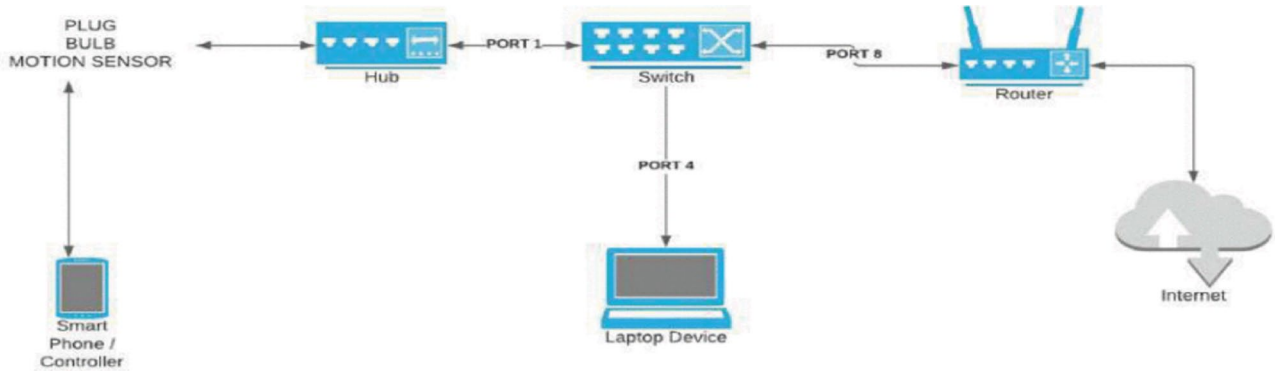


Fig. 4. Smart home architecture[35].

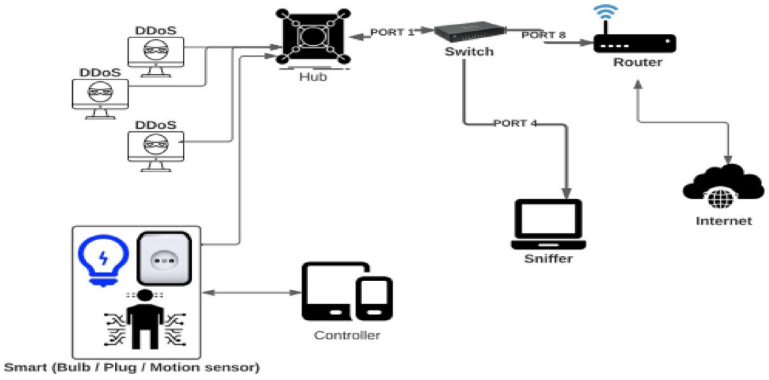


Fig. 5. DDoS attack on smart home architecture[30].

Data preparation
Data preparation, also referred to as data preprocessing, is a critical stage of any ML workflow. Scikit-learn preprocessing tools were used to transform raw packet features into ML-ready numerical representations.

Steps in Data Pre-processing.

1. Import required libraries (Pandas, NumPy, Matplotlib, Scikit-learn).
2. Import datasets.
3. Remove irrelevant attributes (e.g., Info column).
4. Label each dataset (Normal or DDoS).
5. Encode categorical attributes.
6. Apply feature scaling.
7. Split data into training and testing sets.

Main selected features

- Available features: No., Time, Source, Destination, Protocol, Length, Info.
- Required features: No., Time, Source, Destination, Protocol, Length.
- Target variable: Traffic Class (Normal or DDoS).

Label encoding
Categorical features (Time, Source, Destination, Protocol) were converted into numerical form using the Label Encoder. Although One-Hot Encoding is commonly used for protocol fields, label encoding was preferred because one-hot vectors can sometimes degrade performance in certain models and high-cardinality datasets[47].

For more details and when initiating 'Select Data', the following options are selected:

- Two datasets are selected: Benign Data and Attack data in csv format.
- Available Features: 'No.', 'Time', 'Source', 'Destination', 'Protocol', 'Length', 'Info'.
- Required features: 'No.', 'Time', 'Source', 'Destination', 'Protocol', 'Length'.
- Target: Traffic Class – Label [Normal or DDoS].
- Clean data: The Info feature of the network traffic dataset is not required for this project at this point, so this column must be deleted. There are no missing values or null values before processing that could affect the.
- ML algorithm, so it is left as it is.
- Construct data: An extra feature is added to both datasets called "label". The label defines that the network traffic row belongs to a normal or attack dataset. The labelling helps in training the ML algorithm.
- Format data: This method is very crucial in data pre-processing. In this process, we convert the raw data into ML-suitable data.
- 'Time', 'Source', 'Destination', 'Protocol'.
- The above features need to be transformed into a suitable format by using the Label Encoder method and normalization.

Normalization
After encoding, all features were normalized using Min–Max Scaling, which rescales values to the range 0–1 without distorting relative differences. Aniruddha[48] describes normalization as an effective scaling strategy for heterogeneous numeric ranges, and it is widely implemented using the Min–Max method[49]:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

When splitting the data, the whole data is split into train and test sets. The data in the training set is used to train the ML algorithm model, and to test the proper accuracy of the model, the test set can be used to predict. Usually, 70% of the data is allocated to the training set and 30% is allocated test and validation set. But in this work, the train set is 80% and the test set is 20%, as we are dealing with DDoS attack data the model needs to be trained well to implement the functionality.

Modelling phase

Model selection
Based on a review of existing literature, two models were selected:

- k-Nearest Neighbour (k-NN) for lightweight ML.
- Artificial Neural Network (ANN) for compact deep learning[50].

k-NN was selected due to its simplicity and suitability for binary classification tasks (Normal vs. DDoS). ANN was chosen to explore whether a compact neural architecture provides improved generalization on IoT traffic.

The main configuration parameters and their corresponding rationale for both models are summarized in Table 3 to ensure transparency and reproducibility of the experimental setup. As mentioned earlier in this work,

Model	Key parameters	Rationale
k-NN	k = 5, Euclidean distance	Balances bias–variance trade-off; widely used for network traffic classification.
ANN	3 layers (6-12-1), ReLU activation, Adam optimizer, 10–50 epochs	Compact architecture suited for limited data; ReLU ensures non-linear mapping.

Table 3. Model hyperparameters and rationale.

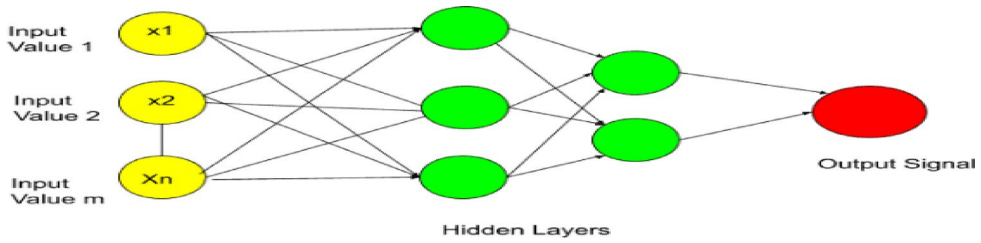


Fig. 6. Simple ANN.

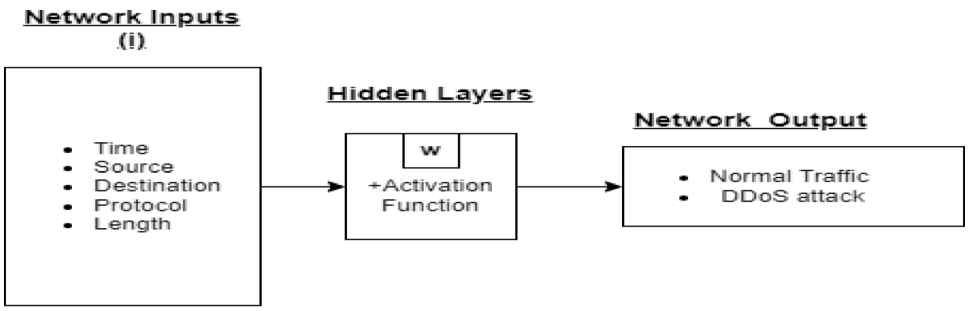


Fig. 7. ANN for this work.

k-NN is chosen to learn how it classifies the data, as the k-NN is based on classification and the data just has 2 labels to classify, and it can categorize using nearest features.

k-NN model

k-NN is a non-parametric classification algorithm that assigns labels based on proximity to neighbouring data points. The choice of K and the distance metric directly influence classification effectiveness[51]. In this project, k-NN operates in two stages:

1. Learning step: storing labelled feature vectors.
2. Classification step: computing Euclidean distances to identify nearest neighbours.

Given the binary nature of the problem, k-NN is effective in identifying DDoS traffic based on even subtle deviations in feature behaviour.

ANN model

Artificial Neural Networks (ANNs) are inspired by biological neural systems and consist of interconnected nodes organised into layers[52]. ANN architectures typically fall into two categories: Feedforward Neural Networks and Feedback Neural Networks. This work uses a Feedforward ANN, where data moves in a single direction through the network[53].

ANNs are widely used for pattern recognition, anomaly detection, and traffic classification, which are key requirements for DDoS detection[52–54]. The network contains three primary components:

- Input layer.
- Hidden layer(s).
- Output layer.

Activation functions (e.g., ReLU, sigmoid) determine the node outputs, while weights represent learned relationships. Figure 6 shows a basic ANN architecture, and Fig. 7 illustrates the ANN used in this study.

In this research, the ANN-based model is illustrated in Fig. 7 with its main inputs. Although the numeric column is processed by the model, the figure only depicts the core inputs.

Model configuration and reproducibility
To ensure reproducibility, key hyperparameters and configuration choices are summarised below.
k-NN Configuration.

- k = 5.
- Distance metric: Euclidean.
- Rationale: balances bias–variance, low computational cost, suitable for edge devices.

ANN Configuration.

- Architecture: 3 layers (6–12–1).
- Activation: ReLU (hidden), Sigmoid (output).
- Optimizer: Adam.
- Loss function: Binary cross-entropy.
- Epochs: 10–50.
- Rationale: compact architecture prevents overfitting on limited IoT datasets.

Training Environment.

- Train/test split: 80/20.
- Environment: Google Colab (Python 3.10).
- Libraries: Scikit-learn, TensorFlow/Keras.

Table 3 summarises the hyperparameters and their rationale.

Evaluation phase

The models are critically evaluated. The performance of both models is evaluated using a few standard metrics as follows:

- Confusion Matrix: The confusion matrix identifies the false positives (FP) and False Negatives (FN) along with True Positives (TP) and True Negatives (TN) which play an important role in defining the efficiency of a model.
- Precision: It is defined as the ratio of actual positives from total predicted positive values.

$$Precision = \frac{TP}{TP + FP} = \frac{Predictions\ Actually\ Positive}{Total\ Predicted\ Postive} \tag{2}$$

- Recall: It defines the actual correct positive predictions.

$$Recall = \frac{TP}{TP + FN} = \frac{Prediction\ Actually\ Postive}{Total\ Actual\ Postive} \tag{3}$$

- F1 -Score: It is an average of precision and recall.

$$F1 - Score = 2 * \frac{(Recall\ *\ Precision)}{(Recall + Precision)} \tag{4}$$

- Accuracy: It is the overall true predictions both positives and negatives.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{Correct\ Predictions}{Total\ Predictions} \tag{5}$$

- FPR and FNR: This is the false positive rate (FPR - the type I error) and the false negative rate (FNR- the type II error) is calculated as follows:

$$\begin{aligned} \text{FPR} &= \text{False positive / Total Negative} \\ \text{FNR} &= \text{False Negative / Total Positive.} \end{aligned}$$

- ROC Curve: It means Receiver operating Characteristic curve. It is an important graph that is used to visualize the performance of the classification model at all thresholds. This graph consists of two factors: True Positive Rate (TPR) and False Positive Rate (FPR). It is said that the model efficiency is defined by the curve area.

