



ARTICLE

A Federated Learning Based Deep Autoencoder Framework for Robust Malware Detection in Edge Cloud Networks

Munam Ali Shah^{1,*} and Shazil Gul²

¹Department of Computer Networks and Communication, CCSIT, King Faisal University, Al-Ahsa, Saudi Arabia

²Department of Computer Science, COMSATS University Islamabad, Islamabad, Pakistan

*Corresponding Author: Munam Ali Shah. Email: mashah@kfu.edu.sa

Received: 08 March 2026; Accepted: 07 August 2026; Published: 22 October 2026

ABSTRACT: The edge cloud-based Internet of Things (IoT) devices are diverse in size, type, and the function it performs. The tremendous increase in the number of cyberattacks, coupled with the adverse technologies, has rendered current security measures increasingly ineffective. Addressing security and privacy issues in edge cloud networks using machine learning and deep learning-based solutions is effective up to a certain extent; however, these intrusion detection solutions (IDS) rely heavily on the quantity and quality of data. This dependency results in high inaccuracy and high false positive rates in identifying malware. Moreover, the existing machine learning-based malware detection techniques are less efficient and fail to detect malware in unknown or newly joining edge cloud IoT devices in the network. In this paper, our contribution is twofold: Firstly, a federated learning-based method using a deep autoencoder (DAE) has been proposed to detect malware attacks in the edge cloud network. This approach enhances privacy by ensuring that the device data is not transmitted or moved outside the network edge. Instead, the deep learning computation occurs where the data is generated, providing better data protection. Secondly, we have designed a novel method that utilizes DAE, which can effectively detect malware in unknown or new IoT devices. The experimental results show that the proposed DAE model has achieved an AUC of 89%. Similarly, we have compared the performance of our proposed model with other deep learning models such as DNN, CNN, and RNN, and the proposed model has 5% better accuracy than CNN, 17% better than DNN, and 21% better accuracy than RNN in detecting malware in both known and unknown devices.

KEYWORDS: Federated learning; malwares analysis; edge cloud; intrusion detection system (IDS); deep autoencoder; DNN; CNN; RNN

1 Introduction

The researchers predict that by 2027, the number of online IoT devices exceeds 64 billion [1]. Different types of devices connected together and transmitting data have given rise to the concept of Internet of Everything (IoE), which is a dynamic and heterogeneous network infrastructure [1,2]. The rapid expansion, the market size, and the distributed nature of the IoT and IoE present huge potential for cyber criminals. This is the reason these cyber hackers are paying more and more attention to breaching the security of the IoT for different purposes. The difficulty of mapping risks in linked devices expands due to the Internet's development and heterogeneity. Additionally, the proliferation of cyber attacks makes it quite possible to launch a network attack conveniently [2,3]. The use of IoT devices is expanding, and with that growth has come a significant increase in intrusion activities within the network traffic. Network traffic threats bring security hazards and fraud in any firm that deals with important information.

Today, preventing and protecting against risks from anomalous activities is primarily challenged by the requirement to identify abnormal events or behaviors. The most popular security mechanism uses a network intrusion detection system (NIDS) to track all activity to spot any suspicious activity [1]. Generally, a NIDS is a device or piece of software installed at network gateways that identifies whether or not the traffic is most likely an assault to safeguard server information systems. NIDS can be used with various strategies, including anomaly-based and signature-based systems. The fundamental conceptual difference between anomaly-based and signature-based approaches is that the latter seeks out specific, well-known attacks. At the same time, the former is ineffective at identifying unknown, uncharted hostile activity [2]. In contrast, applying the multi-classification model to labeled traffic data suggests that the attack has already been deployed. It will not be able to quickly identify any unusual new events that could cause data leakage. On the other hand, the previous system performs better in intrusion detection analysis and has a greater capacity for identifying any familiar attacks [2,4,5].

This research addresses this gap by using a dataset with unique characteristics from a medium-sized IoT network. This dataset aids in botnet/malware detection and IDS implementation, overcoming the limitations of current datasets. Furthermore, we employ a decentralized approach using Federated Learning (FL) DAE model to detect anomalies while preserving privacy. FL trains deep models across multiple clients without sharing data samples [6]. The federated DAE model is trained using federated averaging and federated averaging with momentum to test its effectiveness. This approach significantly enhances the robustness and efficiency of malware detection in IoT devices, demonstrating exceptional performance in differentiating between anomalous and benign activities [7].

Most malware detection in IoT devices has been using machine learning models, which are less efficient and pose a risk to data privacy [2,7–9]. In addition, some models address the data privacy problem by utilizing the Federated Learning technique; nevertheless, these approaches do not produce satisfactory results for detecting malware in unknown IoT devices in terms of the false positive rate [1]. This research pioneers the application of federated learning with deep autoencoders (DAE) for malware detection in edge cloud networks, addressing a critical gap in the literature regarding privacy-preserving methods. The research demonstrates unprecedented accuracy in detecting malware across known and unknown IoT devices, establishing a significant improvement of at least 5% over prior detection techniques that have been limited by smaller datasets and conventional approaches. The rest of the paper is organized as follows: [Section 2](#) presents the literature review. [Section 3](#) details the proposed solution, while [Section 4](#) describes Model testing. The results are discussed in [Section 5](#), followed by the case study and test cases in [Section 6](#). Lastly, the conclusions and future directions are presented in [Section 7](#).

2 Literature Review

Recent work on IoT malware detection shows clear progress, but three gaps remain: (i) weak detection of zero-day and previously unseen attacks, (ii) high false-positive rates in heterogeneous IoT traffic, and (iii) privacy risks in centralized training pipelines [3–6,9]. Prior studies have applied deep learning, behavior-based detection, and transfer learning to improve robustness, while federated learning has been adopted to reduce raw-data sharing across devices [10–13].

Federated and collaborative approaches improve privacy and scalability, but they can still suffer under non-IID data distributions, client drift, and poisoning/adversarial updates, which degrade global-model quality [8,11–14]. Existing approaches also report performance variation when moving from controlled environments to larger and more diverse IoT populations [15–19].

One of the existing works used a Medium-scale Botnet IoT (MedBIoT) dataset to build an intelligent IoT botnet detection system [20]. Similarly, some other research works performed malware analysis and

evaluated parameters such as accuracy, precision, F1 and recall, etc. [21–23]. There are similar studies that aim to evaluate malware and use machine learning or federated learning approaches [15,16,24,25]. Machine learning and deep learning significantly improve the detection of botnets and malware through advanced classification and feature extraction techniques [26–28]. Some of the other studies use supervised machine learning, LSTM-based and Transformer-based architectures to analyze and evaluate emerging cyber threats [29–31].

Recent FL-based IDS and anomaly-detection systems demonstrate encouraging outcomes, including improved malware discrimination and reduced centralized data exposure [32,33]. However, consistent detection of malware in unknown/newly joining IoT devices remains insufficiently addressed in many studies, especially when balancing accuracy, false-positive rate, and communication cost [1,17,19].

Recent state-of-the-art studies further reinforce this direction. Hybrid FL deep models have shown improved multi-class intrusion detection in IIoT settings [34], while ResNet-based FL architectures have addressed high-dimensional traffic and privacy constraints in industrial IoT networks [35]. More recent 2025 studies report stronger cross-device generalization with LSTM-based FL across multiple IoT datasets [36], improved performance under data-scaling constraints for resource-limited devices [37], and optimized FL-IDS designs for IoT environments [38]. The existing studies clearly lag in detecting malware in new devices with no background knowledge and are completely unknown to the detection systems. This research focuses on the detection of malware in both existing and new devices, making the defense system stronger and more resilient against attacks.

3 Proposed Solution

This paper designs and implements a deep learning model to detect malware in new IoT devices, facilitating secure and privacy-preserving communication. The proposed methodology combines empirical and experimental research under a quantitative framework using high-computation servers in a controlled environment with the primary objective of detecting malware in both seen/known and unseen/unknown IoT devices using a federated learning approach. The subsequent sections provide further details of the proposed model.

3.1 Design Details

In this research, we used standard approaches, i.e., DNN, CNN, and RNN, for anomaly detection. Note that applying the DAE model can significantly improve the model's ability to work with non-linear representations. Secondly, the reconstruction error is utilized for the purpose of determining whether the detected behaviour is one of the normal or abnormal devices. In addition, implementing the local model into the FL model resulted in three significant accomplishments: the preservation of users' privacy, increased operational efficacy, and increased robustness, as also analyzed in [32]. We have performed the following traditional steps, i.e., collection and analysis of data; Extraction of features from the dataset; Training the Auto-encoders on benign binaries; calculating the reconstruction error; predicting malware and benign binaries and applying a threshold; and lastly, identifying behaviors in unknown/unseen devices.

Data Collection

In this research, the MedBIoT dataset is known as a medium-sized network. This dataset is used in many studies, such as Guerra-Manzanares et al., [19]. This architecture is made up of both botnet malware activity and legitimate traffic. After installing three distinct types of IoT botnet malware in both emulated and real

IoT devices, data is collected from the early phases of botnet deployment. These stages include infection, diffusion, and contact with the command-and-control server.

For malware and botnet analysis, different datasets are available that include multiple IoT malware families and attack scenarios, as summarized in [Table 1](#). The malware families reported in the referenced datasets include Hajime, Aidra, BashLite, Mirai, Doflo, Tsunami, and Wroba, among others. The data sets that are currently available are summarized in [Table 1](#) [19].

Table 1: Datasets for IoT anomaly-based IDS.

