A Novel Paradigm for IoT Security: ResNet-GRU Model Revolutionizes Botnet Attack Detection

Jyotsna A[1], Mary Anita E.A[2]
Dept. of Computer Science and Engineering, Christ (Deemed to be University), Bangalore, India[1]
Dept. of Computer Science and Engineering, RSET, Kochi, India[1]
Dept. of Computer Science and Engineering, Christ (Deemed to be University), Bangalore, India[2]

***Abstract*—The rapid proliferation of the Internet of Things (IoT) has engendered substantial security apprehensions, chiefly due to the emergence of botnet attacks. This research study delves into the realm of Intrusion Detection Systems (IDS) by leveraging the IoT23 dataset, with a specific emphasis on the intricate domain of IoT at the network's edge. The evolution of edge computing underscores the exigency for tailored security solutions. An array of statistical methodologies, encompassing ANOVA, Kruskal-Wallis, and Friedman tests, is systematically employed to illuminate the evolving trends across multiple facets of the study. Given the intricacies entailed in feature selection within edge environments, Chi-square analyses, Recursive Feature Elimination (RFE), and Lasso-based techniques are strategically harnessed to unearth meaningful feature subsets. A meticulous evaluation encompassing 19 classifiers, meticulously selected from both machine learning (ML) and deep learning (DL) paradigms, is rigorously conducted. Initial findings underscore the potential of the Gated Recurrent Unit (GRU) model, especially when coupled with intrinsic lasso-based feature selection. This promising outcome catalyzes the formulation of an ensemble approach that harnesses multiple LassoCV models, aimed at amplifying feature selection proficiency. Furthermore, an optimized ResNet-GRU model emerges from the fusion of the GRU and ResNet architectures, with the objective of augmenting classification performance. In response to mounting concerns regarding data privacy at the edge, a resilient federated learning ecosystem is meticulously crafted. The seamless integration of the optimized ResNet-GRU model into this framework facilitates the employment of FedAvg, a widely acclaimed federated learning methodology, to adeptly navigate the intricacies associated with data sharing challenges. A comprehensive performance evaluation is undertaken, wherein the ResNet-GRU model is benchmarked against FedAvg and a diverse array of other federated learning algorithms, including FedProx and Fed+. This extensive comparative analysis encompasses a spectrum of performance metrics and processing time benchmarks, shedding comprehensive light on the capabilities of the model.**

Keywords*—Internet of things; federated learning; Gated Recurrent Neural Networks; Long Short Term Memory (LSTM)

I. INTRODUCTION

The Internet of Things (IoT) has transformed device connectivity, bringing benefits and challenges in anomaly detection. IoT anomalies can stem from various factors like environmental changes, cybersecurity breaches, or device failures [1]. Detecting and understanding these anomalies are vital for ensuring dependability, security, and performance. Anomalies can disrupt operations, compromise data security, or invade privacy, necessitating proactive identification. Specialized methods, including artificial intelligence, machine learning, and statistical analysis, are essential for anomaly detection. IoT devices are susceptible to cybersecurity threats like unauthorized access and data breaches, involving atypical network traffic, unusual access patterns, or suspicious user behavior. Detecting these anomalies is crucial for preventing security breaches. Additionally, individual IoT devices may exhibit unexpected behavior due to software, firmware, or hardware issues [2]. Promptly identifying and resolving these device anomalies is essential for maintaining device reliability.

Addressing IoT anomalies involves various techniques, including artificial intelligence, machine learning, statistical analysis, and anomaly detection algorithms. These methods aim to establish normal behavior, detect deviations, and trigger appropriate responses. Machine learning and deep learning are particularly effective due to their ability to analyze vast datasets and identify patterns, enhancing IoT system safety and reliability [3]. In this article, we explore how machine learning and deep learning are applied in IoT anomaly detection. Supervised learning is used with labeled datasets, training models on historical data to recognize normal behavior patterns and identify anomalies in real-time data. Algorithms like decision trees, support vector machines (SVM), random forests, and gradient boosting are employed [4]. In cases of limited or unlabeled data, unsupervised learning becomes essential. It identifies anomalies by analyzing data structures and trends, utilizing techniques such as clustering to group data and detect anomalies as outliers [5].

Autoencoders are a type of neural network, excel in IoT anomaly detection by reducing input data dimensionality and detecting anomalies through reconstruction errors. Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, are ideal for handling sequential IoT data. Generative Adversarial Networks (GANs) are suitable for anomaly detection, as they can simulate complex IoT data distributions [6]. Federated learning is a decentralized approach for IoT anomaly detection that preserves data privacy [7]. IoT devices locally train models, transmitting only model updates, reducing the need for data transfer to a centralized server. This approach is beneficial for scenarios with limited bandwidth or intermittent connectivity, enhances model stability, and accommodates device-specific constraints [8]. However, it introduces communication overhead and security concerns [9].

The proposed work analyzes the IoT23 dataset, focusing on botnet attacks, using statistical tests and three feature selection approaches (filter, wrapper, embedded). Fifteen classifiers, including machine learning and deep learning models, are evaluated, with the best-performing one being the GRU model with embedded lasso-based feature selection. An ensemble of LassoCV models and ResNet architecture further improves feature selection and classifier performance. To address privacy concerns, a federated learning environment is established, and the optimized ResNet-GRU model is deployed and compared with existing federated learning algorithms, considering various metrics and processing time.

- The study focuses on Intrusion Detection Systems (IDS) within the context of the Internet of Things (IoT) at the network's edge, addressing heightened security concerns due to botnet attacks.
- Edge computing's evolution necessitates customized security solutions, and this research endeavors to provide them.
- Various statistical methodologies, including ANOVA, Kruskal-Wallis, and Friedman tests, are employed to reveal evolving trends in IoT security at the network's edge.
- Innovative feature selection techniques such as Chi-square analyses, Recursive Feature Elimination (RFE), and Lasso-based methods are applied to navigate the complexities of feature selection in edge environments.
- The study rigorously evaluates 19 classifiers from both machine learning (ML) and deep learning (DL) domains, with a particular focus on the Gated Recurrent Unit (GRU) model, which shows promise in conjunction with lasso-based feature selection.
- A novel ensemble approach harnessing multiple LassoCV models is developed to enhance feature selection efficiency.
- The introduction of an optimized ResNet-GRU model, combining GRU and ResNet architectures, aims to improve classification performance.
- To address data privacy concerns at the edge, a resilient federated learning ecosystem is created, integrating the optimized ResNet-GRU model and employing FedAvg, a widely acclaimed federated learning methodology.
- Comprehensive performance evaluation includes benchmarking against FedAvg and various other federated learning algorithms, such as FedProx and Fed+, covering a wide range of performance metrics and processing time benchmarks.

II. Related Works

This section presents the related works carried out by several research scholars in the area of anomaly detection using deep learning and machine learning with the aid of federated learning.

Brett Weinger et al. [10] employed Federated Learning (FL) for collaborative mobile and IoT projects but faced technological challenges. Distributing ML training across devices reduced prediction accuracy compared to centralized learning. Limited data access led to issues like constrained local ML models and class imbalances due to diverse event contributions. They addressed these challenges with data augmentation, resulting in a significant 22.9% performance improvement in IoT anomaly detection across three datasets.