Review Process: Each process and step are reviewed and experimented with different ways to learn the best effective way and to avoid errors.

Deployment phase

This phase is executed including all the steps whereby the model is developed and described during the implementation phase. The implementations are documented and reported, and it's monitored to avoid any major issues.

Tools and libraries used

The device used for this research project is a DELL laptop which is windows 10 operating system with 64-bit OS, x64- based processor, 8GB RAM and Intel(R) Core (TM) i5- 7200U CPU @ 2.5 GHz 2.70 GHz processor.

The programming language used for this project is Python as it is easy to learn and widely used language for ML research and projects. Google Collaboratory (Google Colab) is used as a development tool/platform. It is very easy to import vast and varies libraries compared to other platforms. Wireshark is used to monitor and analyse the traffic and to extract the csv file. Libraries used are as follows:

- NumPy – to perform mathematical calculations.
- Scikit-learn – to perform all the ML processes and analysis.
- Pandas – to handle dataset.
- Matplotlib – to plot and visualize data.

Given the class imbalance in the dataset (18.2% normal vs. 81.8% attack traffic), relying solely on accuracy is insufficient, as a model that always predicts 'attack' would achieve a deceptively high accuracy of 81.8%. Therefore, we employ a suite of metrics (Precision, Recall, F1-score, and confusion matrices) to provide a nuanced evaluation of model performance on both the minority and majority classes.

Implementation and results

AI models are implemented on the collected smart home dataset to see the performance of each model in identifying the DDoS attack.

Smart home data set

The smart home dataset is the network traffic of the home network (Fig. 8). The traffic is first monitored in Wireshark. The data is collected under two scenarios: (1) The general traffic flow of the network; and (2) The attack traffic that is by creating a DDoS attack on the network hub using a tool called LOIC.

Normal dataset

The normal traffic as captured in the Wireshark is collected and converted to .csv file (Fig. 9). This .csv file consists of normal dataset which is benign data.

The normal dataset includes features as shown in Table 4.

The normal dataset contains 22,296 data entries in 7 columns which are features. The normal dataset contains traffic flow for approximately 10 h and 45 min. The basic information of the dataset is shown below:

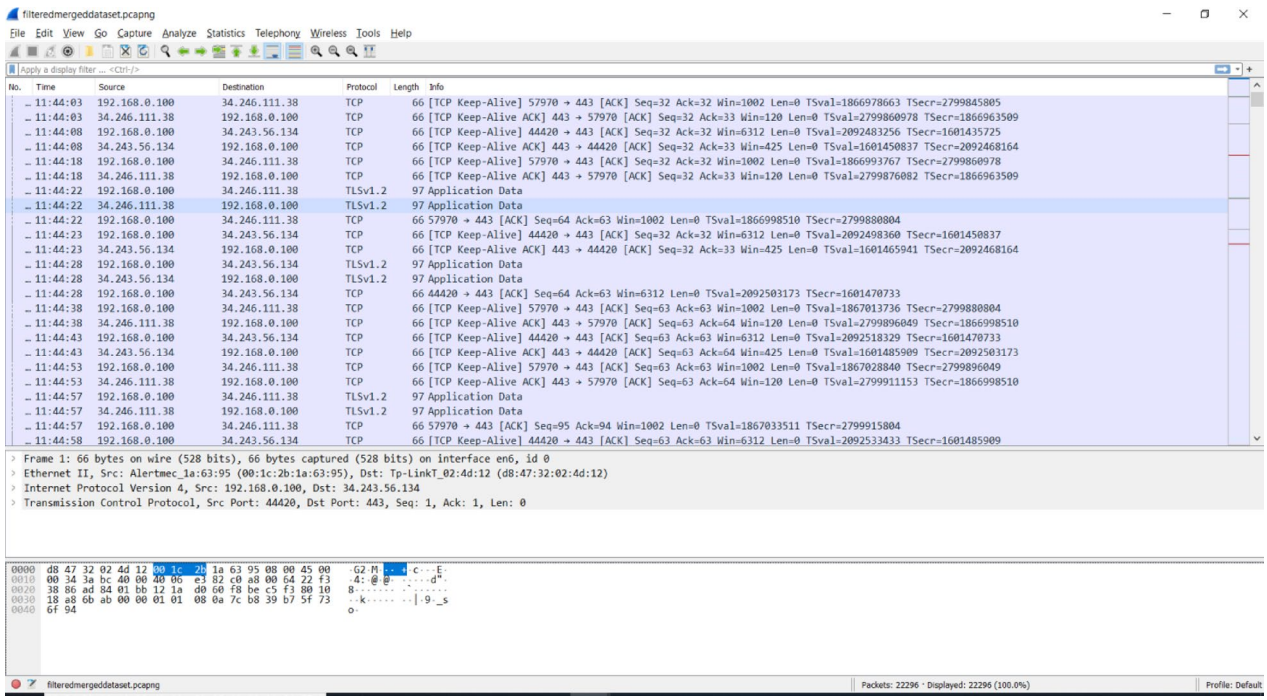


Fig. 8. Home network traffic in Wireshark.

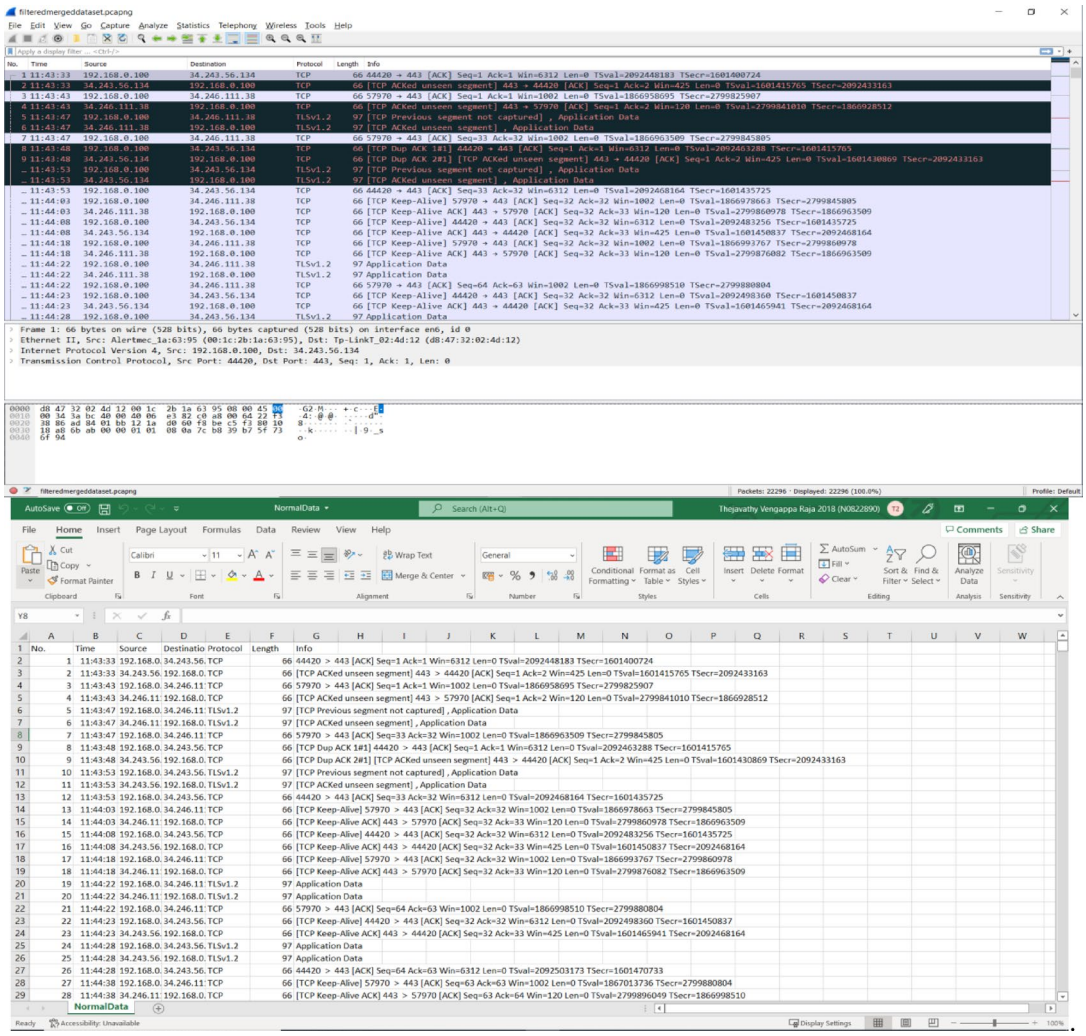


Fig. 9. Home network traffic in Wireshark where CSV file consists of a normal dataset.

Parameter	Description
Total data duration	Approximately 10 h 45 min of normal smart home activity and 42 min of DDoS attack simulation
Smart home devices used	3 IoT devices: bulb, plug, and motion sensor
Network configuration	Wi-Fi-based LAN connected to the internet through a residential gateway
Total traffic records	Approximately 122,000 network packets (flows) captured
Attack types simulated	UDP Flood and ICMP Flood DDoS attacks launched on the smart home network
Extracted features	No., Time, Source, Destination, Protocol, Length, Info (7 features total)
Label distribution	Normal traffic: 18.2% Attack traffic: 81.8%
Data collection environment	Single smart home setup representing a realistic residential network environment

Table 4. Smart home dataset summary.

Range Index: 22,296 entries, 0 to 22,295 Data columns (total 7 columns):			
#	Column	Non-Null Count	Dtype
0	No.	22,296 non-null	int64
1	Time	22,296 non-null	object
2	Source	22,296 non-null	object
3	Destination	22,296 non-null	object
4	Protocol	22,296 non-null	object
5	Length	22,296 non-null	int64
6	Info	22,296 non-null	object
dtypes: int64(2), object(5) memory usage: 1.2 + MB			

The data columns describe the following information:

- No.: The traffic sequence number.
- Time: Time stamps for each traffic flow or network conversation.
- Source: Source Id of each current traffic flow.
- Destination: Destination Id of each current traffic flow.
- Protocol: Describes which protocol is being used for communication.
- Length: Length of the data packet.
- Info: This describes the state of the network conversation in detail.

DDoS dataset

The DDoS dataset was similarly collected using Wireshark during a simulated attack on the smart home network, as previously described. Despite the ongoing attack, no noticeable service disruption or delay occurred for the smart home resources. Wireshark successfully captured the attack traffic alongside normal flows, flagging the suspected packets (Fig. 10). For the purpose of this study, we focused exclusively on capturing traffic during the active attack window to obtain DDoS patterns for model training. This collected data was subsequently converted into a .csv file for analysis.