Dataset	Botnet	Number of Devices	Device Type	Real or Emulated	Network Size	Dataset Features	Ref.
N-BaIoT	Mirai, BashLite	9 or 30%	Doorbell, Webcam, Thermostat, Baby Monitor, Security Camera	Real	Small	115 statistics	[11]
MedBIoT	Hajime, Aidra, BashLite, Mirai, Doflo, Tsunami, Wroba	2 or 20%	Multimedia Center, Security Camera	Emulated	Small	PCAP & Net-flow/Host	[19]
MedBIoT Dataset	Mirai	83 or 23%	IoT devices	Real	Small to Medium	Command and Control	[20]
Bot-IoT	No actual malware (simulated)	5 or 25%	Refrigerator, Smart Garage Door, Smart Lights, Smart Thermostat	Emulated	Small	115 statistics	[22,23]

The data sets concentrate on small-sized networks, which typically have fewer than ten devices, and they use either emulated or real devices. As a result, they only provide a limited number of interactions between devices that are contained within the network.

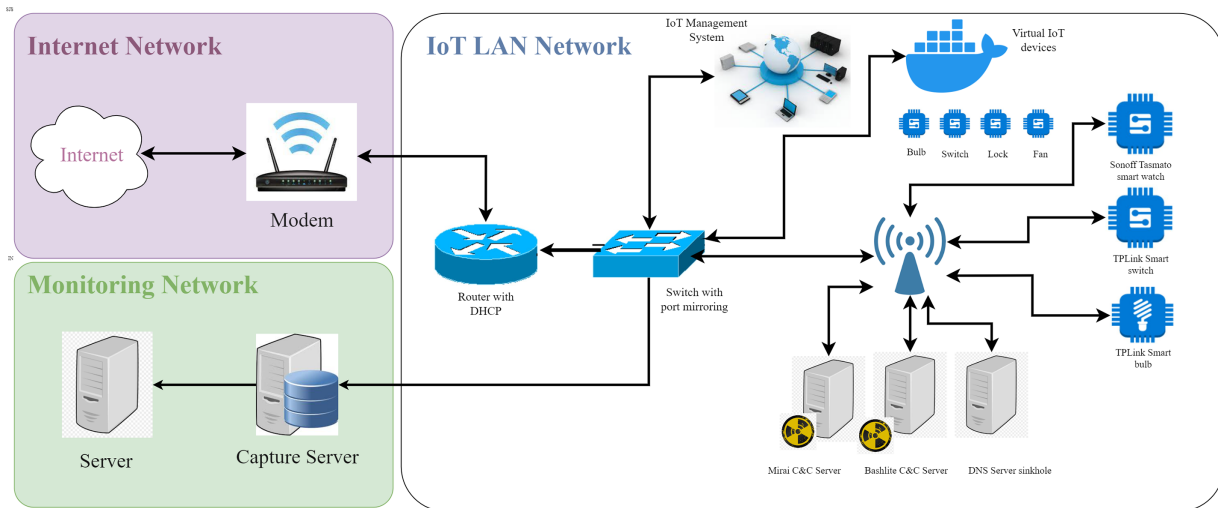
3.2 Data Analysis

A total of 83 devices are used across a medium-sized network, consisting of virtual devices that are generated and deployed together with real devices that are also used. Note that 80 of the devices are simulations, while the remaining 3 are the real thing. Although, 80 devices in MedBIoT are emulated, their traffic patterns are generated using device-specific behavioral profiles validated in prior work. These profiles preserve the inherent Non-IID characteristics across clients, including protocol diversity, traffic periodicity, and benign-malicious distributional shifts. This environment is used to illustrate both typical and actual botnet malicious network traffic. In the early stages of Torii infection, before any subsequent botnet activities could occur, the investigators working on BashLite and Mirai focused their attention on the data obtained from the networks that connected all the endpoints and servers. The table that follows provides an overview of the number of attack traffic data and benign traffic data for Internet of Things devices [Table 2](#) [19].

Table 2: Analysis of the data traffic.

Number of Packets	Traffic Type	Number of Devices
4,143,276	BashLite	40
842,674	Mirai	25
319,139	Torii	12
12,540,478	Benign	83

In order to make subsequent data analysis more straightforward, the source codes for Mirai and BashLite have been set to facilitate the transmission of various kinds of communications across a variety of ports. In this context, harmful traffic analysis reveals that 68% of the data gathered is associated with the activity of malware propagation, while the remaining 32% is related to the communication that takes place between C&C servers and bots. Because the real CC was prevented from reaching the device by countermeasures, malicious traffic in the scenario of Torii only includes data pertaining to the initial infection of the device. This is because the containment measures did not allow subsequent botnet events, such as propagation, to be triggered. The data collection that was developed is made available to the general public at the following link: <https://cs.taltech.ee/research/data/medbiot/>. The architecture of MedBloT is shown in Fig. 1.

**Figure 1:** An example of a medium-sized IoT network.

3.3 Features

A total of one hundred statistical features on the network's traffic are computed. The capabilities are summarized in Table 3 [32], which may be found here. It is clear that statistical features are computed with respect to the four primary categories for each of the five time frames (i.e., 100 ms, 500 ms, 1.5 s, 10 s, and 1 min).

These features are used for malware detection based on their behavior. The list and detailed description of extracted features is shown in Table 4.

Following the organization of the features, a sample group of data points is chosen for each class. These data points are then utilized to train and test models of machine learning via 10-fold cross-validation.

Table 3: Feature categories.

Categories	Features
Host-MAC&IP	Packet count, mean and variance
Channel Packet	Packet count, mean, variance, magnitude, radius, covariance, and correlation
Network Jitter	Packet count, mean and variance of packet jitter in channel
Socket	Packet count, mean, variance, magnitude, radius, covariance, and correlation

Table 4: Features description.

Features	Description
Packet count	The packet count is comprised of two identifying numbers, the first of which represents the total number of information packets that a single piece of data was broken up into, and the second of which specifies the location of the specific packet within the larger information set.
Mean	Finding the mean, also known as the average, of a set of numbers requires first adding up all of the numbers in the set and then dividing that total by the total number of values in the set.
Variance	A data set's variance can be thought of as a measurement of dispersion because it takes into consideration the spread of all of the data points in the collection. In addition to the standard deviation, which is defined as the "square root of the variance," it is the measure of dispersion that is utilized the majority of the time.
Magnitude	The term "magnitude" most often refers to either the quantity or the distance. It is possible to correlate the amplitude of the movement with the size of the object and the speed at which it is traveling when it is in motion. Its magnitude can be thought of as either the size of the object or the amount.
Radius	A statistical measure derived from network traffic data and used as one of the features for characterizing communication patterns during feature extraction.
Covariance	A measure of the degree to which two variables are related to each other, or the degree to which two variables vary with each other, is referred to as covariance.
Correlation	The concept of correlation is used to explain the relationship between two or more variables. Those variables may be considered to be aspects of the input data that have been utilized in the process of forecasting the target variable. The statistical method known as correlation examines the relationship between two variables to establish how one variable moves or changes with respect to the other variable.

We applied correlation-based filtering to remove highly collinear features with a threshold >0.85 . We then used mutual information (MI) and recursive feature elimination (RFE) to quantify each feature's contribution to the deep autoencoder's reconstruction-error sensitivity. Features with consistently low MI scores and low RFE rankings across all five time windows were selected and carefully removed. This filtering process ensured that the final feature set reduces the noise and that there is no overfitting.

3.4 Normalization

It is essential to place all of the values on a standardized scale before applying any machine algorithms or classifiers. The attributes of the variables in the dataset are going to be normalized so that they are measured on a scale that is consistent throughout. The max normalization method is utilized by the preprocessing module in order to bring together all of the essential features. After the data has been normalized, it will be divided into a training dataset consisting of 80% and a testing dataset consisting of 20% before being submitted to the proposed deep learning model.

As a result of the utilization of min–max feature scaling, each client k is able to locally compute $x^{\min} \in \mathbb{R}^{100}$ and $x^{\max} \in \mathbb{R}^{100}$, and the server is able to locally compute the global minimum and maximum x as $x^{\min} \in \mathbb{R}^{100}$ and $x^{\max} \in \mathbb{R}^{100}$, the element-wise minimum and maximum, respectively, of those values. This procedure is described in detail in Algorithm 1. The disadvantage of this is that it requires every client to disclose their precise value of minimum and maximum for each of the 100 features. Any information exchanged in a federated setting must be carefully evaluated for potential privacy leakage. The shared values are global feature-level aggregates, not per-sample data. Min and max values do not encode individual records, labels, or behavioral patterns. These statistics are non-invertible, and to the best of our knowledge, it is mathematically impossible to reconstruct raw traffic traces, payloads, or user-specific activity from given data. Note that a few other federated learning algorithms, such as FedAvg and FedProx, etc., do share model parameters, which contain far more sensitive information than this simple approach.