Zhuotao Lian et al. [11] enhanced IoT anomaly detection while addressing security concerns. They proposed a distributed federated learning approach using neural networks, as traditional methods proved inaccurate. This technique improved detection accuracy while safeguarding locally stored data through decentralized learning, eliminating central failure points and raw data flow. Simulations using the IoT23 dataset validated its effectiveness, showcasing the promise of distributed learning for secure and accurate IoT anomaly detection, nearly matching centralized federated learning in performance.

Truong Thu Huong et al. [12] developed the FedeX architecture for efficient distributed anomaly detection in IoT-based Industrial Control Systems (ICSs) for Smart Manufacturing. FedeX outperformed 14 other methods on various detection measures, offering rapid learning, lightweight deployment, and interpretability. It allows real-time edge deployment with 7.5 minutes of training and 14% memory use, enhancing Smart Manufacturing practices. Explainable AI (XAI) to improve model interpretability, helping experts make confident decisions.

Subir Halder et al. [13] developed Hawk, an anomaly detection system for LoRa-enabled IIoT networks to address cybersecurity challenges. Hawk uses unique Carrier Frequency Offset (CFO) measurements to create device "fingerprints" and detect suspicious behavior. Employing federated learning, Hawk outperformed other systems by over 8% in detection accuracy and demonstrated high resilience against cyberattacks, reducing storage overhead by 40%. It's an effective solution for securing LoRa-enabled IIoT networks against novel threats.

Xabier Sáez-de-Cámara et al. [14] addressed IoT cybersecurity challenges in their study. They proposed a system using unsupervised models for network intrusion detection in large and diverse IoT and IIoT deployments. To overcome issues like network overhead and heterogeneity, they leveraged Federated Learning (FL) for cooperative training. Their architecture, tested on a simulated network with 100 nodes and subjected to real-world attacks, demonstrated efficient and robust intrusion detection for large-scale IoT and IIoT environments.

Huong Thu Truong et al. [15] developed a scalable anomaly detection system for continuously operating Industrial Control Systems (ICS) in smart manufacturing and IIoT. Their system combines Federated Learning, Autoencoder, and Transformer, with a Fourier mixing sublayer for improved performance. It offers rapid training within minutes, is lightweight with low computational and memory requirements, and minimizes communication costs. Compared to existing

methods, it reduces training time by 50% to 1200 seconds, adapting to changing conditions and mitigating false positives in ICS data patterns, ensuring robust anomaly detection for smart manufacturing.

Jiamin Fan et al. [16] developed Score-VAE, a novel root cause analysis method for IoT anomaly detection systems. It addresses the challenge of distinguishing false positives from malicious attacks. Score-VAE combines the training and testing schemes of the VAE network within the federated learning (FL) architecture, resulting in improved generalization, learning, collaboration, and privacy protection. It effectively identifies the sources of anomaly detection alarms in real-world IoT data, outperforming standard approaches and enhancing the accuracy of root cause analysis in IoT anomaly detection.

Ali Raza et al. [17] introduced AnoFed, a novel federated framework for anomaly detection in digital healthcare, particularly in ECG analysis. To overcome limitations in threshold selection and privacy concerns in centralized machine learning, they combined transformer-based AE and VAE with Support Vector Data Description (SVDD). AnoFed enhances privacy, improves interpretability, and facilitates adaptive anomaly detection. Experiments in ECG anomaly detection demonstrated its effective performance with low computational costs. AnoFed's efficiency and privacy-preserving capabilities make it a valuable solution for digital healthcare applications, suitable for deployment on low-powered edge devices.

J. Jithish et al. [18] conducted a technical study in the past, focusing on anomaly detection in the smart grid using Federated Learning (FL). Anomaly detection is crucial for identifying energy theft, cyberattacks, and excessive power usage. In this approach, smart metres locally train machine learning models without sending data to a centralised server. Smart metres download a global model from the server for on-device training, and after local training, upload model parameters to fine-tune the global model, protected by the SSL/TLS protocol. Experiments on industry-standard datasets demonstrated that FL models matched the accuracy of centralised ML models while preserving individual privacy. The research showcased the efficiency of FL-based models in terms of memory, CPU utilization, bandwidth, and power consumption at edge devices, making them suitable for deployment in resource-constrained settings like smart metres in the smart grid.

III. MATERIALS AND METHODS

This study focuses on IoT network botnet attack detection using feature selection and classification techniques. It employs three feature selection methods (filter, wrapper, and embedded) to reduce dataset dimensionality. Out of 15 classifiers tested, the GRU model with embedded lasso-based feature selection emerges as the top performer [19]. To enhance detection capabilities, an ensemble approach is applied, incorporating 10 LassoCV models. Additionally, optimization with the ResNet architecture is employed to improve detection accuracy and convergence speed by addressing the vanishing gradient problem. This comprehensive approach aims to provide more effective and efficient botnet attack detection in IoT networks using the IoT23 dataset. The detailed architecture is presented in Fig. 1.

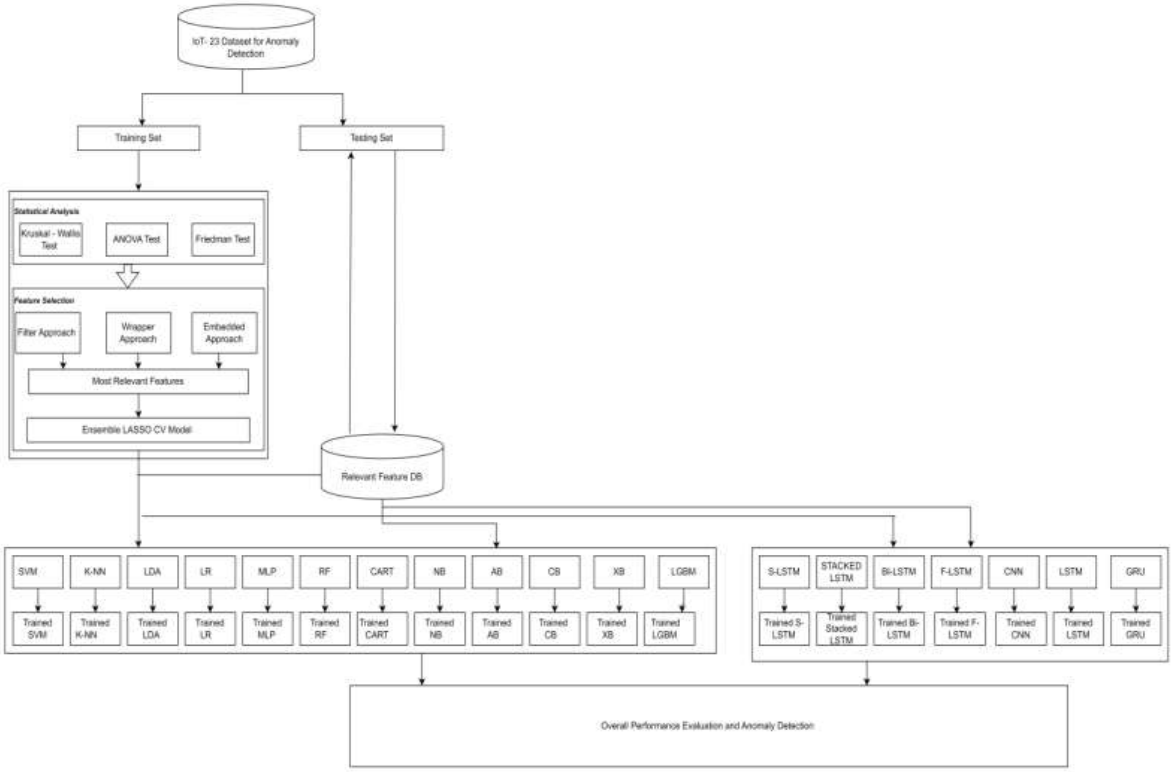

Fig. 1. The overall system architecture.