Like the normal dataset, this dataset also has the same feature columns. The Attack data set contains 1,00,014 entries in 7 columned features. This data is captured for approximately only 42 min. The basic information of the dataset is as below:

RangeIndex: 100,014 entries, 0 to 100,013 Data columns (total 7 columns):			
#	Column	Non-Null Count	Dtype
0	No.	22,296 non-null	int64
1	Time	22,296 non-null	object
2	Source	22,296 non-null	object
3	Destination	22,296 non-null	object
4	Protocol	22,296 non-null	object
5	Length	22,296 non-null	int64
6	Info	22,296 non-null	object
dtypes: int64(2), object(5) memory usage: 5.3 + MB			

EDA analysis

While Wireshark provides granular, packet-level inspection, EDA was employed to understand broader dataset characteristics, feature distributions, and class relationships. This high-level perspective is crucial for informing the subsequent modelling phase. For this study, EDA was implemented using Python libraries in Google Colab due to their efficiency and strong visualization capabilities.

The EDA process provided insights into the fundamental properties of the smart home traffic. For instance, it enabled a comparative analysis of protocol distributions between normal and attack scenarios, revealing distinct behavioural patterns. The analysis conducted in this work, which examined the entire dataset prior to pre-processing, is illustrated in Figs. 11 and 12, and 13.

Protocol Count in DDoS dataset.

TCP	92,361
ICMP	5945
TLSv1.2	1660

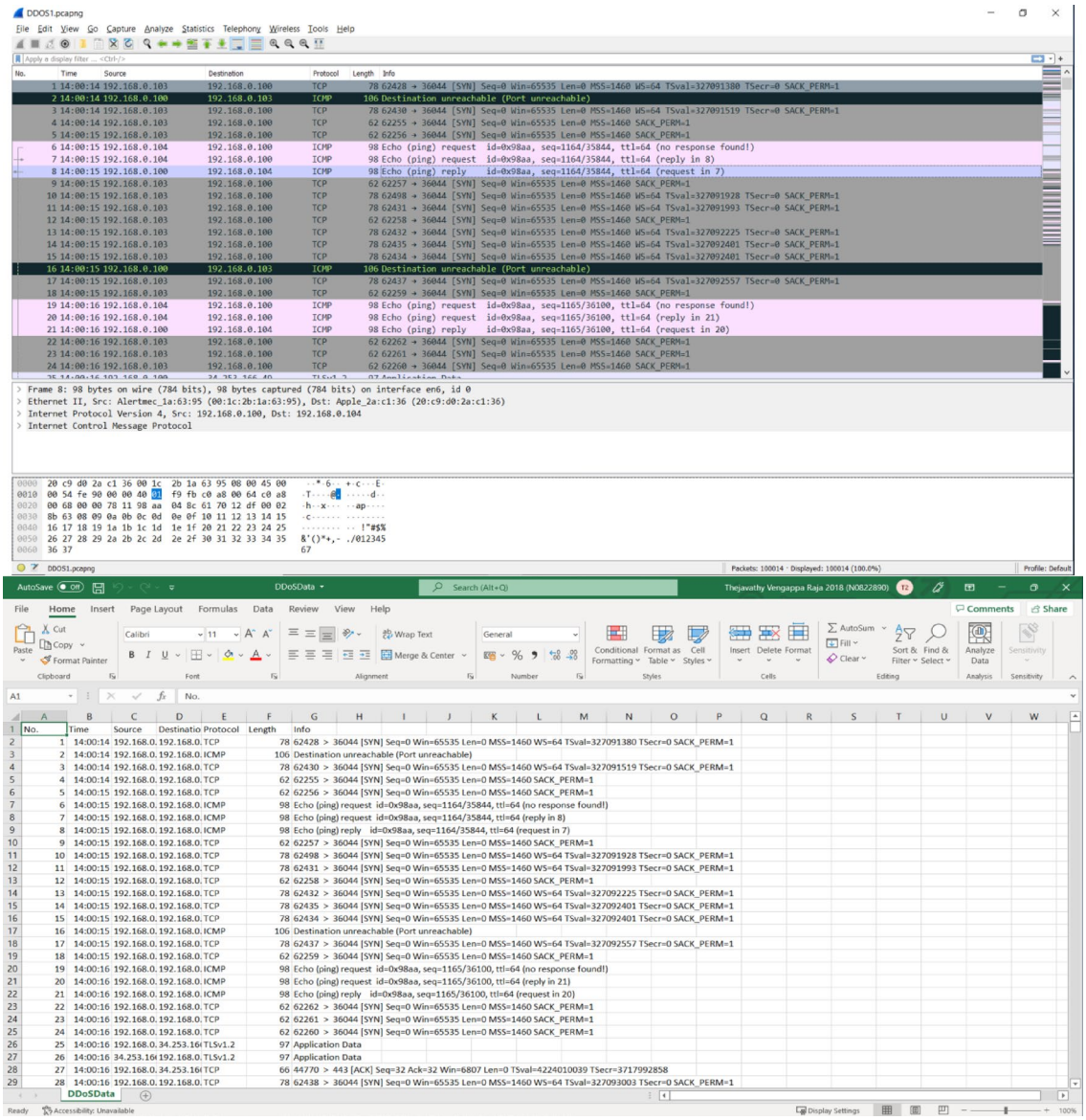


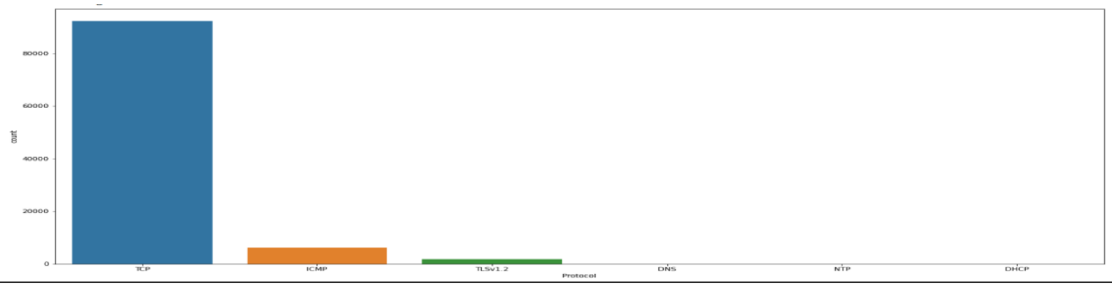
Fig. 10. DDoS attack traffic in Wireshark (top), DDoS attack traffic data in CSF file (bottom).

DNS	42
NTP	4
DCHP	2

Protocol count in normal dataset.

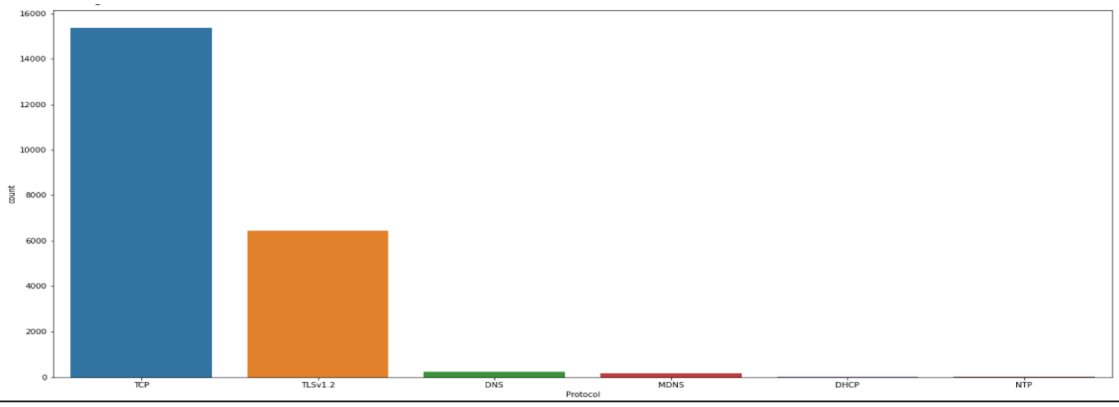
TCP	15,378
TLSv1.2	6449
DNS	238
MDNS	171
NTP	38
DCHP	22

Source and destination are like a handshake communication in a network. When looking at the above diagram clearly shows that there was an incomplete handshake showing that the IP address might be spoofing. It's just a visual graph which makes it easy for understanding.



TCP	92361
ICMP	5945
TLSv1.2	1660
DNS	42
NTP	4
DHCP	2

Fig. 11. Protocol count in DDoS dataset.



TCP	15378
TLSv1.2	6449
DNS	238
MDNS	171
NTP	38
DHCP	22

Fig. 12. Protocol count in normal dataset.

Data pre-processing

The collected raw data, as displayed in the previous figures, requires pre-processing to be suitable for machine learning models. The initial step involved loading the dataset into the Google Colab environment. The data files, stored in a Google Drive folder, were mounted and loaded into Pandas DataFrames. Prior to pre-processing, the datasets were labeled for the binary classification task: normal traffic was assigned a label of 0, and DDoS attack traffic was assigned a label of 1, as illustrated below:

```
#label dataset with respective type  : Normal data or DoS data
# dos  = 1 , normal = 0
dos_df['label'] = 1
nor_df['label'] = 0
```

The DDoS and normal datasets include an additional column called "label," which contains their respective labels (Figs. 14 and 15). The entries in other columns remain in their original, raw format and must be converted into specific ranged numeric values for model compatibility.

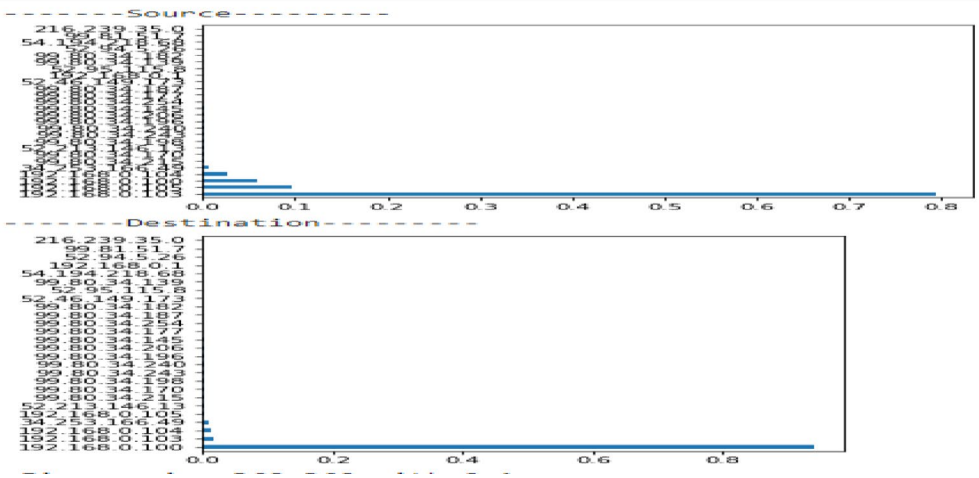


Fig. 13. Source and destination plotting for DDoS data.

	No.	Time	Source	Destination	Protocol	Length	Info	label
0	1	14:00:14	192.168.0.103	192.168.0.100	TCP	78	62428 > 36044 [SYN] Seq=0 Win=65535 Len=0 MS...	1
1	2	14:00:14	192.168.0.100	192.168.0.103	ICMP	106	Destination unreachable (Port unreachable)	1
2	3	14:00:14	192.168.0.103	192.168.0.100	TCP	78	62430 > 36044 [SYN] Seq=0 Win=65535 Len=0 MS...	1
3	4	14:00:14	192.168.0.103	192.168.0.100	TCP	62	62255 > 36044 [SYN] Seq=0 Win=65535 Len=0 MS...	1
4	5	14:00:15	192.168.0.103	192.168.0.100	TCP	62	62256 > 36044 [SYN] Seq=0 Win=65535 Len=0 MS...	1

Fig. 14. DDos attack dataset.