Algorithm 1: Normalization: Client k uses $[K]$ as a collection of clients, and D_k^{Train} as a set of data points for training; max and min represent the element-wise maximum and minimum. They always produce a value in $\mathbb{R}^{100}$ since they are applied with vectors in $\mathbb{R}^{100}$ every time

```

1: Input: Client  $k$ 
2: Output: min and max values
3: Server executes:
4: Start Loop 1 (Parallel Client MinMax)
5: for each client  $k \in [K]$  in parallel do
6:    $x_k^{\min}, x_k^{\max} \leftarrow ClientMinMax(k)$ 
7: end for
8: End Loop 1
9:  $x^{\min} \leftarrow \min_{k \in [K]} \{x_k^{\min}\}$ 
10:  $x^{\max} \leftarrow \max_{k \in [K]} \{x_k^{\max}\}$ 
11: Start Loop 2 (Parallel Store MinMax)
12: for each client  $k \in [K]$  in parallel do
13:    $ClientStoreMinMax(k, x^{\min}, x^{\max})$ 
14: end for
15: End Loop 2

16:  $ClientMinMax(k)$ : // Run on client  $k$ 
17:  $x_k^{\min} \leftarrow \min \{D_k^{Train}\}$ 
18:  $x_k^{\max} \leftarrow \max \{D_k^{Train}\}$ 
19: return  $x_k^{\min}, x_k^{\max}$  to server

20:  $ClientStoreMinMax(k, x^{\min}, x^{\max})$ : // Run on client  $k$ 
21: Store  $x_k^{\min}$  // Client  $k$  now has access to  $x_k^{\min}$ 
22: Store  $x_k^{\max}$  // Client  $k$  now has access to  $x_k^{\max}$ 

```

3.5 Autoencoder

A deep learning model called the autoencoder has been successfully applied to learning unlabeled data. In feedforward neural networks, autoencoders are a specific subclass in which the input and output of the network are identical. The output is then recreated using this representation as a starting point after the input is compressed into a code with fewer dimensions. The latent-space representation, which is another name for the code, is a compressed or condensed summary of the input. The three components that make up an autoencoder are the encoder, the code, and the decoder. Before the encoder creates the code, the input is first compressed, and the decoder is in charge of reassembling the input using only the code.

Our proposed DAE consists of a symmetric encoder/decoder structure. The encoder includes three fully connected layers with 512, 256, and 128 neurons, respectively, and the bottleneck layer contains 64 neurons. The decoder mirrors the encoder with layers of 128, 256, and 512 neurons, followed by a final reconstruction layer with linear activation to match the input. We selected this architecture to balance computational efficiency for deployment in edge-cloud environments.

The proposed framework adopts a three-tier Hierarchical Federated Learning (HFL) architecture designed for edge-cloud malware detection. *Tier 1—Client Level:* The model training occurs in the federated cloud, whereas, inference is performed locally on IoT edge devices using the trained DAE model. *Tier 2—Edge Aggregation Level:* Multiple clients are grouped under an edge aggregator, which performs an aggregation using average weights. The aggregated model is then forwarded to the cloud coordinator. *Tier 3—Cloud Coordination Level:* The cloud server performs a second-level aggregation across all edge servers where the updated global model is redistributed to edge servers. Afterwards, the global model is propagated to the respective clients.

3.6 Robustness to Benign Traffic Noise

Robustness to benign traffic noise: IoT devices generate highly variable and bursty traffic. 1) Our multi-window statistical feature design and DAE latent-space compression help filter out short-term fluctuations, jitter and noise; 2) Robustness to distribution shifts (non-IID data): Edge devices exhibit heterogeneous traffic patterns due to the nature of IoT devices, network conditions and user behavior; 3) Robustness to unseen or zero-day malware: Because the DAE is trained only on benign data, it detects anomalies based on reconstruction error rather than signatures or labels. This enables detection of previously unseen malware families, early-stage botnet activity, and absence of variants in training data. Note that the adversarial robustness (e.g., against gradient-based attacks) is outside the scope of this work, but we take this as a highly potential future research direction. The proposed model has several characteristics of edge-cloud IoT environments, such as lightweight IoT device inference, which demonstrates that the DAE is suitable for edge-level deployment. The Cloud-assisted federated training is computationally expensive and occurs in the cloud. The edge devices perform lightweight local updates, and communication overhead is minimized via model compression and periodic retraining. To validate the robustness of the proposed federated DAE framework, we conducted two additional experiments: (i) *Noise Injection:* Gaussian noise ($\alpha = 0.05-0.2$) was added to 20% of the samples. The proposed model maintained an average accuracy drop of only 2.8%, compared to 7.4% for the centralized baseline. (ii) *Unseen Malware Families:* When tested on previously unseen malware families, the proposed model achieved 11.6% lower MAD and 6.2% higher F1-score than the non-federated DAE. These results demonstrate that the federated learning setup and the DAE's latent representation contribute to improved robustness against distribution shifts and noisy inputs.

3.7 Handling Potentially Contaminated Training Data

A known challenge in unsupervised anomaly detection and federated learning is the presence of malicious samples in the benign training set. The contamination analysis and training enhancements are explained in subsequent sections.

3.7.1 Empirical Contamination Analysis

We injected controlled contamination levels, i.e., 1%, 3%, 5% into the benign training data and evaluated the impact on reconstruction error distributions. Results show that the DAE remains stable up to 3% contamination, with less than 2% drop in AUC. At 5% contamination, the model begins to partially reconstruct malicious patterns, reducing anomaly separability.

3.7.2 Robust Training Enhancements

To mitigate contamination, two mechanisms have been added: i) Median Absolute Deviation (MAD), which is more robust to outliers introduced by contaminated samples and ii) Federated contamination dilution. This is because FL aggregates updates from many IoT devices, and malicious contamination at a single client has limited influence on the global model.

3.8 Method for Locally Detecting Anomalies

As it was indicated in the aforementioned research, there are wide, distinct varieties of neural networks from which one might select in order to do anomaly detection. During the process of this research, it is suggested to use the deep autoencoder method for an anomaly detection application. This model was presented both as the non-FL model and as the local model for FL. This is possible due to the fact that the deep autoencoder is able to learn the features as well as the representation of the observations through the usage of multiple layers and neurons.

Beginning with the pre-processing of the data, an overview of the entire process of anomaly detection using the deep auto-encoder model is presented in [Fig. 2](#). This model is applied locally. Once the datasets of various sorts of attacks and benign are gathered and labeled as traffic data, the dataset is then separated into two different kinds: benign data and anomalous data. After the 80%–20% split, which defines the overall training and testing datasets, the benign portion of the training data was subsequently partitioned into three equal parts for model training and validation. These parts are referred to as benign tr, benign train, and benign test.

Our rationale behind selecting an unsupervised deep autoencoder and how the proposed framework manages noise in benign IoT traffic is that in real IoT deployments, malware samples are rare and highly imbalanced. New botnet variants frequently appear, making supervised models prone to poor generalization when trained on outdated labels. Similar to other approaches, the proposed deep autoencoder learns the normal behavior of benign traffic by observing any deviation from its learnt representation and producing a high reconstruction error. The proposed approach is capable of detecting previously unseen malware, early-stage botnet activity, and zero-day anomalies. Moreover, the proposed approach allows all types of IoT devices to participate without requiring labels, improving model robustness and coverage. For noise handling, the feature selection is aggregated using temporal windows, which reduces the impact of short-term fluctuations and random spikes in benign traffic. Furthermore, the proposed approach compresses the input into a low-dimensional latent representation, which removes random noise and jitter. Because the global model aggregates updates from multiple heterogeneous IoT devices, any type of device-specific noise is averaged out. The obtained values are then encoded and decoded as they go from the input layer to the hidden layers and then to the output layers, as also used in other studies [\[18,25\]](#).

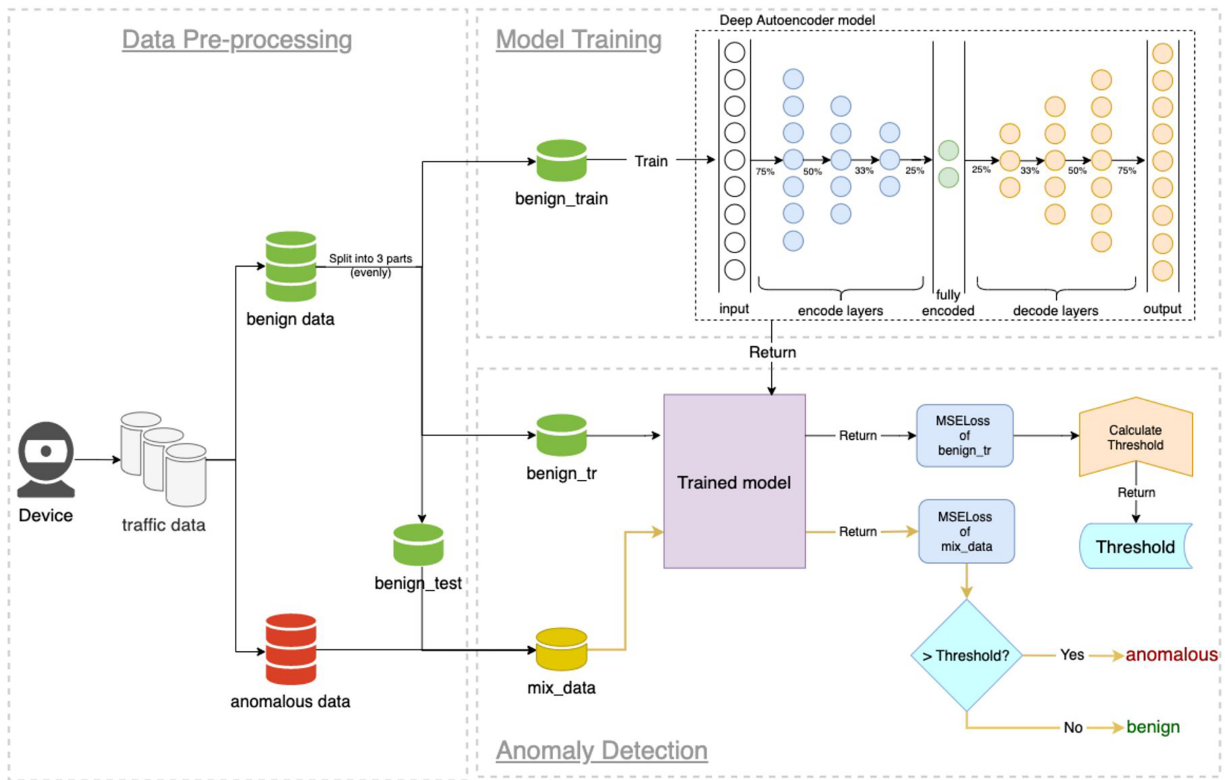


Figure 2: Malware detection with deep auto-encoder model.