A. System Architecture

The study's system architecture focuses on analyzing the IoT23 dataset for botnet attacks, beginning with data collection and cleaning. Features are extracted and the dataset is divided into training and testing sets, followed by statistical tests to uncover data patterns. Feature selection is performed using filter, wrapper, and embedded methods, with the GRU model with embedded Lasso-based selection standing out as the top classifier. An ensemble approach with 10 LassoCV models enhances feature selection's reliability. Further improvements are achieved by integrating the ResNet architecture into the GRU model, addressing deep learning challenges. To ensure privacy, federated learning is introduced, allowing decentralized model training without sharing raw data [20]. The optimized ResNet-GRU model's performance is compared with existing federated learning algorithms like FedProx, FedAvg, and Fed+ in terms of metrics and processing time. This approach offers a comprehensive solution for botnet attack detection in IoT networks while addressing privacy concerns [21] [22].

B. Dataset Description

The IoT-23 dataset, released in January 2020, comprises network activity data from IoT devices. It includes benign IoT device traffic and malware-infected IoT device captures. This dataset, created by the Stratosphere Laboratory at CTU University, aims to support machine learning research in IoT malware detection. It consists of twenty-three scenarios, featuring malware execution on Raspberry Pi devices and real IoT device captures like Philips HUE smart LED lamps and Amazon Echo. This dataset offers a valuable resource for training algorithms to detect IoT malware and enhance IoT security.

C. Data Preprocessing

Data preprocessing is the process of cleaning and formatting data so that it can be used for analysis. This can involve removing outliers, imputing missing values, and transforming the data into a format that is suitable for the analysis method being used. The Kruskal-Wallis test is a non-parametric test that can be used to compare the distributions of two or more groups. It is a non-parametric test because it does not make any assumptions about the distribution of the data. This makes it a versatile test that can be used with a variety of data types [23].

To perform a Kruskal-Wallis test on the IoT-23 dataset, you would first need to preprocess the data. This would involve removing any outliers, imputing any missing values, and transforming the data into a format that is suitable for the Kruskal-Wallis test. Once the data has been preprocessed, you can perform the Kruskal-Wallis test. The Kruskal-Wallis test will output a p-value. If the p-value is less than a significance level (typically 0.05), then you can conclude that there is a significant difference between the distributions of the two or more groups. The proposed work focuses mainly on the attacks namely Torri, Okiru , Mirai and also normal labels - benign as class labels out of 13 labels present in the IoT-23 dataset.

D. Kruskal – Wallis Test

The Kruskal-Wallis test is utilized to compare the medians of three or more distinct groups. Unlike parametric tests such as ANOVA, which rely on assumptions of normal distribution and equal variances, this non-parametric test is employed when these assumptions are not met. The Kruskal-Wallis test involves the following aspects:

Hypotheses: The null hypothesis *(H0)* assumes that all group medians are equal, while the alternative hypothesis *(HA)* posits that at least one median differs from the others. The hypothesis is presented in Eq. (1).

$$H = \frac{12}{n(n+1)} \sum \frac{R_j^2}{n_j} - 3(n+1) \quad (1)$$

where, H is the Kruskal-Wallis test statistic, n is the total number of observations across all groups, R_j is the sum of ranks for group j, n_j is the number of observations in group j. The test statistic H follows a chi-square distribution with degrees of freedom equal to the number of groups minus 1(df = k-1), where k is the number of groups being compared. The significance of the test can be determined by comparing the obtained test statistic with the critical value from the chi-square distribution with the appropriate degrees of freedom [24].

H statistic	345.78
Degrees of Freedom	4
p-value	< 0.001

Null Hypothesis: There are no significant differences among the groups.

Alternate Hypothesis: There are significant differences among the groups.

Conclusion: The p-value (< 0.001) is smaller than the significance level (usually 0.05), so we reject the null hypothesis. This indicates significant differences among the groups.

IV. PROPOSED METHODOLOGY

In this study, we comprehensively investigate cyber-attack detection in IoT environments using the IoT23 dataset. We compare the performance of different classifiers, including LSTM, GRU, CNN, and traditional classifiers, to identify the most effective one for detecting malicious activities in IoT networks.

The CNN module described in the article consists of four stages, each comprising multiple convolution blocks with different sizes of convolution kernels. The convolutional layer performs operations on input images, such as feature extraction, feature mapping, weight sharing, and local connection. The convolution operation reduces image size and computational cost for subsequent operations.

The formula for the convolution operation is given as:

$$v(i,j) = (X * w)(i,j) + b = \sum n(k=1)\,(Xk * wk)(i,j) + b \quad (2)$$

Here, 'n' represents the number of input matrices, Xk denotes the k^{th} input matrix, and ωk represents the k^{th} sub

convolution kernel matrix of the convolution kernel. The activation layer applies a non-linear mapping, specifically the Rectified Linear Unit (ReLU) activation function, to the output of the convolution layer. The ReLU function is defined as:

$$ReLU(x) = \begin{cases} x, if\ x > 0 \\ 0, if\ x \leq 0 \end{cases} \quad (3)$$

A Simple Recurrent Unit (SRU) serves as the foundation for Recurrent Neural Networks (RNNs), but RNNs can be challenging to train due to gradient issues. Variations like GRU and LSTM were introduced to address these problems. LSTM, for example, includes memory cells and gates to capture temporal sequences and improve recognition accuracy. However, LSTM's complexity can be an issue, so a simplified gating unit was introduced to streamline calculations. LSTM and GRU differ in how they update the next hidden state and handle content exposure. LSTM uses summation for updates, while GRU considers the time needed to save information in memory. Recent comparisons have shown that GRU often performs slightly better than LSTM in various machine learning applications.

In the structure of a Bi-GRU, both reset and update gates are present. These gates allow GRU to pass information across multiple time windows for better classification or prediction. Specifically, weights and data are stored in memory to be used with a given state for updating future values. In the update gate, GRU computes zt at a given time t to solve the vanishing gradient problem using the following formula:

$$zt = \sigma(Wz[ht-1, xt] + bz). \quad (4)$$

whereas, in the reset gate, GRU calculates zt at a given time t to illustrate how much past information to forget. The gate executes the following calculation:

$$rt = \sigma(Wr[ht-1, xt] + br). \quad (5)$$

The current storage content stage is calculated according to the following formula:

$$h\tilde{} = tanh(W[rtht-1, xt]) \quad (6)$$

Finally, ht is calculated in the final memory of the current time step to store the current unit information for calculating the output vector ot, as follows:

$$ht = (1-zt)\,ht-1 + zth\tilde{}t \quad (7)$$

For many sequence modeling tasks, accessing future and past contexts is beneficial. However, the standard GRU network processes the sequence in chronological order, disregarding the future context. Bi-GRU networks extend the unidirectional GRU network by introducing a second layer in which the hidden connections flow in reverse chronological order.

A. Long Short Term Memory (LSTM)

The LSTM network is a specialized type of deep neural network that excels at capturing long-term dependencies in time-series data. It achieves this by incorporating memory cells and gating operations. The memory cells are updated through gating operations that determine what information to remember and what to forget in the temporal sequence. This makes LSTM highly suitable for modelling temporal dynamics effectively.