	No.	Time	Source	Destination	Protocol	Length	Info	label
0	1	11:43:33	192.168.0.100	34.243.56.134	TCP	66	44420 > 443 [ACK] Seq=1 Ack=1 Win=6312 Len=0...	0
1	2	11:43:33	34.243.56.134	192.168.0.100	TCP	66	[TCP ACKed unseen segment] 443 > 44420 [ACK]...	0
2	3	11:43:43	192.168.0.100	34.246.111.38	TCP	66	57970 > 443 [ACK] Seq=1 Ack=1 Win=1002 Len=0...	0
3	4	11:43:43	34.246.111.38	192.168.0.100	TCP	66	[TCP ACKed unseen segment] 443 > 57970 [ACK]...	0
4	5	11:43:47	192.168.0.100	34.246.111.38	TLSv1.2	97	[TCP Previous segment not captured] , Applicat...	0

Fig. 15. Normal dataset with label column.

Raw smart home data conversion

Label Encoder: All the columns need to be encoded to standard values without losing data except the number column. The time is converted to a Unix timestamp (Fig. 16).

As the source and destinations are in IP address format, they are converted in numerical characteristics. Each IP address is assigned to a number (Fig. 17).

The final one that needs to be label encoded is protocol. Each protocol is assigned to a numerical label (Fig. 18).

The labels assigned to each protocol used in our ML model are as follows:

- When assigning labels to the protocol, the following output is given. To Identify the protocol assignment. The counter function i used to calculate the count of total protocols before and after labelling. Therefore, DHCP, DNS, ICMP. NTP, and TCP are assigned labels 0, 1, 2, 3, and 4 respectively.
- Protocols are labelled with numbers while processing data uses label encoder. The scikit-learn has a pre-processing library in which the Label encoder function is available.
- le = LabelEncoder(); Encoded_values = le.fit_transform(column1 values).
- Importing the label encoder function and passing the required column values to the label encoder object along with fit_transform function converts all the values and assigns them to the specified variable.
- Filtering Features: The info section of the dataset is not required at this stage. So, we are clearing the unnecessary data at this point by dropping the column.

```
#for Time Series Object conversion
def convert_to_timestamp(x):
    """Convert date objects to integers"""
    return time.mktime(x.to_pydatetime().timetuple())


# Convert to date objects
dos_df['Time'] = pd.to_datetime(dos_df['Time'])
nor_df['Time'] = pd.to_datetime(nor_df['Time'])

# print(dos_df['Time'])

# Now df has date objects like you would, we convert to UNIX timestamps
dos_df['Time'] = dos_df['Time'].apply(convert_to_timestamp)
nor_df['Time'] = nor_df['Time'].apply(convert_to_timestamp)

print(nor_df['Time'].head())

# # # dos_df.dtypes
```

```
0    1.649677e+09
1    1.649677e+09
2    1.649677e+09
3    1.649677e+09
4    1.649677e+09
Name: Time, dtype: float64
```

Fig. 16. Time conversion to Unix timestamp.

```
[ ]  # Label encode the Source and detination ip address
     from sklearn.preprocessing import LabelEncoder
     le = LabelEncoder()
     #Source Column
     dos_df.Source = le.fit_transform(dos_df.Source.values)
     nor_df.Source = le.fit_transform(nor_df.Source.values)

     #Destination Column
     dos_df.Destination = le.fit_transform(dos_df.Destination.values)
     nor_df.Destination = le.fit_transform(nor_df.Destination.values)
     print(dos_df)
```

	No.	Time	Source	Destination	Protocol	Length	\
0	1	14:00:14	2	1	TCP	78	
1	2	14:00:14	1	2	ICMP	106	
2	3	14:00:14	2	1	TCP	78	
3	4	14:00:14	2	1	TCP	62	
4	5	14:00:15	2	1	TCP	62	
...	...	...	...	...	...	...	
100009	100010	14:42:01	1	6	TCP	66	
100010	100011	14:42:01	6	1	TCP	66	
100011	100012	14:42:08	1	7	TLSv1.2	97	
100012	100013	14:42:08	7	1	TLSv1.2	97	
100013	100014	14:42:08	1	7	TCP	66	

Fig. 17. Source and destination label encoder.

```
#Protocol Column Encode
dos_df.Protocol = le.fit_transform(dos_df.Protocol.values)

nor_df.Protocol = le.fit_transform(nor_df.Protocol.values)

print(nor_df.Protocol)
Counter(nor_df.Protocol)

# """ Labels: 'DHCP': 0,'DNS': 1, 'ICMP': 2, 'NTP': 3, 'TCP': 4, 'TLSv1.2': 5 """
```

```
0        4
1        4
2        4
3        4
4        5
        ..
22291    5
22292    5
22293    4
22294    4
22295    4
Name: Protocol, Length: 22296, dtype: int64
Counter({0: 22, 1: 238, 2: 171, 3: 38, 4: 15378, 5: 6449})
```

Fig. 18. Protocol label encoder.

```
dos_df.head()
```

	No.	Time	Source	Destination	Protocol	Length	label
0	1	1.649686e+09	2	1	4	78	1
1	2	1.649686e+09	1	2	2	106	1
2	3	1.649686e+09	2	1	4	78	1
3	4	1.649686e+09	2	1	4	62	1
4	5	1.649686e+09	2	1	4	62	1

```
nor_df.head()
```

	No.	Time	Source	Destination	Protocol	Length	label
0	1	1.650541e+09	2	6	4	66	0
1	2	1.650541e+09	5	2	4	66	0
2	3	1.650541e+09	2	7	4	66	0
3	4	1.650541e+09	6	2	4	66	0
4	5	1.650541e+09	2	7	5	97	0

Fig. 19. Data pre-processing: part-1.

```
"""Method 1"""
#Normalize the original data

from sklearn import preprocessing

df = preprocessing.normalize(df)

print(df)
```

```
[[6.05858795e-10 1.00000000e+00 1.21171759e-09 ... 2.42343518e-09
  4.72569860e-08 6.05858795e-10]
 [1.21171759e-09 1.00000000e+00 6.05858795e-10 ... 1.21171759e-09
  6.42210323e-08 6.05858795e-10]
 [1.81757638e-09 1.00000000e+00 1.21171759e-09 ... 2.42343518e-09
  4.72569860e-08 6.05858795e-10]
 ...
 [1.35067681e-05 1.00000000e+00 1.21169535e-09 ... 2.42339070e-09
  3.99859466e-08 0.00000000e+00]
 [1.35073739e-05 1.00000000e+00 1.21169535e-09 ... 2.42339070e-09
  3.99859466e-08 0.00000000e+00]
 [1.35079798e-05 1.00000000e+00 3.02923838e-09 ... 2.42339070e-09
  3.99859466e-08 0.00000000e+00]]
```

Fig. 20. Data normalization.

```
[ ]  #Drop Info column
     dos_df = dos_df.drop(['Info'], axis=1)
     nor_df = nor_df.drop(['Info'], axis=1)
     display(list(nor_df.columns.values))
```

```
['No.', 'Time', 'Source', 'Destination', 'Protocol', 'Length', 'label']
```

Only the required features are carried forward to the next process. After data cleansing and conversion, the dataset appears as shown in Fig. 19 for both attack and normal data.

Feature Scaling: Data is converted, in which the length and most importantly time values are in a huge range different from the rest. To bring data uniformity, the models require data to be in similar ranges. The whole data is converted to a specific range without affecting the values. We are using Normalization at this step to do scaling which brings the data range to 0 and 1 by using min/max. The scikit-learn preprocessing library provides the normalize function, which scales the data into the specified range (Fig. 20).

Normalized_df = preprocessing.normalize(df).

Before doing the normalization the two datasets are integrated, both the normal dataset and the attack dataset to train the model with their labels.

```
# combining two datasets - both normal and DDoS
df = pd.concat([dos_df, nor_df])
df
```

Splitting Dataset: After data pre-processing, the main process is to split the dataset into train and test set so that the model can train and learn from the allocated training dataset and the test set is used to predict, test or sometime to do cross validation testing to evaluate the efficiency of the model as shown below:

```
[41] from sklearn.model_selection import train_test_split

     #Split dataset into train and test
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

The implementation flow, including the core step-by-step process of this project, is presented in a simplified flow diagram (Fig. 21).

Implementation of AI models

After splitting the dataset to train and test the data sets. The ML model needs to be trained with the training data. As we chose k-NN and ANN as our AI models to work on in this research. The below sections show each model implementation.

k-NN

In this model the k-NN is executed on the integrated smart home dataset containing benign and attack data, it learns and trains the model with the available data. The n_neighbors (K) is chosen to be 5 and the metric value is chosen to be 'Euclidean' as it is the most popular one and works well with this dataset.

The k-NN model code snippet:

```
[ ]  """ K Nearest Neighbor """

     from sklearn.neighbors import KNeighborsClassifier

     knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
     knn.fit(X_train, y_train)
```

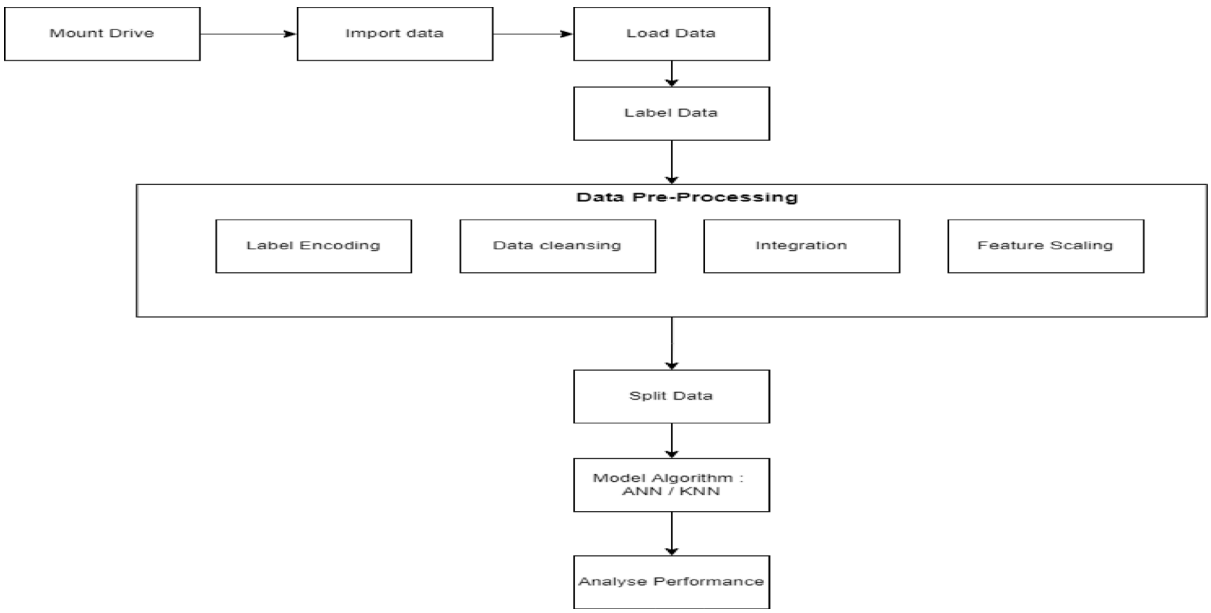


Fig. 21. Project flow.

It looks like a simple line of coding but plays a major role in classification. This model is used to find the efficiency as it is one of the popular 'ML classification' models.

ANN

This model is chosen because it is well known for recognising patterns and for classification in 'deep learning'. The smart home dataset is trained through this model to learn the patterns and characteristics along each layer as shown below:

```
""" ANN """
import keras
from keras.models import Sequential
from keras.layers import Dense

classifier = Sequential()
classifier.add(Dense(6, kernel_initializer='uniform', activation = 'relu',input_dim = 6))
classifier.add(Dense(6,activation = 'relu'))
classifier.add(Dense(1,activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
classifier.fit(X_train, y_train, batch_size = 10, epochs = 10)
```

The ANN was structured as a feedforward network with three layers. The input and first hidden layer each contained six neurons, matching the six input features: No., Time, Source, Destination, Protocol, and Length[50,51]. Weights were initialized uniformly. The ReLU activation function was used in the hidden layers for efficient training, while the output layer used a sigmoid function for binary classification[52,53]. The model was compiled with the Adam optimizer and a binary cross-entropy loss function. An initial training run of 10 epochs was conducted to establish a baseline performance while minimizing the risk of overfitting on the limited dataset (Figs. 22 and 23, and 24).

Results and discussions

Implementation findings

k-NN

The confusion matrix value is as follows:

[[4005 453]

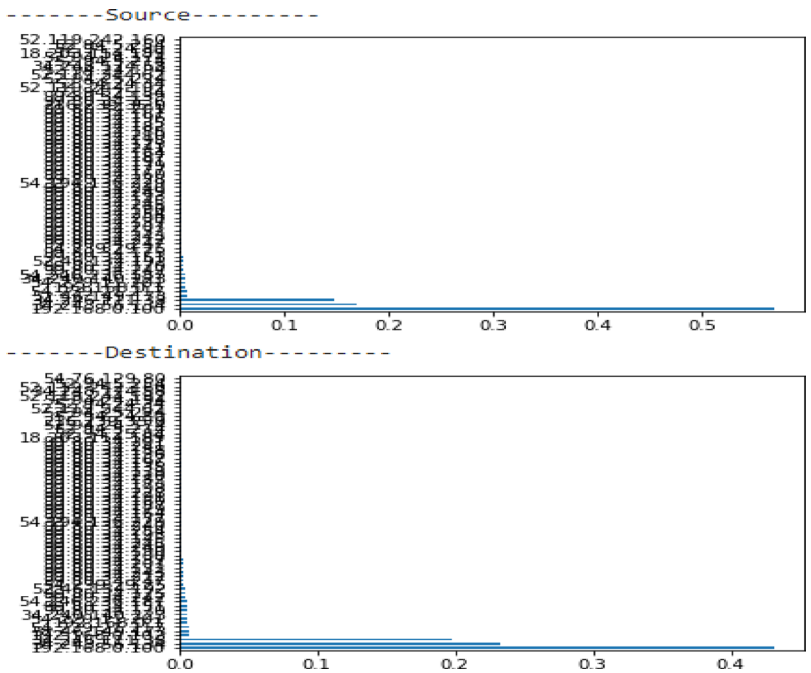


Fig. 22. Source and destination plotting for normal data.

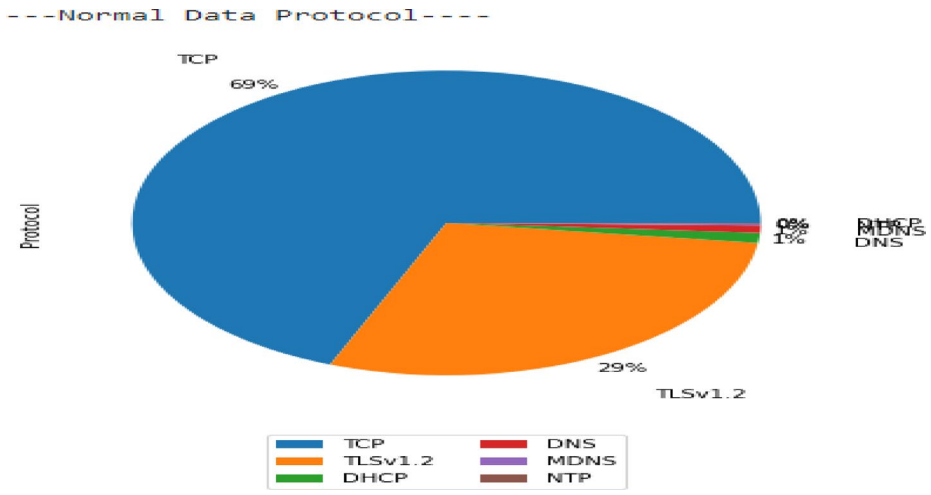


Fig. 23. Protocols in normal traffic flow.

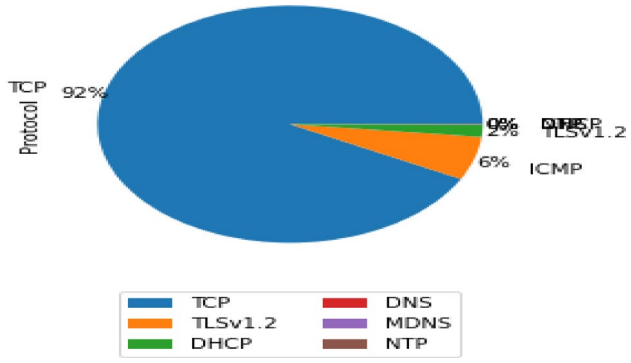
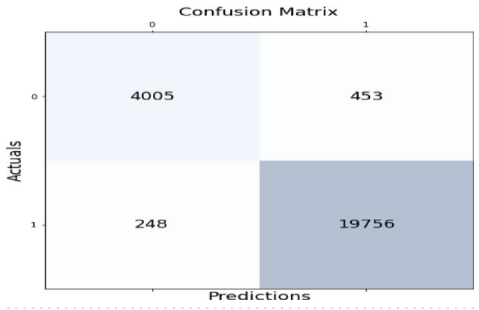


Fig. 24. Protocols in DDoS traffic flow.

[248 19756]]

Confusion matrix and Classification report for the k-NN model are shown below:



Classification report:

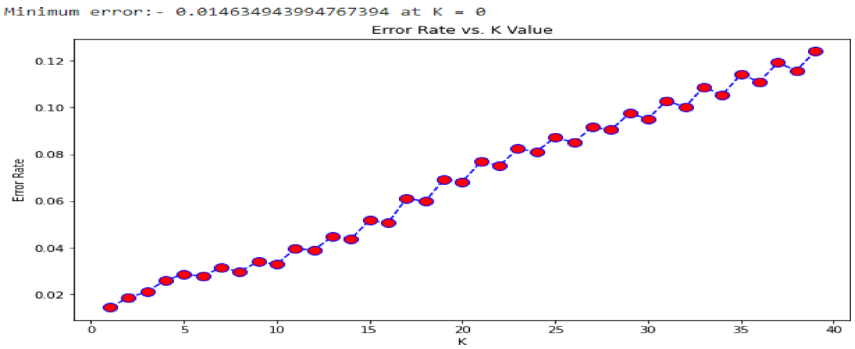


Fig. 25. Error rate for the k-NN model.

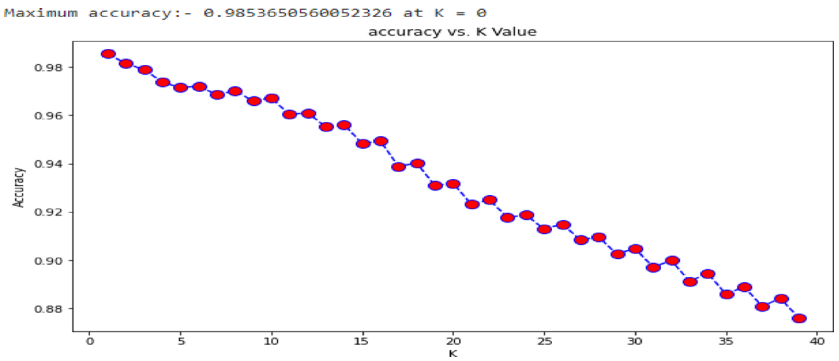


Fig. 26. Accuracy rate for the k-NN model.

	precision	recall	f1-score	support
0	0.94	0.90	0.92	4458
1	0.98	0.99	0.98	20004
accuracy			0.97	24462
macro avg	0.96	0.94	0.95	24462
weighted avg	0.97	0.97	0.97	24462

The error rate, accuracy rate, and ROC curve for the k-NN model are depicted in Figs. 25 and 26, and 27.

Figure 26 illustrates the k-NN model's error rate versus the K value, with the training error remaining below 12%. Figure 27 shows the corresponding training accuracy, which achieved 98%; this high value is indicative of performance on the training subset. The model's discriminatory performance is further visualized by the ROC curve in Fig. 28.

ANN

The ANN was tuned and tested to improve the efficiency. Each trial is shown below:

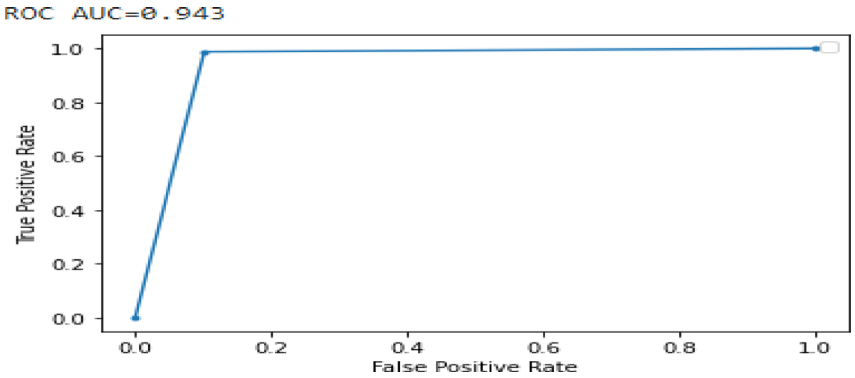


Fig. 27. ROC curve for the k-NN model.

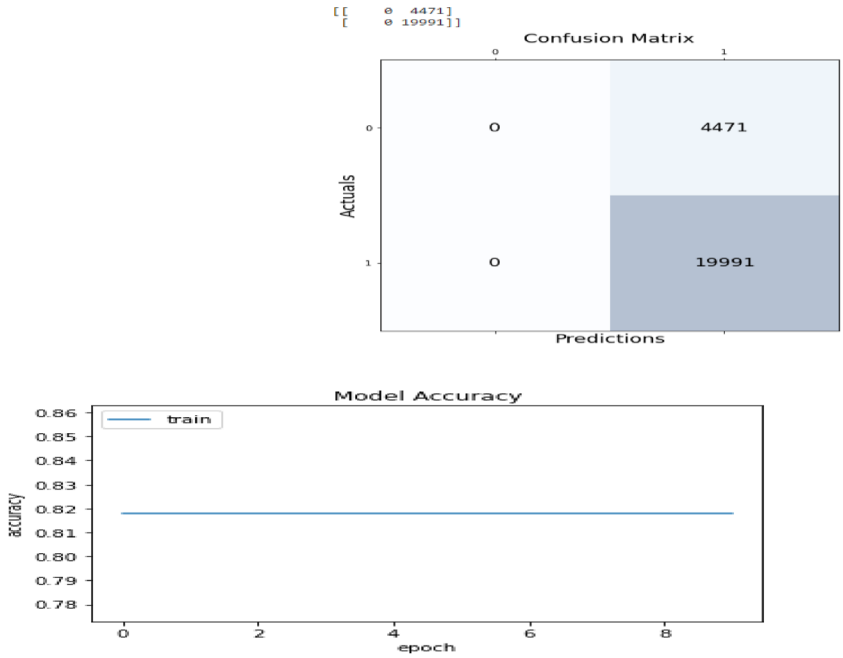


Fig. 28. ANN model accuracy.