Throughout the process of training, the hidden layers pick up on the most common patterns within the benign data and ignore the “noises.” Mean Squared Error Loss is the name of the loss function that was applied to this DAE model by us (MSELoss). After the model has been trained, the benign tr dataset is explicitly used as input into the trained model, with the trained model outputting the MSELoss for computing the threshold as the reconstruction error for identifying anomalies. It is critical to calculate the reconstruction error by making use of the loss values obtained from both the output and the input. If the loss value is low, for example, one can conclude that the model correctly identifies the instance as falling inside the parameters of the known distribution.

In this technique, benign test and anomalous data are eventually merged together under the name mix data. This occurs after the model has been trained and the reconstruction error has been determined. In order to do an evaluation, the mixed data values will be substituted into the training model. After receiving the loss values from mixed data, they will be compared with the threshold values after those values have been returned. The input is considered to be anomalous data if the value is greater than the threshold values. In any other case, the data are considered to be safe to use. More details on this can be seen in [17].

The parameters and hyperparameters of each trained model are adjusted in such a way that, when the model is applied to traffic it has not seen before or untrained traffic, the TPR (which means identifying attacks once they have occurred) is maximized while the false positive rate is minimized (FPR, wrongly marking benign data as malicious). The two distinct datasets are being used, both of which only contain benign data, in order to train and optimize the model. The model learns patterns of normal activity from those datasets.

3.9 Model Training

3.9.1 Detection Threshold

Following the training of the model and the extraction of its mean squared error (MSE), as indicated by [13,18], the equation for the threshold is the addition of the sample's mean MSE value and the sample's standard deviation MSE value when applied to the benign dataset. If the value of the instance is greater than the threshold value, then the data is seen as being anomalous; otherwise, the data is regarded as being benign. The threshold (tr) is employed in Eq. (1) [33,34] given to determine normal and abnormal data.

$$tr = \overline{MSE}_{benign_tr} + k \cdot std(\overline{MSE}_{benign_tr}) \quad (1)$$

Eq. (1) assumes that benign reconstruction errors are concentrated around MSE_{benign_tr} and anomalies lie in the upper tail of the distribution. Eq. (1) provides a simple, data-driven way to separate normal and abnormal behavior without requiring labeled attack data at the IoT device. The constant k is selected to balance true positive and false positive rates. The value of k can range from 1.0 to 3.0. The chosen value of k will determine a balanced operating point with a reasonable tradeoff between true positive and false positive rates. k can also be tuned to tolerate false alarms vs. missed detection. Note that the selected threshold corresponds to the “knee” region of the ROC curve, where marginal gains in false positive reduction would cause disproportionate losses in true positive rates. Our selected range follows standard statistical practice for deviation-based anomaly detection, where thresholds within 1–3 standard deviations capture the majority of benign reconstruction error distributions while allowing controlled sensitivity to anomalies. Furthermore, we conducted a sensitivity analysis using $k \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$. The model exhibited stable performance trends, with variations in F1 score remaining within a narrow band. This indicates that the reconstruction error distribution is well separated for the MedBioT dataset, reducing the risk of threshold-induced instability. The proposed framework remains robust across a reasonable range of k values, indicating that the method is not overly sensitive to small threshold changes.

3.9.2 A Method for Malware Detection with FL

For our experiments, the FL method was included in the deep auto-encoder model as a means of discovering malware in data. According to the review of the relevant research, FL might be able to assist in training the deep learning model on a variety of devices in order to process data while maintaining data confidentiality. One of the benefits of utilizing FL is that it does not call for the storing or trading of any personal data, which eliminates a potential threat to users' right to privacy.

Additionally, the weights of the DAE model on each device are transmitted to the global server, where they are processed by an aggregation function such as FedAvgM or FedAvg. These weights are forwarded to the global server from each device. The training procedure would then proceed with each deep auto-encoder model of each device being given the aggregated weights to move forward.

Quantifying the degree of data heterogeneity across IoT devices is essential for demonstrating the non-IID nature of the dataset and for motivating the retraining strategy. We have computed the distributional differences between the 83 IoT devices using two widely accepted metrics, i.e., i) Kullback-Leibler (KL) Divergence; and ii) Earth Mover's Distance (EMD). KL-divergence was calculated between each client's benign-traffic feature distribution and the global aggregated distribution. Results show strong non-IID characteristics. EMD was computed for key statistical feature groups (e.g., jitter, packet counts, socket activity). Several clients' EMD values indicate significant shifts in feature distributions due to device type, usage patterns, and network conditions. Note that client IoT devices with high KL/EMD values contribute updates that differ significantly from the global model, slowing convergence. The dataset is strongly non-IID,

not mildly heterogeneous. Retraining is necessary to prevent the global model from overfitting to dominant client distributions. Clients with extreme divergence values benefit the most from periodic model refresh, improving both local and global performance. The detailed view of client architecture during training and evaluation with steps is presented in Fig. 3. Note that the processes are iterative until the model is tested.

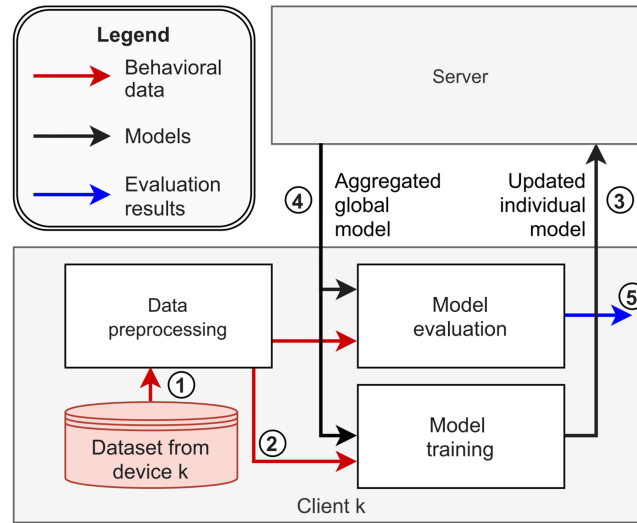


Figure 3: Detailed view of the client architecture during training and evaluation. Steps 1, 2, 3 and 4 are meant to be repeated several times before the model is evaluated in step 5.

3.9.3 Client-Selection Federated Learning

The datasets used in this research are non-identical Independent Distribution datasets and heterogeneity datasets (non-IID). On the other hand, in the FL model, the setup of the whole thing is quite sensitive to the distribution of the data classes and features of the client. Two methods, i.e., client selection and retraining procedure in the FL with the DAE model, are used with the intention of enhancing the model's overall performance. During the process of determining which clients to work with, the model selects partial training devices at random to use in the process of training the model. For example, five different types of electronic equipment are used to train the model as clients.

3.9.4 Federated Retraining Process

Additionally, the retraining process is optionally used in conjunction with the client selection procedure to further enhance the FL model. It also seeks to improve the model's applicability in real-world settings by fixing the problem of using non-IID datasets. In order to retrain, 1000 random examples will be taken from the benign train dataset. Before aggregating weights on the global server, the client's model will be retrained, so that the combined weights from the communication round and the retrained round can be used. Before the next training round begins, the local models on every client will be upgraded with the global weights. As a result, the global model is put to the test as a malware detection tool once it has completed its training and update process. The selection of 1000 benign samples was based on empirical sensitivity analysis, where we evaluated retraining with 200, 500, 1000, 2000 samples. We observed diminishing returns beyond 1000 samples, while smaller sample sizes led to unstable convergence. Thus, 1000 samples provided the best trade-off between computational cost and performance stability. Fig. 4 shows the structure of the proposed malware detection model using federated learning and a deep auto-encoder.

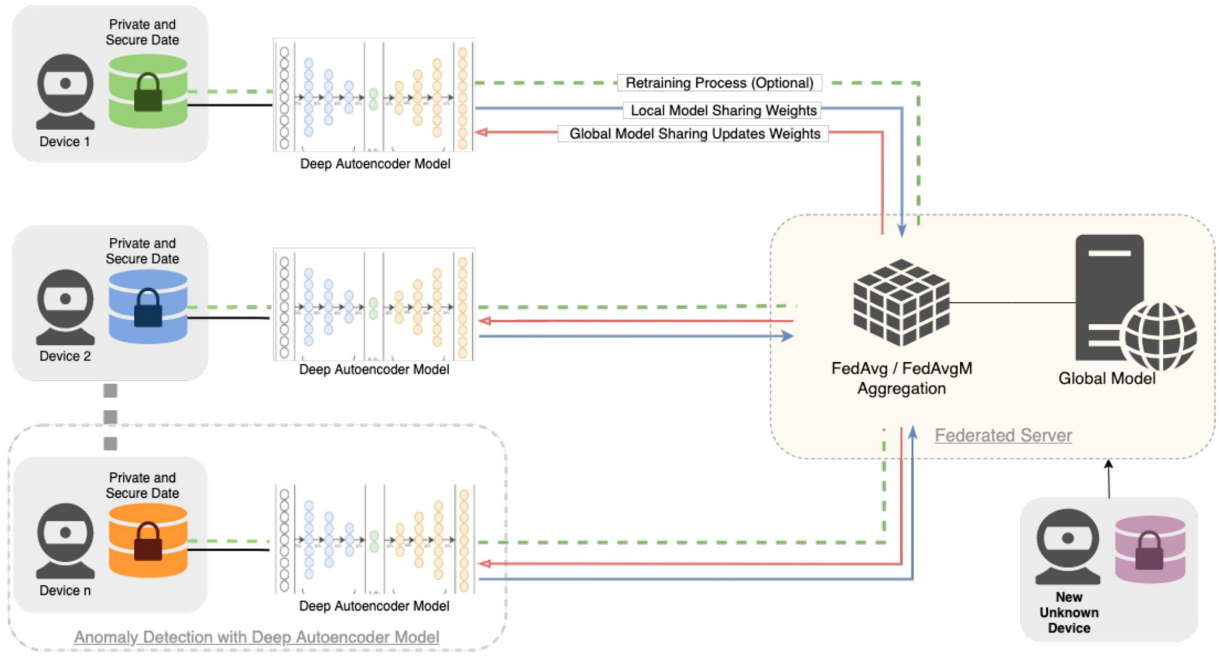


Figure 4: Proposed malware detection model using federated learning and deep auto-encoder.