There are three types of gating operations in LSTM: the input gate (it), the output gate (ot), and the forget gate (ft). The expressions that form the foundation of LSTM are as follows:

Input Gate:

$$i_t = \sigma_t(W_i[h_t - 1, x_1] + b_i) \quad (8)$$

Forget Gate:

$$f_t = \sigma_f(W_f[h_t - 1, x_1] + b_f) \quad (9)$$

Cell State Update

$$c_t = f_t . c_{t-1} + i_t . \sigma_c(W_c[h_{t-1}, x_t] + b_c) \quad (10)$$

Output Gate

$$o_t = \sigma_0(W_0[h_{t-1}, x_t] + b_0) \quad (11)$$

Hidden State Update

$$h_t = o_t . \sigma_h(c_t) \quad (12)$$

where:

- x_t is the input data sequence.
- i_t, f_t and o_t represent the input, forget and output gates respectively.
- c_t and h_t correspond to the cell and hidden states respectively.
- b_i, b_f, b_c *and* b_o are biases related to the input gate, forget gate, cell state and output gate respectively.
- W_i, W_f, W_c *and* W_o are the weight matrices of the input gate, forget gate, cell state and output gate respectively.
- $\sigma_t, \sigma_f, \sigma_c, \sigma_o$ *and* σ_h are the activation functions of the input gate, forget gate, cell state, output gate and hidden state respectively.

In this study, we employed three distinct variations of LSTM, namely the single cell, stacked, and bidirectional LSTM models. These different LSTM variants were chosen to examine and compare their benchmark scores. By incorporating these diverse LSTM architectures, we aimed to explore the performance differences and identify the most suitable model for the given task.

1) Single Cell LSTM: Single-cell LSTM (Long Short-Term Memory) is a variation of the traditional LSTM neural network architecture that is designed to process individual data points or sequences one at a time. It is particularly useful in tasks where the input data has a temporal or sequential nature, such as natural language processing, speech recognition, and time series analysis. LSTMs are a type of recurrent neural network (RNN) that is capable of capturing long-term dependencies and addressing the vanishing gradient problem, which is a common issue in training RNNs. The single-cell LSTM architecture extends the basic LSTM by

removing the concept of cell state, resulting in a simpler and more efficient model.

2) Stacked LSTM: Stacked LSTM (Long Short-Term Memory) is an extension of the traditional LSTM architecture that involves stacking multiple LSTM layers on top of each other. This allows the model to learn more complex and abstract representations of sequential data by capturing hierarchical dependencies. Each LSTM layer in a stacked LSTM consists of multiple LSTM cells, and the output of one layer serves as the input to the next layer. This stacking of LSTM layers enables the network to learn higher-level features and representations by building upon the representations learned in the preceding layers.

3) Bidirectional LSTM: Bidirectional LSTM (Long Short-Term Memory) is an extension of the traditional LSTM architecture that processes the input sequence in both forward and backward\\directions. This allows the model to capture dependencies from both past and future context, enabling better understanding of the input sequence. In a bidirectional LSTM, the input sequence is processed by two separate LSTM layers: one layer processes the sequence in the forward direction, and the other layer processes it in the backward direction. The outputs of these two layers are then combined to produce the final output.

4) Forward LSTM: A Forward LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) architecture used in machine learning and deep learning for sequential data processing tasks. LSTM networks are particularly effective in handling sequences of data because they can capture long-range dependencies and mitigate the vanishing gradient problem, which is common in traditional RNNs. In a Forward LSTM, the input sequence is processed from the beginning to the end, one-time step at a time, without considering future time steps during the computation at each step. By processing the input sequence in both directions, the bidirectional LSTM can capture both past and future context, which can be beneficial in tasks such as natural language processing, sentiment analysis, and speech recognition. It allows the model to make more informed predictions by considering the complete context of the sequence.

B. Federated Learning

Federated learning is a machine learning approach that allows training of deep learning models across a network of decentralized devices while preserving data privacy. It enables the aggregation of local model updates from multiple devices without the need to transfer raw data to a central server. In this proposed work, a detailed introduction to popular federated learning algorithms: FedAvg, FedProx, Fed+ (FedPlus) has been discussed.

1) FedAvg (Federated Averaging): FedAvg is a fundamental federated learning algorithm that utilizes the idea of model averaging. It follows a simple iterative process where each device trains a local model using its local data and shares only the model's updates with the central server. The central server aggregates the updates from all devices by taking the average and updates the global model accordingly. The algorithm can be summarized as follows:

Initialization: Initialize the global model parameters, θ.

Iteration: Randomly select a subset of devices for participation.

For each selected device i:

Step 1: Send the current global model parameters to device i.

Step 2: Device i trains the local model on its local data, optimizing for a specific loss function, and obtains updated local model parameters, θ_i.

Step 3: Device i calculates the update difference: $\Delta\theta_i = \theta_i$ - θ.

Step 4: Device i sends the update difference back to the central server.

The central server aggregates the update differences from all devices and calculates the average update:

$$\Delta\theta_{avg} = \frac{1}{N} * \sum \Delta\theta_i \quad (13)$$

The central server updates the global model:

$$\theta = \theta + \Delta\theta_{avg} \quad (14)$$

2) FedProx (Federated Proximal): FedProx extends FedAvg by introducing a proximal term to regularize the updates sent by each device. This regularization term helps in controlling the magnitude of the updates, preventing devices from deviating too far from the global model. The objective function of FedProx can be defined as:

$$L(\theta) = \left(\frac{1}{N}\right) * \sum\left(1(\theta, D_i)\right) + {}^{\lambda}/_{2} * \|\theta - \theta_{old}\|^2 \quad (15)$$

where, $1(\theta, D_i)$ represents the loss function on device i's local data D_i, λ is a hyperparameter controlling the proximal term, and θ_{old} is the model parameters from the previous round. FedProx can be seen as minimizing a weighted sum.

3) Fed+ (FedPlus): Fed+ is an extension of FedAvg that addresses the issue of device heterogeneity by assigning different weights to each device during aggregation. The weights reflect the devices' relative contributions to the global model. This approach helps to mitigate the impact of devices with varying computation capabilities or imbalanced datasets. The Fed+ algorithm can be summarized as follows:

Initialization: Initialize the global model parameters, θ.

Assign an initial weight for each device i, w_i.

Iteration:

Step 1: Randomly select a subset of devices for participation.

Step 2: For each selected device i:

Step 3: Send the current global model parameters to device i.

Step 4: Device i trains the local model on its local data, optimizing for a specific loss function, and obtains updated local model parameters, θ_i.

Step 5: Device i calculates the update difference:

$$\Delta\theta_i = \theta_i - \theta. \quad (16)$$

Step 6: Device i sends the update difference back to the central server.

The central server aggregates the update differences from all devices by weighted averaging:

$$\Delta\theta_{avg} = \frac{\sum(w_i * \Delta\theta_i)}{\sum w_i} \quad (17)$$

The central server updates the global model:

$$\theta = \theta + \Delta\theta_{avg}. \quad (18)$$

Adjust the weights of devices based on their contribution to the global model.

These algorithms represent different approaches to addressing the challenges of federated learning, such as heterogeneity, privacy preservation, and data distribution variations. The equations and explanations provided offer a high-level understanding of the algorithms, but specific implementation details may vary depending on the framework or research work.