```
""" ANN """
import keras
from keras.models import Sequential
from keras.layers import Dense

classifier = Sequential()
classifier.add(Dense(6, kernel_initializer='uniform', activation = 'relu',input_dim = 6))
classifier.add(Dense(6,activation = 'relu'))
classifier.add(Dense(1,activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
classifier.fit(X_train, y_train, batch_size = 10, epochs = 10)
```

```
Epoch 1/10
9785/9785 [==============================] - 15s 1ms/step - loss: 0.4774 - accuracy: 0.8175
Epoch 2/10
9785/9785 [==============================] - 14s 1ms/step - loss: 0.4752 - accuracy: 0.8176
Epoch 3/10
9785/9785 [==============================] - 14s 1ms/step - loss: 0.4752 - accuracy: 0.8176
Epoch 4/10
9785/9785 [==============================] - 14s 1ms/step - loss: 0.4751 - accuracy: 0.8176
Epoch 5/10
9785/9785 [==============================] - 14s 1ms/step - loss: 0.4751 - accuracy: 0.8176
Epoch 6/10
9785/9785 [==============================] - 14s 1ms/step - loss: 0.4750 - accuracy: 0.8176
Epoch 7/10
9785/9785 [==============================] - 14s 1ms/step - loss: 0.4749 - accuracy: 0.8176
Epoch 8/10
9785/9785 [==============================] - 14s 1ms/step - loss: 0.4748 - accuracy: 0.8176
Epoch 9/10
9785/9785 [==============================] - 14s 1ms/step - loss: 0.4748 - accuracy: 0.8176
Epoch 10/10
9785/9785 [==============================] - 14s 1ms/step - loss: 0.4747 - accuracy: 0.8176
```

2s 3ms/step - loss: 0.4729 - accuracy: 0.8191.
In the second trial the no. of neurons in each layer are modified 12, 8,1.

```
""" ANN """
import keras
from keras.models import Sequential
from keras.layers import Dense

input_dim = X_train.shape[1]
# print(input_dim)

classifier = Sequential()
classifier.add(Dense(12, kernel_initializer='uniform', activation = 'relu',input_dim = input_dim))
classifier.add(Dense(8,activation = 'relu'))
classifier.add(Dense(1,activation = 'sigmoid'))

#compile
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

#fit model

hist = classifier.fit(X_train, y_train, batch_size = 10, epochs = 10)
```

```
Epoch 1/10
9785/9785 [==============================] - 20s 2ms/step - loss: 0.4776 - accuracy: 0.8174
Epoch 2/10
9785/9785 [==============================] - 17s 2ms/step - loss: 0.4756 - accuracy: 0.8174
Epoch 3/10
9785/9785 [==============================] - 17s 2ms/step - loss: 0.4755 - accuracy: 0.8174
Epoch 4/10
9785/9785 [==============================] - 17s 2ms/step - loss: 0.4755 - accuracy: 0.8174
Epoch 5/10
9785/9785 [==============================] - 17s 2ms/step - loss: 0.4753 - accuracy: 0.8174
Epoch 6/10
9785/9785 [==============================] - 17s 2ms/step - loss: 0.4753 - accuracy: 0.8174
Epoch 7/10
9785/9785 [==============================] - 20s 2ms/step - loss: 0.4751 - accuracy: 0.8174
Epoch 8/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4750 - accuracy: 0.8174
Epoch 9/10
9785/9785 [==============================] - 17s 2ms/step - loss: 0.4749 - accuracy: 0.8174
Epoch 10/10
9785/9785 [==============================] - 17s 2ms/step - loss: 0.4747 - accuracy: 0.8174
```

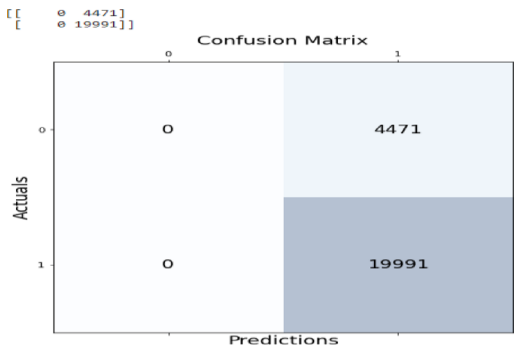
The third trial is depicted below:

```
""" ANN """
import keras
from keras.models import Sequential
from keras.layers import Dense

input_dim = X_train.shape[1]
# print(input_dim)

classifier = Sequential()
classifier.add(Dense(64, kernel_initializer='uniform', activation = 'relu',input_dim = input_dim))
classifier.add(Dense(64, activation = 'relu'))
classifier.add(Dense(1,activation = 'sigmoid'))