3.9.5 FL Model Aggregation: Federated Averaging

Federated optimization uses stochastic gradient descent (SGD) and has the following properties:

- Implemented naively to the federated optimization issue, where a single batch gradient computation is done per round of communication
- Computationally efficient
- Requires many training rounds to create good models (also with batches normalization)

As indicated, FL initially selects the C-fraction of clients on every round and calculates the gradient of the loss across all the data held by these clients.

FedSGD is often implemented with C equal to 1 and a constant learning rate. This is an example of a common implementation. Each customer k will compute the following Eq. (2) [1].

$$g_k = \nabla F_k(w_t) \quad (2)$$

The central server compiles all of these gradients and applies the update after determining the average gradient based on its own local data at the current model w_t in Eq. (3) [1].

$$w_{t+1} \leftarrow w_t - \eta \sum_k \frac{n_k}{n} g_k \quad (3)$$

Given that in Eq. (3), $\sum_k \frac{n_k}{n} g_k = \nabla F_k(w_t)$, an update that is comparable to this is denoted as $\forall k, w_{t+1} \leftarrow w_{t+1}^k - \eta g_k$ in Eq. (4) [1],

$$w_{t+1} \leftarrow w_t - \eta \sum_k \frac{n_k}{n} w_{t+1}^k \quad (4)$$

Following the completion of one step of gradient descent on the current model by each client locally using its own local data, the server then performs a weighted average on the models that have been produced. More computation can be added to each client by iterating the local update [Eq. \(5\)](#) [1]:

$$w^k \leftarrow w^k - \eta F_k(w^k) \quad (5)$$

Calculating federated averaging, also known as FedAvg, involves adding computation to each client in advance of the averaging step. This is accomplished by repeatedly iterating the local update in [Eq. \(6\)](#) [1].

$$w^k \leftarrow w^k - \eta \nabla F_k(w^k) \quad (6)$$

The quantity of computation is determined by three important parameters, which are as follows:

- C, the percentage of clients that carry out computation during each round.
- E, the number of training passes that each client carries out over its own local dataset during each round.
- B, the local minibatch size that is utilized for client updates.

From a statistical point of view, it has been demonstrated that the FedAvg algorithm produces different results empirically in situations in which the data is not dispersed across devices in the same fashion.

In Algorithm 2 [1], the pseudocode for FedAvg is presented as follows:

Algorithm 2: Federated averaging: The K clients are indexed by k; B is local mini-batch size, E is the number of local epochs and η is the learning rate

```

1: Input:  $w_0$ 
2: Output: Updated weight  $w$ 
3: Server executes:
4: Initialize  $W_0$ 
5: Initialize  $v_0 = 0$ 
6: Start Loop 1 (Server Rounds)
7: for each round  $t = 1, 2, \dots$  do
8:    $m \leftarrow \max(C \cdot K, 1)$ 
9:    $S_t \leftarrow$  (random set of  $m$  clients)
10:  Start Loop 2 (Parallel Client Update)
11:  for each client  $k \in S_t$  in parallel do
12:     $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$ 
13:  end for
14:  End Loop 2
15:   $w_{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$ 
16:   $v_{t+1} \leftarrow \beta v_t + \Delta w_t$ 
17:   $w_{t+1} \leftarrow w_t + v_t + 1$ 
18: end for
19: End Loop 1

```

20: *ClientUpdate*(k, w):

// Run on client k

(Continued)

Algorithm 2 (continued)

```

21:  $\mathcal{B} \leftarrow$  (split  $P_k$  into batches of size  $B$ )
22: Start Loop 3 (Local Epochs)
23: for each local epoch  $i$  from 1 to  $E$  do
24:   Start Loop 4 (Mini-Batch Update)
25:   for each batch  $b \in \mathcal{B}$  do
26:      $w \leftarrow w - \eta \nabla l(w; b)$ 
27:   end for
28:   End Loop 4
29: end for
30: End Loop 3
31: return  $w$  to server

```

3.10 FL Model Aggregation: Federated Averaging with Momentum

The second approach on FL model aggregation is averaging with momentum. The momentum is used to improve the performance of the FedAvg algorithm. In light of this, the anticipation is that the FedAvgM algorithm will yield superior performance over the FedAvg aggregation technique during the course of this investigation.

The federated averaging with momentum (FedAvgM) algorithm is one that shares the majority of the same theory and algorithm that are presented in the FedAvg section; however, it adds momentum on top of the SGD at the server rather than on the client side. However, rather than only updating the weights of the updated model at the server by taking an average of the weights from all of the client models, as the first algorithm suggests, the momentum vector is included in order to update the weights of the updated model at the server. The procedure for updating the weights changes as shown in [Eq. \(7\)](#) [1].

$$w \leftarrow w - \Delta w \quad (7)$$

where w is the weighted update from the k th client to the following [Eq. \(8\)](#) [1].

$$w \leftarrow w + \Delta w \quad (8)$$

where v is the momentum vector derived as $v \leftarrow \beta v + \Delta w$. This change occurs because w is the weighted update from the k th client. It is possible for us to accelerate the convergence of the optimization techniques by speeding up the learning in the optimization direction when the momentum is used on top of the SGD at the server. The reason is that when momentum is 0.9, the convergence rates on both loss function and accuracy also steadily increase [16,21], it is decided to set the momentum values for this experiment to 0.9. The analysis of the hyperparameter indicates that low and high momentum values will respond differently. For low momentum values, the global model converges more slowly and exhibits higher variance across rounds, especially under non-IID client distributions. As previously discussed, the proposed model remains robust with small momentum values, indicating that the proposed approach is not overly sensitive to minor changes in this hyperparameter. The proposed global model is not intended as a universal solution for all IoT device categories. Instead, it serves as a baseline demonstrating that the DAE enhanced federated framework can generalize across heterogeneous clients within the MedBioT environment. In real deployments, hierarchical or device-clustered FL (e.g., per device type models) can be integrated, which aligns with emerging FL

architectures for large-scale IoT ecosystems. This decision was based on the fact that when momentum is 0.9, the convergence rate increases steadily.

4 Model Testing

The proposed model is able to detect complex malware functions from a benign dataset. For testing, all features are extracted. Some of these files are given to the trained model, which will give reconstruction error files for that device. These MSE scores are then mapped with their features; if MSE value of a function is less than the threshold, it will be considered benign; otherwise, if it is greater, then it will be detected as malware. Malware analysts get an immediate benefit from using an unsupervised deep autoencoder because it directs their attention toward relevant functions based on the reconstruction error, and it filters out functions that would have taken the analyst a significant amount of time to interpret, potentially even days.

4.1 ROC Curve: Receiver Operating Characteristic

ROC curves have become increasingly popular among researchers in recent years and are currently employed by a large number of them to gauge performance, which is pretty clear. In addition, ROC curves are currently being utilized in a wide variety of statistical procedures that combine a variety of hints, test results, and other related details. These methods have now been plotted and reviewed subjectively. When calculating performance, the ROC is essentially a plot in which the TPR is plotted on the y -axis and the FPR is displayed on the x -axis.

The TPR is calculated differently for each conceivable classification based on the circumstances in which the actual classification is positive and the frequency with which the classifier has predicted it as positive. The FPR takes into account the frequency with which an inaccurate positive prediction was made by the classifier while the actual classification is negative. The TPR and the FPR both have a range of 0 to 1. The area that is covered below the ROC curve is referred to as the Area Under the ROC Curve (AUC), and it is frequently used for measuring the effectiveness of weighting classifiers. The AUC values each reflect a different level, and [Table 5](#) organizes a variety of research endeavors according to those levels.

Table 5: AUC levels.

Definition	Excellent Prediction	Great Prediction	Good Prediction	Better Prediction	Poor Prediction	Random Prediction	Unreliable Prediction
Label	1.0	0.9	0.8	0.7	0.6	0.5	Less than 0.5

While the AUC provides a global view of classifier separability, operational IDS performance is governed primarily by threshold-specific metrics such as F1 score, precision, recall, and false negative rate. These metrics remain consistently high in our model, indicating strong practical reliability despite an AUC of 0.89. Despite the AUC of 0.89, the model achieves high recall and low false negative rates, which are critical for detecting stealthy malware families such as Torii. This indicates that the model reliably identifies malicious behavior even when reconstruction errors are close to benign traffic.