C. ResNet-GRU Combined Architecture

Federated Learning is a groundbreaking approach that enables model training across distributed devices while protecting data privacy. The ResNet-GRU model, combining Residual Networks (ResNets) and Gated Recurrent Units (GRUs), excels at capturing spatial and temporal patterns, especially in scenarios with data distributed across multiple devices. ResNets, introduced in 2016, revolutionized deep learning, addressing the vanishing gradient problem in deep neural networks. They use "residual blocks" with skip connections, allowing gradients to flow more effectively through many layers. A residual block comprises stacked convolutional layers, batch normalization, and activation functions, with shortcut connections enabling input to bypass some convolutional layers. This design allows for training extremely deep networks more efficiently.

The central idea behind residual learning is to model the residual function $\Delta F(x) = F(x) - x$, where $F(x)$ represents the mapping learned by the convolutional layers, and x denotes the input to the residual block. Instead of attempting to learn the complete mapping $F(x)$, the network focuses on learning the difference or residual $\Delta F(x)$, which is subsequently added back to the input x to obtain the output of the block. This element-wise addition operation facilitates the preservation of prior knowledge, simplifying the learning process for deep networks.

Federated Learning is revolutionizing machine learning by training models on distributed devices while protecting data privacy. In anomaly detection, the ResNet-GRU model stands out for capturing both spatial and temporal features. It combines Residual Networks (ResNets) and Gated Recurrent Units (GRUs), making it ideal for federated learning scenarios with diverse data. Federated Learning decentralizes data to protect user privacy and data security. The ResNet-GRU model excels by blending ResNets' spatial prowess and GRUs' sequential data modeling capabilities. Residual blocks, key to the ResNet-GRU model, enable training deep networks by allowing information to bypass certain layers. Federated training is collaborative, with clients training local ResNet-GRU models on their data subsets. Models iteratively update, and global models are aggregated while preserving data privacy. Evaluation metrics like precision, recall, and ROC-AUC assess the model's anomaly detection performance. The ResNet-GRU model, adept at capturing spatial and temporal nuances, is a powerful tool for real-time anomaly detection while respecting federated learning principles.

Algorithm: ResNet-GRU Model in Federated Learning for Anomaly Detection

Input: Federated dataset (split into multiple clients), hyperparameters

Output: Trained ResNet-GRU model for anomaly detection

Step 1: Initialize the ResNet-GRU model

- Define the architecture of the ResNet-GRU model.

- Set hyperparameters such as the number of ResNet blocks, GRU units, learning rate, batch size, and number of training rounds.

Step 2: Federated Learning Setup

- Split the federated dataset into multiple clients or devices, each having its own subset of data.

- Distribute the ResNet-GRU model to all clients.

Step 3: Federated Training

- For each training round (t = 1 to number of training rounds):

- For each client i in the federated dataset:

- Load the ResNet-GRU model parameters from the global model.

- Train the ResNet-GRU model on client i using its local subset of data:

- For each mini-batch in client i's data:

- Perform forward pass through the ResNet to extract spatial features.

- Convert the spatial features into temporal sequences (if needed).

- Pass the temporal sequences through the GRU to capture temporal patterns.

- Calculate the loss using an appropriate anomaly detection loss function.

- Perform backward pass and update the model's parameters using an optimization algorithm (e.g., stochastic gradient descent).

- After training, send the updated model parameters back to the server.

Step 4: Model Aggregation

- Aggregate the model parameters from all clients to create a global ResNet-GRU model:

- For each layer and parameter in the ResNet-GRU model:

- Calculate the weighted average of the parameters from all clients.

- Update the global model's parameters with the weighted averages.

Step 5: Evaluation

- After each training round, evaluate the global ResNet-GRU model on a separate test set (not used for training) to monitor its performance.

- Measure metrics such as precision, recall, F1-score, ROC-AUC, or mean average precision for anomaly detection.

Step 6: Repeat Training and Aggregation

- Repeat Steps 3 to 5 for the desired number of training rounds or until the global model achieves satisfactory performance.

Step 7: Deployment

- Once the global ResNet-GRU model achieves satisfactory performance, deploy it to the production environment for anomaly detection on new data.

V. RESULTS AND DISCUSSIONS

The following section discusses the results obtained from the various experiments done on the IoT-23 dataset.

TABLE. I PERFORMANCE EVALUATION WITHOUT FEATURE SELECTION

SL No	Classifiers	Accuracy	Precision	Recall	F1 score	Time Taken (Sec)
1	Support Vector Machine	0.62	0.62	0.64	0.63	289.325
2	K-Nearest Neighbor	0.66	0.66	0.67	0.66	256
3	Linear Discriminant Analysis	0.71	0.75	0.71	0.73	290
4	Logistic Regression	0.65	0.65	0.63	0.64	300.25
5	Multi-Layer Perceptron	0.7	0.7	0.69	0.69	300.24
6	Random Forest	0.64	0.64	0.63	0.63	483.987
7	Decision Tree	0.69	0.69	0.68	0.69	356.355
8	Naïve Bayes	0.63	0.62	0.63	0.63	478.9
9	AdaBoost	0.68	0.68	0.68	0.68	225.36
10	XGBoost	0.62	0.63	0.61	0.62	290.93
11	CatBoost	0.67	0.68	0.67	0.68	600.32
12	LightGBM	0.61	0.61	0.6	0.61	542.03
13	Convolutional Neural Network	0.66	0.656	0.65	0.65	320
14	Single Cell LSTM	0.61	0.702	0.7	0.69	430
15	Stacked LSTM	0.66	0.748	0.75	0.73	345
16	Bidirectional LSTM	0.61	0.794	0.8	0.77	389
17	Forward LSTM	0.66	0.84	0.85	0.81	225
18	Long Short Term Memory	0.6	0.6	0.61	0.6	245.36
19	Gated Recurrent Neural Network	0.75	0.75	0.74	0.75	158.96

Table I shows classifier performance without feature selection on the dataset, evaluated by accuracy, precision, recall, F1 score, and processing time. Results vary significantly: Support Vector Machine achieves 62% accuracy, K-Nearest Neighbor 66%, and Linear Discriminant Analysis 71%. Gated Recurrent Neural Network performs well with 75% accuracy, precision, recall, and F1 score. Processing time varies, from 158.96 seconds for the Gated Recurrent Neural Network to 600.32 seconds for CatBoost, indicating different computational demands. Visualization plots are in Fig. 2 and Fig. 3.

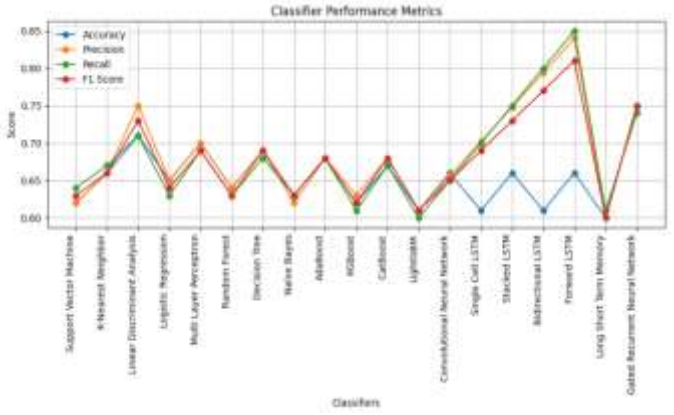

Fig. 2. Performance evaluation without feature selection.