#compile
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

#fit model

hist = classifier.fit(X_train, y_train, batch_size = 10, epochs = 10)
```

```
Epoch 1/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4771 - accuracy: 0.8172
Epoch 2/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4760 - accuracy: 0.8174
Epoch 3/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4758 - accuracy: 0.8174
Epoch 4/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4756 - accuracy: 0.8174
Epoch 5/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4752 - accuracy: 0.8174
Epoch 6/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4751 - accuracy: 0.8174
Epoch 7/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4747 - accuracy: 0.8174
Epoch 8/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4740 - accuracy: 0.8174
Epoch 9/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4721 - accuracy: 0.8174
Epoch 10/10
9785/9785 [==============================] - 19s 2ms/step - loss: 0.4714 - accuracy: 0.8174
765/765 [==============================] - 1s 2ms/step - loss: 0.4748 - accuracy: 0.8172
[0.47479599714279175, 0.8172267079353333]
```

Score	0.8172267079353333
f1_score	0.47479599714279175

Confusion matrix for the ANN is shown below:



Limitations and reliability discussion

This study demonstrates the feasibility of using machine learning to detect DDoS attacks in smart home environments; however, several factors limit the reliability and generalizability of the results.

Dataset Scope:

The dataset was limited to approximately 10 h and 45 min of normal activity and 42 min of simulated DDoS traffic collected from a single smart home setup. This narrow scope constrains the diversity of traffic behavior. While the k-NN model achieved 97.13% test set accuracy, demonstrating effective generalization within this specific environment, the limited dataset size, single-location collection, and restricted device diversity (three IoT devices) may not fully represent the range of traffic patterns, device behaviours, and attack variants present in diverse real-world smart home deployments. Accordingly, the results are interpreted as a validated proof of feasibility rather than a definitive benchmark. Future work will expand the dataset by capturing multi-day, multi-device smart home traffic from multiple households and incorporating publicly available benchmark datasets (e.g., CIC-IoT 2023, TON_IoT) to validate cross-domain generalizability and robustness across varied network conditions and device ecosystems.

Class Imbalance:

The dataset exhibits a significant imbalance (81.8% attack vs. 18.2% normal traffic), which influenced model behaviour. While k-NN successfully handled this imbalance, the ANN model collapsed to predicting only the majority class, highlighting the need for balanced training data or specialized techniques (e.g., SMOTE, class weighting) in future work.

ANN Model Instability:

The ANN model exhibited systemic misclassification, producing no True Negatives or False Negatives. This outcome indicates a model collapse rather than standard underperformance, attributable to data imbalance, insufficient tuning, and a shallow network configuration. Planned corrective actions include applying dropout and regularization, balancing class distributions, and conducting systematic hyperparameter optimization (e.g., of activation functions, optimizers, learning rates, and batch sizes). The investigation of more advanced architectures, including CNN-LSTM hybrids and transfer learning from related IoT datasets, is also planned.

Feature Constraints:

Only six features (No., Time, Source, Destination, Protocol, Length) were used in this study to maintain a lightweight design for resource-constrained smart home devices. However, this limited feature set restricts the models' ability to capture temporal and behavioural patterns characteristic of sophisticated DDoS traffic. Future work will expand feature engineering to include statistical and flow-based metrics such as packet inter-arrival times, flow durations, port entropy, and session-based correlation measures.

These planned actions are intended to improve data robustness, model reliability, and real-world applicability in subsequent research phases. The limitations have been explicitly acknowledged here to ensure a transparent interpretation of the reported results and to guide reproducible experimentation in future studies.

Conclusion and future research

This study presented a comparative analysis of AI models for DDoS detection using a real-world smart home dataset, offering practical insights often absent in benchmark-based evaluations. The core finding is that within the data-constrained, resource-limited environment of a real smart home, a lightweight traditional model (k-NN) significantly outperformed a more complex deep learning model (ANN). This result challenges the assumption that deeper networks are inherently superior and underscores the critical importance of using representative data for model evaluation.

The k-NN model achieved a verified test accuracy of 97.13%, demonstrating that lightweight algorithms can provide reliable DDoS detection even with limited, domain-specific data. While slightly below the ≈98–99% accuracies reported on large centralized datasets, this performance highlights k-NN's practical viability and computational efficiency for edge-based cybersecurity. Furthermore, k-NN proved resilient to class imbalance, maintaining balanced precision and recall, which is crucial for minimizing false alarms in real-world deployment.

In contrast, the ANN model exhibited systemic misclassification, primarily due to dataset size, class imbalance, and architectural constraints. This does not refute the potential of deep learning but emphasizes its dependency on large, rich datasets for effective generalization. Therefore, this work should be interpreted as a proof-of-concept demonstrating methodological feasibility.

To address the identified limitations and enhance generalizability, future research will focus on:

1. *Dataset Expansion and Diversity* Extend data collection to include multi-day traffic from diverse smart home environments and incorporate benchmark datasets (e.g., CIC-IoT 2023, TON_IoT) for cross-validation.
2. *Feature Enrichment* Expand feature engineering to include packet inter-arrival times, flow duration, port entropy, and other statistical attributes to improve model discriminative power.
3. *ANN Optimization* Conduct systematic hyperparameter tuning and explore hybrid architectures (e.g., CNN-LSTM) or transfer learning to overcome data scarcity.
4. *Robustness Testing* Evaluate models against multi-vector, slow-rate, and application-layer DDoS attacks to assess adaptability in complex threat scenarios.
5. *Edge Deployment* Implement and test lightweight models like k-NN on edge gateways to validate real-time performance in live networks.

This research contributes to smart home cybersecurity by providing empirical evidence and practical design guidance for AI-based DDoS detection at the network edge. It establishes a foundational framework for developing more intelligent, adaptive, and scalable defense mechanisms. Subsequent work will explore real-time adaptive ML pipelines and lightweight federated learning to further advance smart home security resilience.

Analysis of metrics in class imbalance context

A detailed analysis of the classification metrics in Table 5 reveals critical insights beyond accuracy. The k-NN model demonstrates robust performance across all metrics, with high F1-scores for both the minority 'Normal' class (0.92) and the majority 'Attack' class (0.98). This indicates a strong ability to distinguish between classes despite the imbalance.

In stark contrast, the ANN model's metrics expose its fundamental flaw: it failed to learn the characteristics of the 'Normal' class. With precision and recall of zero for 'Normal' traffic, the model defaulted to predicting all traffic as an 'Attack'. This is a classic symptom of a model collapsing under class imbalance, where it optimizes for the majority class. While its accuracy was 81.7%, this figure is misleadingly high and simply reflects the underlying data distribution. The high recall for the 'Attack' class (1.00) confirms that it caught all attacks, but at the cost of a 100% false positive rate for normal traffic, which is unacceptable for a practical system."

Model	Accuracy (%)	Precision (Normal)	Recall (Normal)	F1-Score (Normal)	Precision (Attack)	Recall (Attack)	F1-score (Attack)
k-NN	97.13	0.94	0.90	0.92	0.98	0.99	0.98
ANN	81.7	0.00	0.00	0.00	0.82	1.00	0.90

Table 5. Comprehensive performance metrics for k-NN and ANN models.

When compared with earlier DDoS detection studies that achieved between 98% and 99% accuracy on large benchmark datasets such as UNSW-NB15[38], CIC-IDS 2017[39], and NSL-KDD[40], the present k-NN model's verified test accuracy of 97.13% remains competitive. The small difference can be attributed to the limited size (≈ 11 h of traffic) and heterogeneity of the self-collected smart-home dataset, which better represents realistic residential environments than standardized laboratory data.

This finding confirms that lightweight ML models such as k-NN can achieve high reliability in detecting DDoS attacks even under restricted data and hardware resources. In contrast, the ANN model, which attained 82% accuracy, underperformed due to the dataset's scale and class imbalance, conditions known to affect deep-learning generalization.

Overall, the proposed approach complements prior benchmark-based research by validating that a simple, interpretable algorithm can deliver near-state-of-the-art performance in edge-constrained smart-home networks.

Critical evaluation and discussion

This study demonstrates that for DDoS detection in resource-constrained smart home environments with limited data, a lightweight, traditional machine learning model (k-NN) can significantly outperform a more complex deep learning model (ANN).

Performance evaluation

The main performance metrics to consider in this work is model accuracy and loss scores as shown below:

Model	Accuracy score	Loss score
k-NN	0.971	0.015
ANN	0.82	0.47

The ANN model achieved an accuracy of approximately 82%, whereas the k-NN model significantly outperformed it with an accuracy exceeding 97%.

k-NN is an instance-based learner that performs well on smaller, cleaner datasets where class boundaries can be defined by a simple distance metric. Its local decision-making process, based on k-nearest neighbours, made it more resilient to the global class imbalance that severely impacted the ANN. Furthermore, its algorithmic simplicity aligns directly with the "lightweight" and "edge-deployable" objectives outlined in the introduction.

In contrast, the ANN, in its goal to minimize the overall loss function, converged on the trivial solution of predicting only the majority class. This is conclusively demonstrated in Table 5 by the precision and recall of zero for the "Normal" class. Deep learning models generally require large volumes of data to learn meaningful feature representations; the 11-hour dataset was simply insufficient for the ANN to generalize, leading to the model collapse observed. Although the 3-layer ANN was selected for its lightweight properties, its architecture may have been too simplistic to capture the necessary patterns from the available data. This stands in opposition to k-NN, which makes no assumptions about the underlying data distribution.

Overall evaluation

A comparative analysis of the confusion matrices provides a clear performance distinction. The k-NN model successfully learned to discriminate between both classes, as evidenced by a high number of True Positives and True Negatives. In contrast, the ANN model exhibited a pathological failure mode, producing zero True Negatives and zero False Negatives. This indicates it defaulted to predicting all traffic as Class 1 (Attack), effectively becoming a constant function that performs no meaningful classification.

Despite extensive parameter tuning, including increasing the number of epochs to 50, the ANN's accuracy plateaued at approximately 82%. This figure is misleading, as it solely reflects the accuracy of predicting the majority class. The model's inability to improve, coupled with its zero precision and recall for the 'Normal' class and a consequently low F1-score, confirms its ineffectiveness for this specific task under the given data constraints. In contrast, k-NN demonstrated robust performance across all metrics, including F1-score, precision, and recall for both classes.

Regarding the experimental setup, the DDoS attacks were generated using the LOIC tool, which successfully produced malicious traffic patterns captured by Wireshark for model training. It is noteworthy that the smart home system itself demonstrated inherent resilience to these low-to-moderate level SYN flood attacks, as no service disruption or delay was observed for legitimate requests. This resilience can be attributed to the cloud-backed security of the platform (e.g., AWS). Consequently, more severe attacks that could potentially cause service disruption were not conducted due to ethical and legal research limits. The primary objective of this phase was to capture the *pattern* of attack traffic for model training, not to assess the infrastructure's breaking point.

	precision	recall	F1-score	Support
0	0.94	0.90	0.92	4458
1	0.98	0.99	0.98	20,004

The superior performance of k-NN can be theoretically attributed to its instance-based learning mechanism and low model variance, which perform well in small, clean datasets with clearly separable class boundaries.

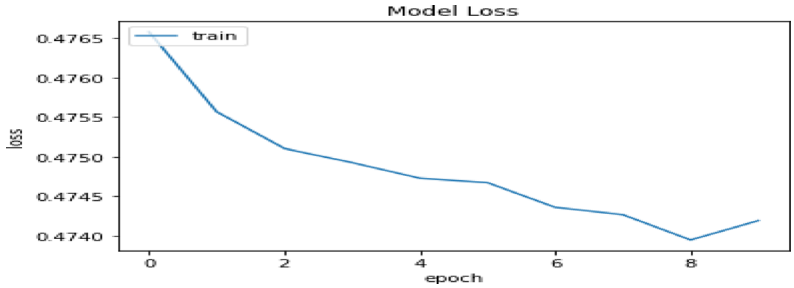


Fig. 29. ANN model loss. The accuracy values reported here (97.13% for k-NN, 82% for ANN) are derived directly from the test-set confusion matrices; earlier drafts that cited 99% reflected preliminary training-set performance.

Conversely, the ANN model, with its higher capacity and reliance on large-scale training data, may have suffered from underfitting due to the limited sample size.

Future experiments will test both models against diverse DDoS vectors, including multi-vector, application-layer, and slow-rate attacks, to assess model adaptability in realistic threat conditions.

Analysis of computational efficiency and deployment feasibility

The performance of a detection model is not solely defined by its accuracy but also by its operational cost in terms of latency and computational resources. This is a critical consideration for smart home gateways and edge devices, which typically have limited processing power (CPU), memory (RAM), and energy budgets. While an empirical hardware benchmark is a valuable direction for future work, we can analyse the inherent computational characteristics of each model to assess their deployment feasibility.

The k-NN algorithm's inference process is computationally straightforward, primarily involving the calculation of Euclidean distances between a new data point and all points in the training set. Although this suggests a complexity of $O(n)$ per prediction (where *n* is the training set size), this cost is manageable in our context for two key reasons: First, our feature set is extremely small (only 6 features), keeping the distance calculation lightweight. Second, and more importantly, the entire training set for a single smart home environment, after our feature selection, would be compact enough to reside in the memory of a modern microcontroller or gateway without issue. The resulting inference latency is predictable and low, making k-NN highly suitable for the near-real-time demands of a smart home network security monitor.