4.2 Scalability and Client Count Sensitivity Analysis

In this research, we conducted our experiments using 10 to 150 IoT devices and measured model accuracy, communication cost and computation time. The key findings are:

- Accuracy improves rapidly when IoT devices increase from 10 $\rightarrow$ 50, due to greater data diversity.
- Beyond 80 to 90 IoT devices, accuracy gains become less significant

- Communication overhead increases with the number of IoT devices
- Local computation time per IoT device remains stable, but global convergence time increases due to more aggregation rounds

Note that 83 clients lie near the inflection point where accuracy gains plateau. Among the evaluated configurations, the highest accuracy of 89% was achieved with 83 clients. This also demonstrates that 83 clients represent a practical balance between performance and overhead for resource-constrained IoT environments. IoT devices often operate under strict bandwidth and energy constraints. Excessive communication rounds can degrade battery life and increase latency. The configuration used in the proposed methodology minimizes overhead while maintaining strong detection performance.

5 Experimental Setup

The execution of the experiments was done on an Artificial Intelligence Labs workstation equipped with 128 GB of RAM, four Nvidia RTX 2080 Ti graphics cards, and the Ubuntu 20.04 operating system. We emphasize that only inference is performed on IoT devices; the training is done on the server, further reducing device burden on the constrained devices. The DAE's shallow architecture makes it inherently more efficient than CNN/RNN-based baselines. Similarly, inference is performed locally on the edge device, while training occurs in the federated cloud. This minimizes the device's computational load. The entire preliminary data processing was done in Python with various public libraries' help. PyTorch was the platform for deep learning that was utilized. PyCharm, an open-source library, was utilized in the development of the FL. The source code has been made available, and a URL and DOI have been created and provided, i.e., <http://dx.doi.org/10.13140/RG.2.2.30808.00007>.

5.1 Communication Overhead

The DAE model requires significantly less bandwidth than deeper CNN/LSTM/ResNet architectures. Communication cost grows linearly with the number of clients, but accuracy gains plateau after 80 clients. In the proposed approach, a three-layer model has been used with the Device layer, Edge (fog) layer, and Cloud layer. Furthermore, an adaptive thresholding strategy based on the following has been used in this study: Percentile-based dynamic thresholds; device-specific calibration windows; local reconstruction-error statistics; and optional edge-level smoothing to reduce false positives.

5.2 Dataset

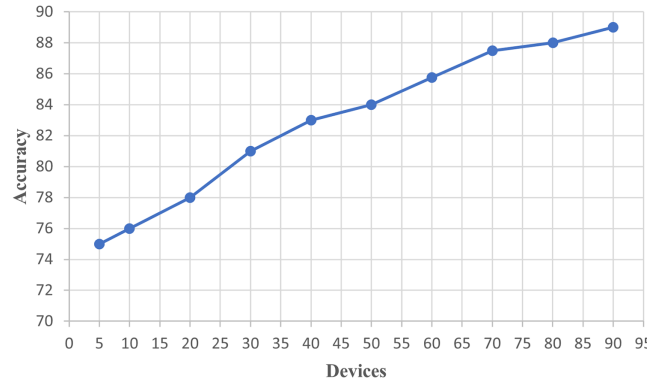
The datasets used in this research provide the potential to identify the danger early, before the incidence of an attack, avoiding such attacks and malware expansion. This study uses three of the most notorious types of botnet malware, one of which is an absolute first (i.e., Torii), as it has never been employed in any other research project using publicly available data. The other two are well-known examples of malware that can infect Internet of Things botnets; their source code is accessible to the public and has been implemented in various data sets (i.e., Mirai and BashLite).

The data sets that are currently available are small in size. The MedBIoT data set overcomes these restrictions by constructing a network consisting of both real and simulated devices in order to simulate a medium-sized environment (i.e., 83 devices) [19]. When compared to IoT networks with a smaller size, those with a bigger size may provide various insights and interactions. Training our proposed model on different numbers of clients has proven and shown that a greater number of clients shows better performance than a small number of clients, as given in Table 6.

Table 6: Performance of different number of clients.

Samples	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6
No. of Clients	5	10	20	40	60	83
Accuracy %	75	78	80	83	85	89

As the number of devices/clients increases, the accuracy of the model also increases, as shown in Fig. 5. These data have been validated through the use of machine learning models, which demonstrates that this data set is suitable for use as a reliable data source for malware detection in general and IDS testing and implementation in particular. The use of this data collection intends to overcome the lack of relevant data sources in Internet of Things network security as well as the limitations of the other data sets that are currently available [19].

**Figure 5:** Relation between accuracy and number of clients.

5.3 Performance Evaluation with Other DL Models

Experiments are run on three different machine learning models, DNN, CNN, and RNN, so that the performance of the proposed model, DAE, is analyzed and evaluated. The outcomes that were produced using the proposed framework were satisfactory for each situation.

To ensure a fair comparison with the proposed DAE-based framework, all baseline models (DNN, CNN, RNN) were converted into autoencoder-style architectures. The DNN is implemented as a fully connected encoder-decoder. For CNN, a 1D convolutional layer is applied to the feature vector. Lastly, the RNN is implemented using stacked LSTM layers to model temporal dependencies. All the baseline models were trained using the same steps, i.e., local training, model sharing, federated sharing, and evaluation consistency. It can be seen in Fig. 6 that our proposed model performs well across the board on all of the criteria for each and every situation. It is important to point out that in all of the cases, we were able to get a good level of accuracy, precision, recall, and F1 score by utilizing the provided framework for the various numbers of edge devices. The relation between the client devices and the accuracy of the model is given in Fig. 5. The accuracy is in terms of percentage, which increases as the number of nodes increases. As mentioned earlier, we have compared our proposed model with DNN, CNN, and RNN. We have compared the model's classification and detection performance with the communication rounds. The detailed view and results of every model are given below.

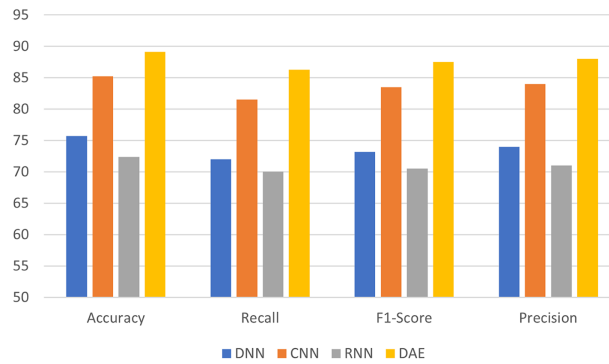


Figure 6: Performance comparison (%) between the proposed model and individual DL models.

5.3.1 Deep Neural Network

The experimental results indicate that the proposed federated deep autoencoder framework achieved better performance than the evaluated DNN model on the considered malware detection dataset. These findings suggest that the proposed approach is more effective under the experimental conditions examined in this study. Due to the complicated data models, training is incredibly expensive and can become costly to the users [26,37]. It can be observed that DNNs haven't shown promising results in detecting malware in unknown IoT devices compared with our proposed model, as shown in Fig. 7.

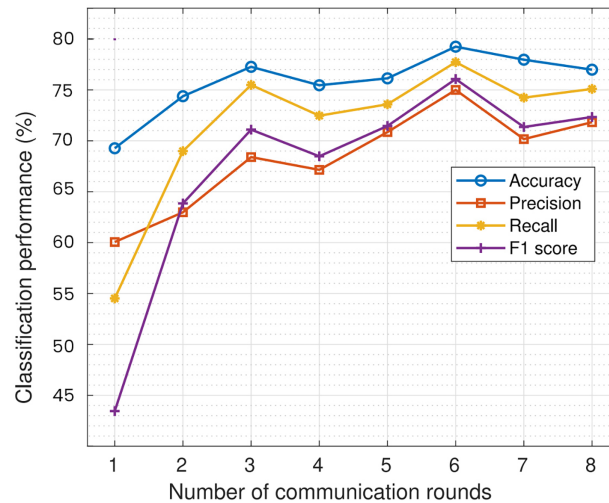


Figure 7: Classification performance of DNN.

5.3.2 Recurrent Neural Network

With reference to RNNs, vanishing and/or exploding gradient difficulties are frequently seen. They occur because long-term conditions are difficult to catch due to a multiplicative factor that might substantially decrease or increase in terms of the number of layers. Therefore, the model may train with null weights (i.e., no training) or exploding weights if the sequence ever grows too long.

Any unfolding RNN is trained across a number of time steps, and the error gradient is determined as the total of all gradient errors over timestamps. The algorithm is sometimes known as "backpropagation through time" because of this (BPTT). The multiplicative term dominates the gradient over time as a result

of the chain rule's use in calculating error gradients, and as a result, the gradient has the propensity to burst or disappear, as evident in other studies [29,33,36].

The gradient will disappear if the biggest eigenvalue is smaller than 1. The gradient explodes if the biggest eigenvalue is greater than 1. Due to the essential feature of RNNs, i.e., they are connected, the main issue is when it comes to parallelizing the training in RNNs or a simple stacking of training. This means that in order to compute over the present node, RNNs need to know the output of the preceding node. The fact that RNNs are recurrent is one of their main issues. In other words, training will take a long time. RNNs train at a much slower rate than feedforward networks overall. Second, RNNs are challenging to build because each node's state change function must be calibrated using both the current inputs and the prior outputs. Customizing RNN training can occasionally be challenging due to training complexity. Note that accuracy improvement is outweighed by the total computational cost, as also reported in other studies [12,28,29].

As was previously said, it might be challenging to train RNNs on excessively long sequences, particularly when utilizing ReLU or tanh activations. Another justification for the introduction of GRU-based networks is this. So, it is concluded that RNNs also haven't shown good results in detecting malware in unknown IoT devices compared with our proposed model. Resultantly, this is shown in Fig. 8.