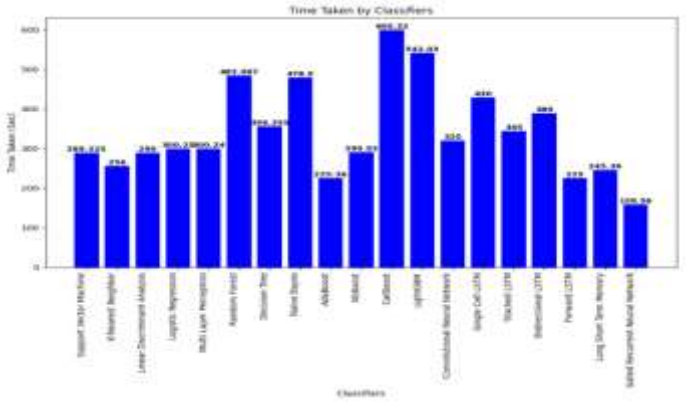

Fig. 3. Time taken without feature selection.

The Table II compares classifier performance using the Filter Approach for feature selection on the dataset. Performance metrics include accuracy, precision, recall, F1 score, and processing time. Results vary: Support Vector Machine achieves 82% accuracy, K-Nearest Neighbor 87%, and Linear Discriminant Analysis 88%. Long Short Term Memory performs at 74%. Gated Recurrent Neural Network excels with 90% precision, 92% recall, and 91% F1 score. Processing time ranges from 215.32 seconds (GRU) to 720.36 seconds (LSTM), indicating varying computational requirements.

TABLE. III PERFORMANCE COMPARISON WITH FILTER APPROACH

SL No	Classifiers	Accuracy	Precision	Recall	F1 score	Time Taken (Sec)
1	Support Vector Machine	0.82	0.82	0.82	0.82	300.25
2	K-Nearest Neighbor	0.87	0.87	0.86	0.87	354
3	Linear Discriminant Analysis	0.88	0.89	0.88	0.9	347
4	Logistic Regression	0.87	0.87	0.86	0.87	333
5	Multi-Layer Perceptron	0.81	0.83	0.82	0.83	365.5
6	Random Forest	0.75	0.8	0.75	0.77	500.24
7	Decision Tree	0.8	0.83	0.8	0.81	256.96
8	Naïve Bayes	0.74	0.78	0.74	0.76	583.13
9	AdaBoost	0.66	0.71	0.66	0.68	300.24
10	XGBoost	0.76	0.81	0.76	0.78	300.25
11	CatBoost	0.81	0.83	0.81	0.82	555.56
12	LightGBM	0.8	0.83	0.8	0.81	657.36
13	Convolutional Neural Network	0.85	0.89	0.85	0.87	330
14	Single Cell LSTM	0.61	0.702	0.7	0.69	678.36
15	Stacked LSTM	0.66	0.748	0.75	0.73	351
16	Bidirectional LSTM	0.61	0.794	0.8	0.77	699.36
17	Forward LSTM	0.66	0.84	0.85	0.81	372
14	Long Short Term Memory	0.74	0.78	0.74	0.76	720.36
15	Gated Recurrent Neural Network	0.9	0.92	0.89	0.91	215.32

The visualization plot for the Table II is presented in Fig. 4 and time taken is presented in Fig. 5.

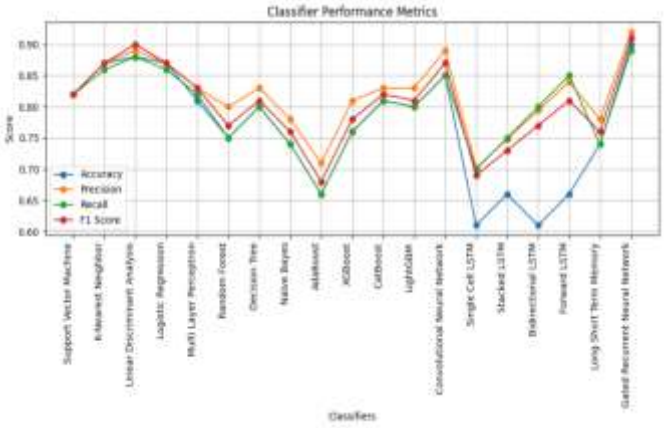

Fig. 4. Performance evaluation with filter approach.

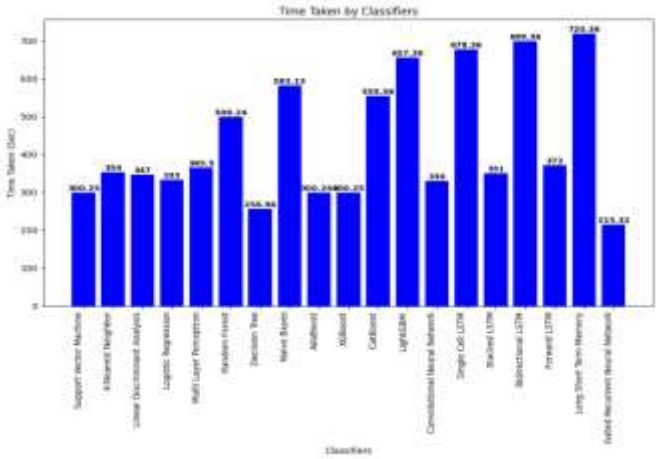

Fig. 5. Time taken for filter approach.

Table III compares classifier performance with the Wrapper Approach for feature selection on the dataset, considering accuracy, precision, recall, F1 score, and processing time. Support Vector Machine achieves the highest accuracy at 89%, while K-Nearest Neighbor reaches 84%, and Linear Discriminant Analysis achieves 80% accuracy. Precision, recall, and F1 score vary across classifiers, with Convolutional Neural Network excelling at 94%, 88%, and 91%, respectively. Processing time ranges from 900 seconds (GRU) to 32,546 seconds (Linear Discriminant Analysis), indicating distinct computational requirements. Visualization plots are available in Fig. 6, and processing time is shown in Fig. 7.

TABLE. IV PERFORMANCE COMPARISON WITH WRAPPER APPROACH

SL No	Classifiers	Accuracy	Precision	Recall	F1 score	Time Taken (Sec)
1	Support Vector Machine	0.89	0.92	0.89	0.91	4456
2	K-Nearest Neighbor	0.84	0.83	0.84	0.84	5478
3	Linear Discriminant Analysis	0.8	0.83	0.8	0.81	32546
4	Logistic Regression	0.75	0.8	0.75	0.77	6589
5	Multi-Layer Perceptron	0.8	0.83	0.8	0.81	9053
6	Random Forest	0.86	0.91	0.86	0.88	4568
7	Decision Tree	0.91	0.92	0.91	0.93	8865
8	Naïve Bayes	0.83	0.82	0.83	0.83	6545
9	AdaBoost	0.85	0.81	0.85	0.86	1866
10	XGBoost	0.82	0.81	0.82	0.82	2000
11	CatBoost	0.8	0.83	0.8	0.81	1500
12	LightGBM	0.78	0.79	0.78	0.78	3456
14	Single Cell LSTM	0.71	0.74	0.71	0.72	1521
15	Stacked LSTM	0.69	0.7	0.69	0.69	3477
16	Bidirectional LSTM	0.62	0.65	0.62	0.63	1542
17	Forward LSTM	0.6	0.61	0.6	0.6	3498
13	Convolutional Neural Network	0.88	0.94	0.88	0.91	1563
14	Long Short Term Memory	0.89	0.92	0.89	0.91	980
15	Gated Recurrent Neural Network	0.93	0.94	0.95	0.96	900

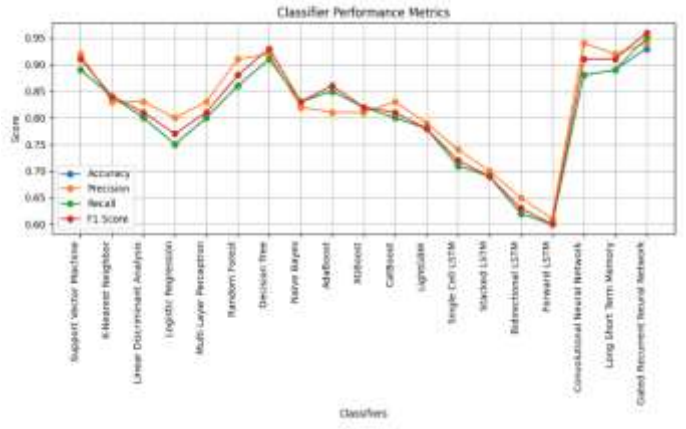

Fig. 6. Performance evaluation with wrapper approach.