In contrast, the ANN model, despite its compact architecture, incurs a fixed cost of conducting forward propagation through its network of layers, activation functions, and weighted sums. This involves a significant number of floating-point operations (FLOPs) even for a small network. While the absolute inference time might still be measured in milliseconds on capable hardware, the relative computational intensity, memory bandwidth usage, and energy consumption are inherently higher than the simple distance calculations of k-NN. This makes the ANN a less optimal choice for the most constrained devices.

Therefore, this architectural analysis reinforces our core finding from a performance perspective. The k-NN model not only achieved superior accuracy on our limited dataset, but its algorithmic simplicity also translates to lower computational overhead, lower power consumption, and more predictable latency. This combination of high detection performance and low operational cost makes k-NN a strongly recommended and feasible candidate for direct integration into the resource-constrained environment of a smart home gateway.

Diagnostic analysis of ANN underperformance

The ANN model's significantly lower accuracy (81.7%) compared to the k-NN model warrants a detailed diagnostic analysis to determine the root causes, specifically investigating the roles of limited training data, architectural choice, and overfitting.

- *Ruling Out Overfitting* The training and validation loss/accuracy curves (Fig. 29) show a clear signature of underfitting, not overfitting. Both training and validation accuracy plateaued at approximately 82% without a significant gap, indicating that the model failed to learn meaningful patterns from the training data itself, rather than memorizing it and failing to generalize.
- *The Impact of Limited Training Data and Class Imbalance* The primary cause of failure was model collapse due to the severe class imbalance (81.8% attack traffic), which was exacerbated by the limited dataset size. Deep learning models like ANN require large, balanced datasets to learn complex feature hierarchies. Our dataset of ~122,000 packets, while realistic for a smart home, was insufficient in both size and balance for the ANN to converge on a useful decision boundary. The model found the simplest path to minimize its loss function: always predicting the majority 'Attack' class. This is conclusively demonstrated by the confusion matrix and per-class metrics, which show a recall of 1.00 for the 'Attack' class and 0.00 for the 'Normal' class.
- *Assessment of Architecture Choice* Regarding architecture, our 3-layer feedforward network was chosen to be lightweight and suitable for edge deployment. While a more complex architecture (e.g., with more layers, dropout, or LSTM nodes) might have had a better capacity to learn, it would have also required more data to train effectively and increased the risk of overfitting on our small dataset. Therefore, the architecture was

a reasonable choice for the deployment context, but it could not compensate for the fundamental data constraints.

The ANN's poor performance is not a simple matter of low accuracy but a case of catastrophic underfitting driven by dataset limitations. The model was starved of both the volume and the balance of data required for effective deep learning. This result critically informs the practical recommendation of our work: in data-scarce, imbalanced environments like the one studied, investing in complex model architectures is unlikely to yield returns without first solving the fundamental data acquisition and balancing challenges.

When evaluated on our real smart home dataset, the k-NN model's superiority highlights a key practical insight that in resource-constrained IoT environments, model simplicity and computational efficiency can be more critical than theoretical complexity for achieving reliable security. The failure of the ANN model in our specific context, a real smart home environment with limited, imbalanced data, underscores the danger of directly applying models trained on ideal benchmark data to real-world IoT scenarios. Crucially, our unique experimental setup, using real smart home traffic, reveals that model complexity is not a guarantee of performance in resource-constrained environments.

Data availability

Data supporting this study are openly available upon reasonable request.

Received: 27 July 2025; Accepted: 7 December 2025
Published online: 17 January 2026

References

1. Mir, S. & Quadri, S. Information availability: An insight into the most important attribute of information security. *J. Inform. Secur.* **7**, 185–194 (2016).
2. Ahmad, R. et al. Zero-day attack detection: A systematic literature review. *Artif. Intell. Rev.* **56**, 10733–10811 (2023).
3. Devi, B. S. et al. An impact analysis: Real time DDoS attack detection and mitigation using machine learning. in *Proceedings of the 2014 Intl. Conference on Recent Trends in Information Technology Chandigarh, India* 1–7 (2014).
4. Gurinaviciute, J. 5 biggest cybersecurity threats. (2025). https://www.securitymagazine.com/articles/94506-5-biggest-cybersecurity-threats. Accessed 10 July 2024.
5. Tripathi, N. & Mehtre, B. DoS and DDoS attacks: Impact, analysis and countermeasures. in *Proc. of Conference on Advances in Computing, Networking and Security Mangalore, India* 1–6 (2013).
6. Rajan, D. M. & Sathya, Priya, S. DDoS mitigation techniques in IoT: A survey. in *2022 Proc. of International Conference on IoT and Blockchain Technology (ICIBT). Ranchi, India* 1–7 (2022). https://doi.org/10.1109/ICIBT52874.2022.9807799.
7. Kaspersky What is a Botnet? (2021). https://www.kaspersky.co.uk/resource-center/threats/botnet-attacks Accessed 25 October 2023.
8. Bederna, Z. & Szadeczky, T. Cyber espionage through botnets. *Secur. J.* **33**, 43–62 (2020).
9. Lupton, D., Pink, S. & Horst, H. Living in, with and beyond the 'smart home': introduction to the special issue. *Conv* **27**(5), 1147–1154 (2021).
10. Alaa, M. et al. A review of smart home applications based on internet of things. *J. Netw. Comput. Appl.* **97**, 48–65 (2017).
11. Madakam, S. & Ramaswamy, R. Smart homes (conceptual views). in *Proc. of the 2014 2nd International Symposium on Computational and Business Intelligence*. 63–66 (Washington, DC, USA, 2014).
12. Sambangi, S. & Gondi, L. A machine learning approach for DDoS (Distributed Denial of Service) attack detection using multiple linear regression. *Proceedings* **63**(1), 51. https://doi.org/10.3390/proceedings2020063051
13. Abiramasundari, S. & Ramaswamy, V. Distributed denial-of-service (DDOS) attack detection using supervised machine learning algorithms. *Sci. Rep.* **15**, 13098 (2025).
14. van Engelen, J. E. & Hoos, H. H. A survey on semi-supervised learning. *Mach. Learn.* **109**, 373–440 (2020).
15. Martin, D. 8 Benefits of using AI for cybersecurity (2021). http://www.cm-alliance.com/cybersecurity-blog/8-benefits-of-using-ai-for-cybersecurity Accessed 25 December 2024.
16. Wanda, P. & Hiswati, M. E. Belief-DDoS: Stepping up DDoS attack detection model using DBN algorithm. *Int. j. inf. Tecnol.* **16**, 271–278. https://doi.org/10.1007/s41870-023-01631-x (2024).
17. Aguru, A. D. & Erukala, S. B. A lightweight multi-vector DDoS detection framework for IoT-enabled mobile health informatics systems using deep learning. *Inf. Sci.* **662**, 120209. https://doi.org/10.1016/j.ins.2024.120209 (2024).
18. Shirvani, G. Enhancing IoT Security Against DDoS Attacks through Federated Learning. arXiv preprint arXiv:2403.10968 (2024). https://arxiv.org/abs/2403.10968?utm_source=chatgpt.com Accessed 1 November 2025.
19. Chidananda, A. J., Murthy, P. & Madhu, B. R. Prevent DDOS attack in cloud using machine learning. *Int. J. Adv. Res. Comp. Sci. Soft Eng.* **6**(6), 575–579 (2016).
20. Glavan, D. et al. : DDoS detection and prevention based on artificial intelligence techniques. *Sci. Bull. Mircea Cel Batran Naval Acad.* **22**(1), 1–11 (2019).
21. Khalaf, B. A. et al. Comprehensive review of artificial intelligence and statistical approaches in distributed denial of service attack and defense methods. *IEEE Access.* **7**, 51691–51713. https://doi.org/10.1109/ACCESS.2019.2908998 (2019).
22. Zhang, B., Zhang, T. & Yu, Z. DDoS detection and prevention based on artificial intelligence techniques. in *Proc. of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC). Chengdu, China* 1276–1280 (2017).
23. Alzahrani, S. & Hong, L. Detection of distributed denial of service (DDoS) attacks using artificial intelligence on cloud. in *Proc. of the 2018 IEEE World Congress on Services (SERVICES)* 35–36 (San Francisco, CA, USA, 2018).
24. Saied, A., Overill, R. E. & Radzik, T. Detection of known and unknown DDoS attacks using artificial neural networks. *Neurocomputing* **172**, 385–393 (2016). https://www.sciencedirect.com/science/article/pii/S092523121501053X
25. Mantas, G., Lymberopoulos, D. & Komninos, N. Security in smart home environment. in *Proc. of the 2010 Wireless technologies for ambient assisted living and healthcare: Systems and applications* 170–191 (IGI Global, 2011).
26. Coboi, A. E., Tran, T. A. & Son, T. Q. Nguyen M. Security problems in smart homes. *ICSES Trans. Comput. Netw. Commun.* 1–9 (2021).
27. Tushir, B., Dalal, Y. & Dezfouli, B. A quantitative study of DDoS and E-DDoS attacks on WiFi smart home devices. *IEEE Internet Things J.* **8**(8), 6282–6292. https://doi.org/10.1109/JIOT.2020.3048883 (2021).
28. Saxena, U., Sodhi, J. S. & Singh, Y. An analysis of DDoS attacks in a smart home networks. in *Proc. International Conference on Cloud Computing, Data Science & Engineering, Uttar Pradesh, India* 272–276 (2020). https://doi.org/10.1109/Confluence47617.2020.9058087.

29. Jeyanthi, N. Internet of things (IoT) as interconnection of threats (IoT). in *Security and Privacy in Internet of Things* (IoTs) (CRC, 2016).
30. Wullems, C., Clark, A. J. & Looi, M. A trivial denial of service attack on IEEE 802.11 direct sequence spread spectrum wireless LANs. in *Proc. 2004 Australasian Conf. Inf. Secur. Privacy (ACISP)* 444–448 (Sydney, Australia, 2004).
31. Stetsko, A., Folkman, L. & Matyáš, V. Neighbor-based intrusion detection for wireless sensor networks. in *Proc. of the 2010 6th International Conference on Wireless and Mobile Communications (ICWMC)* 420–425 (Valencia, Spain, 2010). https://doi.org/10.1109/ICWMC.2010.61.
32. Lemos, M. V. D. S., Leal, L. B. & Filho, R. H. A new collaborative approach for intrusion detection system on wireless sensor networks. in *Novel Algorithms and Techniques in Telecommunications and Networking* (eds Sobh, T., Elleithy, K., Mahmood, A.) 239–244 (2010).
33. Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A. & Knightly, E. DDoS-shield: Dsos-resilient scheduling to counter application layer attacks. *IEEE/ACM Trans. Netw.* **17**(1), 26–39 (2009).
34. Gordon, H., Batula, C., Tushir, B., Dezfouli, B. & Liu, Y. Securing smart homes via software-defined networking and low-cost traffic classification. in *Proc. of 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)* 1049–1057 (Madrid, Spain, 2021). https://doi.org/10.1109/COMPSAC51774.2021.00143.
35. Owusu, A. I. & Nayak, A. An intelligent traffic classification in sdniot: A machine learning approach. in *Proc. of the 2020 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom). Virtual Conference* 1–6 (2020).
36. Reza, M., Sobouti, M. J., Raouf, S. & Javidan, R. Network traffic classification using machine learning techniques over software defined networks. *Int. J. Adv. Comput. Sci. Appl.* **8**(7), 220–225. https://doi.org/10.14569/IJACSA.2017.080729 (2017).
37. Xu, J., Wang, J., Qi, Q., Sun, H. & He, B. Deep neural networks for application awareness in sdn-based network. in *Proc. of 2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)* 1–6 (London, UK, 2018).
38. Hamza, A., Gharakheili, H. H., Benson, T. A. & Sivaraman, V. Detecting volumetric attacks on lot devices via SDN-based monitoring of mud activity. in *Proc. of the 2019 ACM Symposium on SDN Research* 36–48 (San Jose, CA, USA, 2019).
39. Doshi, R., Apthorpe, N. & Feamster, N. Machine learning DDoS detection for consumer internet of things devices. in *Proc. of 2018 IEEE Security and Privacy Workshops* (SPW) 29–35 (San Francisco, CA, USA, 2018).
40. Yang, L. & Zhao, H. DDoS attack identification and defense using SDN based on machine learning method. in *Proc. of the 15th 2018 International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN)* (Yichang, China. 174–178, 2018).
41. Wali, A., Apejoye, O., Raja, T., He, J. & Ma, X. A novel approach to identifying DDoS traffic in the smart home network via exploratory data analysis. in *Proc. of the 2022 Applied Intelligence and Informatics* Vol. 1724 478–498 (Springer, Cham, 2022). https://doi.org/10.1007/978-3-031-24801-6_34.
42. Raja, T. V. et al. : Detection of DDoS attack on smart home infrastructure using artificial intelligence models. in *Proc. of the 2022 International Conference on Cyber-enabled Distributed Computing and Knowledge Discovery (CyberC)* 12–18 (Suzhou, China, 2022).
43. He, J. August : CRISP-DM and case study 1. COMP20121 Machine learning for data analytics. (2024). https://www.kaggle.com/code/jhskaggle/l02-crisp-dm/notebook.
44. Chakure, A. Data preprocessing https://medium.datadriveninvestor.com/data-preprocessing-3cd01eefd438
45. Luna, Z. & Understanding CRISP-DM and its importance in data science projects. https://medium.com/analytics-vidhya/understanding-crisp-dm-and-its-importance-in-data-science-projects-91c8742c9f9b Accessed 15 Aug 2024.
46. Wali, A., Apejoye, O., He, J. & Ma, X. An exploratory data analysis of the network behavior of Hive home devices. in *Proc. of the 2021 8th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)* 1–8 (Malmö, Sweden, 2021). https://doi.org/10.1109/IOTSMS53705.2021.9704944.
47. Baladram, S. Categorical encoding using label-encoding and one-hot-encoder. https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd Accessed 26 November 2024.
48. Aniruddha, B. Feature scaling for machine learning: Understanding the difference between normalization vs. standardization. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/ Accessed 30 Sept 2023.
49. Kamiran, F. & Calders, T. Data preprocessing techniques for classification without discrimination. *Knowl. Inf. Syst.* **33**, 1–33 (2012).
50. Bhaumik, U., Singh, K. U., Akbari, A. S. & Bajpai, M. K. A deep learning–based approach to detect correct Suryanamaskara pose. *SN Comput. Sci.* **3**, 337. https://doi.org/10.1007/s42979-022-01226-6 (2022).
51. Guo, G. et al. : K-NN model-based approach in classification. in *Proc. of the 2003 On the Move to Meaningful Internet Systems* Lecture Notes in Computer Science Vol. 2888, 986–996 (2003). https://doi.org/10.1007/978-3-540-39964-3_62.
52. Jeswal, S. K. & Chakraverty, S. Chapter 10 - Fuzzy eigenvalue problems of structural dynamics using ANN. in *New Paradigms in Computational Modeling and Its Applications* (ed Chakraverty, S.) 145–161 (Academic Press, 2021). https://doi.org/10.1016/B978-0-12-822133-4.00010-4.
53. Mhatre, M. S., Siddiqui, F., Dongre, M. & Thakur, P. A. Review paper on artificial neural network: A prediction technique. *Int. J. Sci. Eng. Res.* **8**(3), 1–3 (2017).
54. Ali, M. et al. An AI based approach to secure SDN enabled future avionics communications network against DDoS attacks. in *Proc. of the 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)* 1–7 (San Antonio, TX, USA, 2019). https://doi.org/10.1109/DASC43569.2019.9081639.

Author contributions

T.V.R. conceived the presented idea and developed the theory and performed the computations. A.W.Z. helped in refining the computation and drawing the graphs. J.H. and X.M. drafted the research proposal, verified the design and analytical methods as well as supervised the findings of this work. Z.E. reviewed the entire work and selected the sections that needed to be focused on for publication purposes. All authors discussed the results and contributed to the final manuscript.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to Z.E.

Reprints and permissions information is available at www.nature.com/reprints.