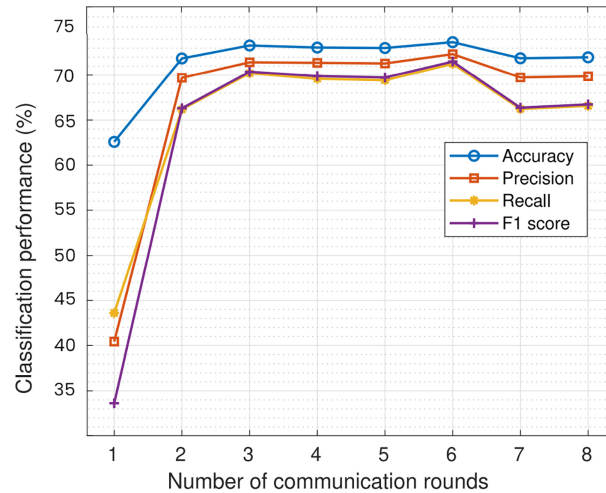


Figure 8: Classification performance of RNN.

5.3.3 Convolutional Neural Network

CNNs have a number of drawbacks, including the fact that they are ineffective at encoding the position and orientation of objects and the fact that they require a significant amount of training data before they can be used effectively. They have trouble classifying photos in which the locations are different from one another. The inherent locality constraints are the most significant drawback of the convolution technique. The capability of the convolutional algorithm to extract visual patterns that are distributed over multiple spatial positions is restricted, as also mentioned in [11,18,21,30]. Using CNN to train a model presents a number of problems, the most significant of which are overfitting, exploding gradients, and class imbalance. The effectiveness of the model may suffer as a result of these concerns. So, it is concluded that CNN also hasn't shown many promising results in detecting malware in unknown IoT devices compared with our proposed model. Resultantly, this is shown in Fig. 9.

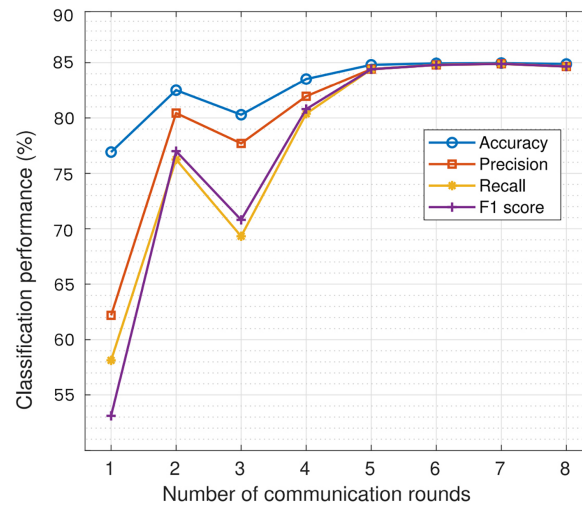


Figure 9: Classification performance of CNN.

5.3.4 Deep Autoencoder

Deep learning models can be created considerably more quickly and effectively by using autoencoders, which offer a handy approach to significantly reducing the noise of input data. They can be applied to remove complexity from datasets, find anomalies, and deal with unsupervised learning issues.

A particular kind of neural network called an autoencoder is capable of learning a compressed representation of raw data. Encoder and decoder sub-models make up an autoencoder. The main benefit of deep learning algorithms is that they attempt to incrementally learn high-level features from data. This does away with the necessity for hard-core feature extraction and domain knowledge.

In order to achieve effective results, autoencoders make unique use of a neural network's properties for training neural networks to learn typical behavior. The autoencoder learns to reconstruct normal data patterns effectively. When an outlier or anomalous data point appears, it is reconstructed less accurately, resulting in a higher reconstruction error that enables its detection. In order to design an anomaly detection module, it is essential to rebuild the original input from the encoded data. Our finding is that the DAE Model outperformed the aforementioned models because of the above-mentioned reasons. The detailed preview of DAE results is shown in [Fig. 10](#).

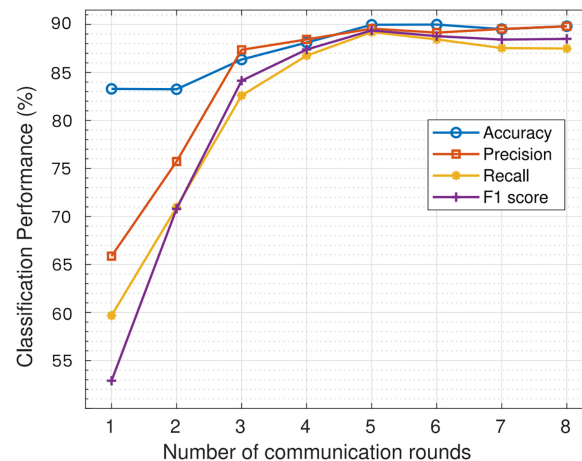


Figure 10: Classification performance of DAE.

5.4 Fairness Considerations

In the proposed research, we consider fairness as the ability of the global model to provide consistent performance across heterogeneous clients, regardless of differences in dataset size, malware family distribution, or feature imbalance. Note that unfairness can result in worse detection performance. We evaluate fairness by analyzing the clients' performance after each global round. We consider the standard deviation of client-level F1-scores and MAD values across all experiments. The proposed hierarchical FL framework maintains low inter-client variance ($\sigma_{F1} = 0.021$ and $\sigma_{MAD} = 0.034$), indicating that no client class is dealt unfairly. We believe that the proposed model provides an equal level of performance across all types of clients and improves the reliability and credibility of the system.

6 Case Study and Test Scenarios

We evaluate the proposed framework in a realistic organizational IoT setting using MedBioT traffic and three malware families (Mirai, BashLite, and Torii). Four test scenarios are considered to assess generalization from known to unknown devices and from small to larger client populations. In Test 1, baseline evaluation on a small set of known devices shows strong discrimination with high TPR, low FPR, and AUC around 0.93. Test 2 demonstrates that when scaled to all available clients, the baseline model degrades (AUC around 0.78), indicating limited scalability without federated adaptation. In Test 3, the proposed federated DAE provides more stable anomaly detection than the baseline, with improved separation between benign and malicious behavior. Lastly, in Test 4, the proposed model maintains better robustness and lower false-positive behavior than non-federated alternatives, supporting practical IoT deployment.

We provide a breakdown and analyze failure cases to help interpret the results more concisely. The following observations summarize the per-malware behavior and key failure modes.

- Mirai shows the highest detection rate due to its aggressive scanning and high-volume traffic, which produces distinct reconstruction errors.
- BashLite exhibits moderate detection performance because its behavior deviates from benign traffic but is less noisy than Mirai.
- Torii is the most challenging to detect in early stages because of its stealthy, low-frequency communication patterns, resulting in lower anomaly contrast.

We also analyzed representative failure cases of these algorithms:

- Some Torii samples produce reconstruction errors close to benign traffic due to their minimal packet-rate deviation, which aligns with Torii's stealth characteristics.
- A small number of false positives occurred in devices with bursty or irregular traffic, such as smart cameras, where benign noise overlaps with early botnet behavior.
- When the global model is trained predominantly on Mirai-like anomalies, it may underperform on Torii-like patterns.

7 Conclusions and Future Directions

In this research, a unique technique is proposed to detect malware in new, unknown, and known IoT devices. A DAE is applied to a medium-sized dataset, and a comparison is performed with similar approaches such as DNN, RNN, and CNN. The results show that this model has achieved high accuracy, and the value of ROC/AUC is 0.89. Its AUC value indicates having low FPR and high TPR, which are much better than the baseline methods such as CNN and RNN, etc.

Running deep autoencoder models on resource-constrained IoT devices presents several challenges due to their limited computational power and memory capacity. To address this, the following are some considerations: reducing the number of layers and neurons, utilizing lightweight algorithms, and optimizations can help reduce the computational complexity. Moreover, reducing the number of parameters in the code and using sparse matrices to represent the model can lower the memory usage.

Federated Learning (FL) offers promising solutions for detecting malware in IoT environments, but it also faces several limitations and challenges in real-world deployment, such as IoT devices often generating data that is not identically and independently distributed (non-IID). This can lead to biased local models. Moreover, frequent communication between devices and the central server can introduce significant delays and is highly costly. Lastly, managing and coordinating updates from a large number of IoT devices can be challenging, as many IoT devices have limited computational resources that can result in being ineffective in detecting sophisticated malware.

The proposed DAE-FL approach is suitable for IoT botnet detection and malware analysis because i) it does not require labeled attack data; ii) it handles non-IID client distributions effectively; iii) it is lightweight for edge devices; iv) it generalizes better to unseen malware.

In the future, we intend to extend this work in three high-impact directions:

- **Larger and more diverse clients:** We aim to extend experiments to larger IoT populations and stronger heterogeneity to stress-test model generalization.
- **Robust aggregation and defense:** One direction can be to evaluate advanced aggregation/robustness strategies (e.g., FedMA-like matching and poisoning-resistant aggregation) to further reduce false positives.
- **Compression with recent FL-based IDS:** One of the future extensions of this work can be compression of most recent FL-based IDS such as FedKD-IDS, FFL-IDS, VAE and FedPROX.

Acknowledgement: None.

Funding Statement: The author would like to thank the Deanship of Scientific Research (DSR), King Faisal University, Kingdom of Saudi Arabia for the Grant No. KFU262642.