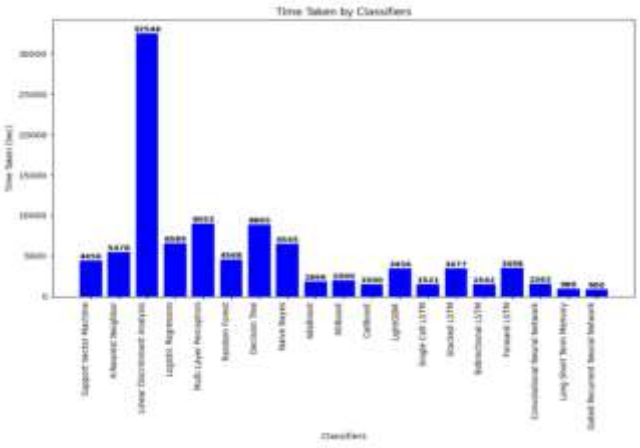

Fig. 7. Performance evaluation with wrapper approach.

Table IV compares classifier performance with the Embedded Approach for feature selection on the dataset, considering accuracy, precision, recall, F1 score, and processing time. Support Vector Machine achieves 82% accuracy, K-Nearest Neighbor reaches 86%, and Linear Discriminant Analysis attains 81% accuracy. Convolutional Neural Network excels with 98% precision, 91% recall, and 94% F1 score, ranking among the top performers. Processing time varies, from 1,200.35 seconds (GRU) to 32,658 seconds (K-Nearest Neighbor), indicating significant computational differences across models. Visualization plots are available in Fig. 8, and processing time is shown in Fig. 9.

TABLE. V PERFORMANCE EVALUATION USING EMBEDDED APPROACH

SL No	Classifiers	Accuracy	Precision	Recall	F1 score	Time Taken (Sec)
1	Support Vector Machine	0.82	0.81	0.8	0.82	12056
2	K-Nearest Neighbor	0.86	0.91	0.9	0.88	32658
3	Linear Discriminant Analysis	0.81	0.798	0.8	0.8	8986
4	Logistic Regression	0.86	0.91	0.9	0.88	7893
5	Multi-Layer Perceptron	0.85	0.87	0.9	0.86	4769
6	Random Forest	0.84	0.83	0.8	0.84	4869
7	Decision Tree	0.89	0.92	0.9	0.91	5478
8	Naïve Bayes	0.83	0.82	0.8	0.83	9866
9	AdaBoost	0.8	0.83	0.8	0.81	8255
10	XGBoost	0.78	0.79	0.8	0.78	11290
11	CatBoost	0.87	0.94	0.9	0.9	8600
12	LightGBM	0.88	0.92	0.9	0.9	6542.03
13	Convolutional Neural Network	0.91	0.98	0.9	0.94	3220
14	Single Cell LSTM	0.79	0.83	0.8	0.81	2320.03
15	Stacked LSTM	0.82	0.89	0.8	0.85	2798
16	Bidirectional LSTM	0.7	0.74	0.7	0.72	1898.03
17	Forward LSTM	0.73	0.8	0.7	0.76	2376
18	Long Short Term Memory	0.9	0.93	0.9	0.91	1295.96
19	Gated Recurrent Neural Network	0.95	0.94	1	0.96	1200.35

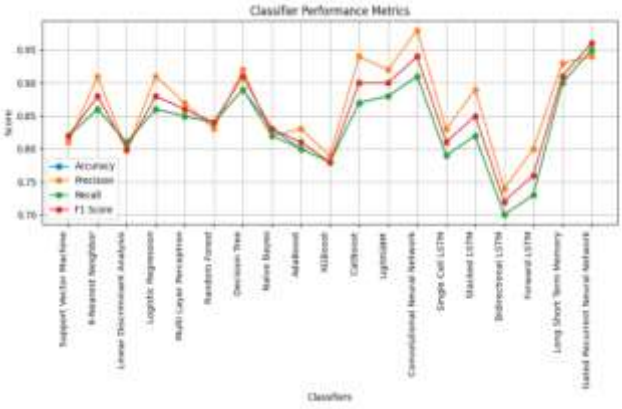

Fig. 8. Performance evaluation with embedded approach.

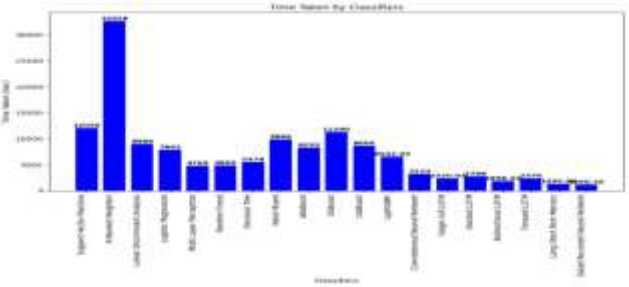

Fig. 9. Performance evaluation with embedded approach.

Table V displays ensemble-based embedded feature selection with bagging results for various classifiers, featuring accuracy, precision, recall, F1 score, and processing time. Evaluated classifiers include Convolutional Neural Network, Long Short Term Memory, Gated Recurrent Neural Network, Single Cell LSTM, Stacked LSTM, Bidirectional LSTM, and Forward LSTM. The Gated Recurrent Neural Network achieved the highest performance, with accuracy, precision, recall, and F1 score around 0.96 and a processing time of 1887 seconds. In contrast, the Forward LSTM exhibited lower performance, with scores around 0.72 and a processing time of 2160 seconds.

TABLE. VI ENSEMBLE-BASED EMBEDDED FEATURE SELECTION

SL No	Classifiers	Accuracy	Precision	Recall	F1 score	Time Taken (Sec)
1	Convolutional Neural Network	0.94	0.93	0.94	0.94	2896
2	Long Short Term Memory	0.89	0.83	0.88	0.91	2005
3	Gated Recurrent Neural Network	0.96	0.95	0.94	0.96	**1887**
4	Single Cell LSTM	0.9	0.89	0.88	0.9	1954
5	Stacked LSTM	0.84	0.83	0.82	0.84	1889
6	Bidirectional LSTM	0.78	0.77	0.76	0.78	2525
7	Forward LSTM	0.72	0.71	0.7	0.72	2160
8	FedProx	0.89	0.88	0.087	0.89	1900
9	FedAvg	0.9	0.89	0.88	0.9	1950
10	Fed+	0.88	0.87	0.87	0.88	2367

The visualization plots for the Table V is presented in Fig. 10 and time taken is presented in Fig. 11.