Author Contributions: Conceptualization, methodology, validation, resource acquisition, project supervision, and funding arrangements by Munam Ali Shah. Writing, visualization, software and results collection by Shazil Gul. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are openly available in [Taltech] at <https://cs.taltech.ee/research/data/medbiot/>.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DAE	Deep Auto Encoder
FL	Federated Learning
IDS	Intrusion Detection System
MSE	Mean Square Error
TPR	True Positive Rate
IID	Identically and Independently Distributed

References

1. Rey V, Sánchez Sánchez PM, Huertas Celdrán A, Bovet G. Federated learning for malware detection in IoT devices. *Comput Netw*. 2022;204:108693. doi:10.1016/j.comnet.2021.108693.
2. Neto HNC, Dusparic I, Mattos DMF, Fernandes NC. FedSA: accelerating intrusion detection in collaborative environments with federated simulated annealing. *arXiv:2205.11519*. 2022.
3. Popoola SI, Ande R, Adebisi B, Gui G, Hammoudeh M, Jogunola O. Federated deep learning for zero-day botnet attack detection in IoT-edge devices. *IEEE Internet Things J*. 2022;9(5):3930–44. doi:10.1109/JIOT.2021.3100755.
4. Meidan Y, Avraham D, Libhaber H, Shabtai A. CADeSH: collaborative anomaly detection for smart homes. *IEEE Internet Things J*. 2023;10(10):8514–32. doi:10.1109/JIOT.2022.3194813.
5. Aslan ÖA, Samet R. A comprehensive review on malware detection approaches. *IEEE Access*. 2020;8:6249–71. doi:10.1109/ACCESS.2019.2963724.
6. Celdrán AH, Sánchez PMS, Castillo MA, Bovet G, Pérez GM, Stiller B. Intelligent and behavioral-based detection of malware in IoT spectrum sensors. *Int J Inf Secur*. 2023;22(3):541–61. doi:10.1007/s10207-022-00602-w.
7. Yadav K, Gupta BB, Hsu CH, Chui KT. Unsupervised federated learning based IoT intrusion detection. In: *Proceedings of the 2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*; 2021 Oct 12–15; Kyoto, Japan. p. 298–301.
8. Jiang C, Yin K, Xia C, Huang W. FedHGCDroid: an adaptive multi-dimensional federated learning for privacy-preserving Android malware classification. *Entropy*. 2022;24(7):919. doi:10.3390/e24070919.
9. Al-Hawawreh M, Sitnikova E, Aboutorab N. Asynchronous peer-to-peer federated capability-based targeted ransomware detection model for industrial IoT. *IEEE Access*. 2021;9:148738–55. doi:10.1109/ACCESS.2021.3124634.
10. Khoa TV, Hoang DT, Trung NL, Nguyen CT, Quynh TTT, Nguyen DN, et al. Deep transfer learning: a novel collaborative learning model for cyberattack detection systems in IoT networks. *IEEE Internet Things J*. 2023;10(10):8578–89. doi:10.1109/jiot.2022.3202029.
11. Mammen PM. Federated learning: opportunities and challenges. *arXiv:2101.05428*. 2021.
12. Kairouz P, McMahan HB. Advances and open problems in federated learning. *Found Trends[®] Mach Learn*. 2021;14(1–2):1–210. doi:10.1561/22000000083.
13. Charyyev B, Gunes MH. Detecting anomalous IoT traffic flow with locality sensitive hashes. In: *Proceedings of the GLOBECOM 2020-2020 IEEE Global Communications Conference*; 2020 Dec 7–11; Taipei, Taiwan. p. 1–6.
14. Shejwalkar V, Houmansadr A. Manipulating the byzantine: optimizing model poisoning attacks and defenses for federated learning. In: *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium 2021*; 2021 Feb 21–25; Virtual Event.
15. Kamal H, Mashaly M. Shared autoencoder-based unified intrusion detection across heterogeneous datasets for binary and multi-class classification using a hybrid CNN-DNN model. *Mach Learn Knowl Extr*. 2026;8(2):53. doi:10.3390/make8020053.
16. Dharshiniya S, Daniel Madan Raja S. Enhancing IoT intrusion detection using autoencoder and temporal fusion transformer. *SN Comput Sci*. 2026;7(3):264. doi:10.1007/s42979-026-04867-z.
17. Mansour Y, Mohri M, Ro J, Suresh AT. Three approaches for personalization with applications to federated learning. *arXiv:2002.10619*. 2020.
18. Huang L, Yin Y, Fu Z, Zhang S, Deng H, Liu D. LoAdaBoost: loss-based AdaBoost federated machine learning with reduced computational complexity on IID and non-IID intensive care data. *PLoS One*. 2020;15(4):e0230706. doi:10.1371/journal.pone.0230706.
19. Guerra-Manzanares A, Medina-Galindo J, Bahsi H, Nömm S. MedBioT: generation of an IoT botnet dataset in a medium-sized IoT network. In: *Proceedings of the 6th International Conference on Information Systems Security and Privacy*; 2020 Feb 25–27; Valletta, Malta. p. 207–18. doi:10.5220/0009187802070218.
20. Guerra-Manzanares A, Medina-Galindo J, Bahsi H, Nömm S. Using MedBioT dataset to build effective machine learning-based IoT botnet detection systems. In: *Information systems security and privacy*. Cham, Switzerland: Springer International Publishing; 2022. p. 222–43. doi:10.1007/978-3-030-94900-6_11.
21. Dhanya L, Chitra R. A novel autoencoder based feature independent GA optimised XGBoost classifier for IoMT malware detection. *Expert Syst Appl*. 2024;237:121618. doi:10.1016/j.eswa.2023.121618.

22. Zhang H, Ye J, Huang W, Liu X, Gu J. Survey of federated learning in intrusion detection. *J Parallel Distrib Comput*. 2025;195:104976. doi:10.1016/j.jpdc.2024.104976.
23. Muhammad S, Hosseinzadeh A, Chen FF. A two-stage hybrid federated learning framework for privacy-preserving IoT anomaly detection and classification. *IoT*. 2025;6(3):48. doi:10.3390/iot6030048.
24. Mughole Kalimumbalo D, Badejo JA, Okokpujie KO, Adetiba E, Mbasa Joaquim M, Vingi Patrick N, et al. SecFedMDM-1: a federated learning-based malware detection model for interconnected cloud infrastructures. *IEEE Access*. 2025;13:101246–61. doi:10.1109/access.2025.3577706.
25. Çıplak Z, Yıldız K, Altınkaya Ş. FEDetect: a federated learning-based malware detection and classification using deep neural network algorithms. *Arab J Sci Eng*. 2025;50(19):16107–34. doi:10.1007/s13369-025-10043-x.
26. Berman DS, Buczak AL, Chavis JS, Corbett CL. A survey of deep learning methods for cyber security. *Information*. 2019;10(4):122. doi:10.3390/info10040122.
27. Taher F, Abdel-Salam M, Elhoseny M, El-Hasnony IM. Reliable machine learning model for IIoT botnet detection. *IEEE Access*. 2023;11:49319–36. doi:10.1109/access.2023.3253432.
28. Baruah S, Borah DJ, Deka V. Reviewing various feature selection techniques in machine learning-based botnet detection. *Concurr Comput*. 2024;36(12):e8076. doi:10.1002/cpe.8076.
29. Kaur J, Prabha M, Samiun M, Hasan SN, Hasan R, Esa H, et al. Comparative analysis of transformer and LSTM architectures for cybersecurity threat detection using machine learning. *EAI Endorsed Trans AI Robot*. 2025;4:1–12. doi:10.4108/airo.9759.
30. Liu D, Zheng X, Wang P, Chuan J, Lv Y, Zhou B, et al. Deep learning-based intrusion detection: a CNN-LSTM-transformer approach for enhanced network security. In: *Proceedings of the 10th International Conference on Cyber Security and Information Engineering*; 2025 Jul 23–25; Xining, China. p. 318–25. doi:10.1145/3759179.3760451.
31. Almalki SS. A deep learning-based framework for real-time detection of cybersecurity threats in IoT environments. *Int J Adv Comput Sci Appl*. 2025;16(3):430–9. doi:10.14569/ijacsa.2025.0160343.
32. Govindaram A, Jegatheesan A. FLBC-IDS: a federated learning and blockchain-based intrusion detection system for secure IoT environments. *Multimed Tools Appl*. 2025;84(17):17229–51. doi:10.1007/s11042-024-19777-6.
33. Olanrewaju-George B, Pranggono B. Federated learning-based intrusion detection system for the Internet of Things using unsupervised and supervised deep learning models. *Cyber Secur Appl*. 2025;3:100068. doi:10.1016/j.csa.2024.100068.
34. Huang J, Chen Z, Liu SZ, Zhang H, Long HX. Improved intrusion detection based on hybrid deep learning models and federated learning. *Sensors*. 2024;24(12):4002. doi:10.3390/s24124002.
35. Chaurasia N, Ram M, Verma P, Mehta N, Bharot N. A federated learning approach to network intrusion detection using residual networks in industrial IoT networks. *J Supercomput*. 2024;80(13):18325–46. doi:10.1007/s11227-024-06153-2.
36. Anwar RW, Abrar M, Salam A, Ullah F. Federated learning with LSTM for intrusion detection in IoT-based wireless sensor networks: a multi-dataset analysis. *PeerJ Comput Sci*. 2025;11:e2751. doi:10.7717/peerj-cs.2751.
37. Albanbay N, Tursynbek Y, Graffi K, Uskenbayeva R, Kalpeyeva Z, Abilkaiyr Z, et al. Federated learning-based intrusion detection in IoT networks: performance evaluation and data scaling study. *J Sens Actuator Netw*. 2025;14(4):78. doi:10.3390/jsan14040078.
38. Karunamurthy A, Vijayan K, Kshirsagar PR, Tan KT. An optimal federated learning-based intrusion detection for IoT environment. *Sci Rep*. 2025;15:8696. doi:10.1038/s41598-025-93501-8.