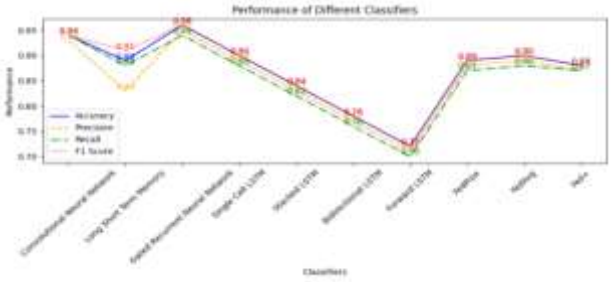

Fig. 10. Evaluation with ensemble-based embedded feature selection with bagging.

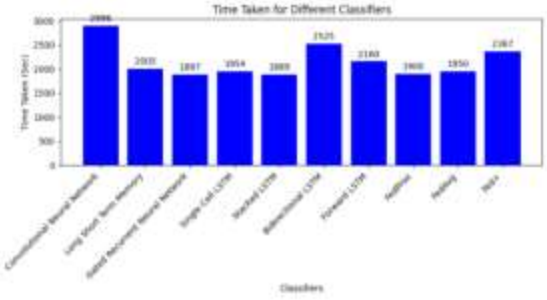

Fig. 11. Time taken with ensemble-based embedded feature selection with bagging.

This Table VI compares the performance of six classifiers in solving the task, considering accuracy, precision, recall, and F1 score. The classifiers are Gated Recurrent Neural Network, ResNet-GRU, Single Cell LSTM, Stacked LSTM, Bidirectional LSTM, and Forward LSTM. ResNet-GRU stands out as the top performer, excelling in all metrics. However, it's important to consider the computational cost, as processing time varies among models. The choice of the best classifier depends on specific application requirements and available computational resources.

TABLE. VII FEDERATED LEARNING PERFORMANCE METRIC.

SL No	Classifiers	Accuracy	Precision	Recall	F1 score	Time Taken (Sec)
1	Gated Recurrent Neural Network	0.96	0.95	0.94	0.96	1887
2	ResNet-GRU	**0.97**	0.96	0.95	0.97	**1550**
3	Single Cell LSTM	0.94	0.93	0.92	0.94	1899
4	Stacked LSTM	0.95	0.94	0.93	0.95	1562
5	Bidirectional LSTM	0.92	0.91	0.9	0.92	1911
6	Forward LSTM	0.93	0.92	0.91	0.93	1574
7	FedProx	0.86	0.86	0.85	0.86	1880
8	FedAvg	0.89	0.88	0.89	0.88	1750
9	Fed+	0.9	0.89	0.88	0.88	1987

The visualization plots for the Table VI is presented in Fig. 12 and time taken is presented in Fig. 13.

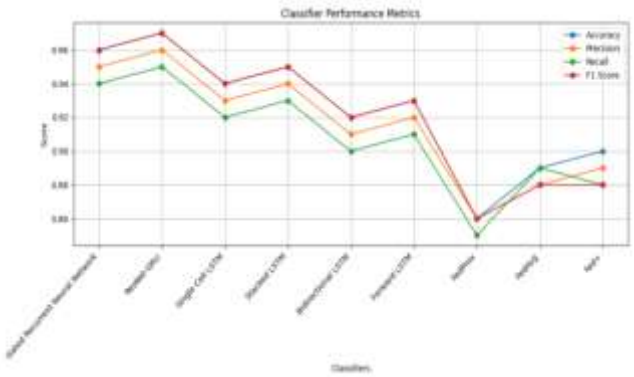

Fig. 12. Performance metric with federated learning.

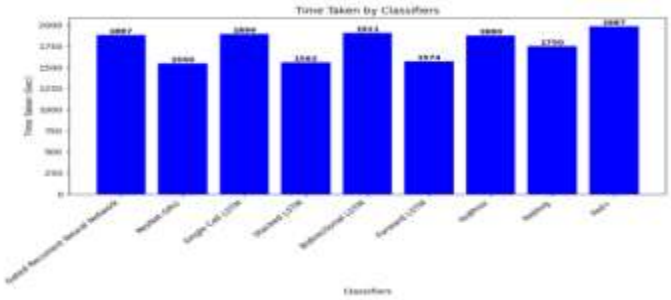

Fig. 13. Time taken with federated learning.

In federated learning, the loss function curve is essential for tracking the model's progress in a privacy-preserving setting. It enables multiple devices or clients to train a global model collaboratively without sharing raw data centrally. Each client trains its model locally, and the loss function measures the model's prediction accuracy compared to actual labels. The loss function curve depicts how this accuracy evolves over federated learning rounds. Initially, it may fluctuate as models adapt to individual data. Over rounds, it generally decreases, indicating improved performance. However, federated learning's unique challenge arises from diverse data distributions across clients, leading to varying loss function curves and potentially non-smooth trajectories due to aggregation of local models. The loss function curve obtained from our setting is shown in Fig. 14.

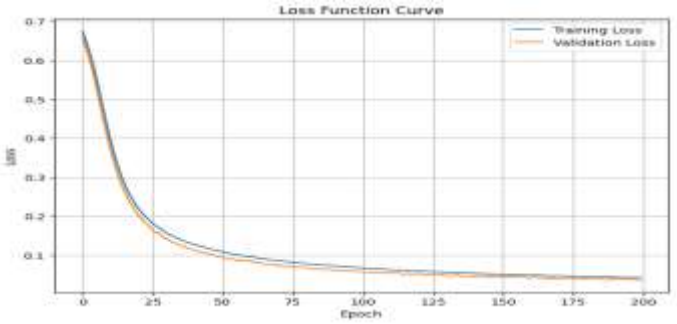

Fig. 14. Loss-Function curve.

VI. Conclusion and Future Enhancement

In this study, we analyzed the IoT23 dataset, focusing on botnet attacks, and used statistical tests to uncover patterns. We employed various feature selection methods and tested 19 classifiers, with the GRU model and embedded lasso-based feature selection performing the best. An ensemble of LassoCV models improved feature selection, and integrating the ResNet architecture further boosted the GRU model's performance. We addressed privacy concerns using federated learning, and the optimized ResNet-GRU model outperformed existing algorithms. Future work should include robustness testing, hyperparameter tuning, and exploring larger datasets. Investigating different federated learning approaches and assessing real-world deployment challenges are also promising directions for further research.

References

[1] Herabad, Mohammadsadeq Garshasbi. "Communication-efficient semi-synchronous hierarchical federated learning with balanced training in heterogeneous IoT edge environments." Internet of Things 21 (2023): 100642.

[2] Ahanger, Tariq Ahamed, Abdulaziz Aldaej, Mohammed Atiquzzaman, Imdad Ullah, and Muhammad Yousufudin. "Federated learning-inspired technique for attack classification in IoT networks." Mathematics 10, no. 12 (2022): 2141.

[3] Ahanger, Tariq Ahamed, Abdulaziz Aldaej, Mohammed Atiquzzaman, Imdad Ullah, and Muhammad Yousufudin. "Federated learning-inspired technique for attack classification in IoT networks." Mathematics 10, no. 12 (2022): 2141.

[4] Wang, Weili, Omid Abbasi, Halim Yanikomeroglu, Chengchao Liang, Lun Tang, and Qianbin Chen. "A VHetNet-Enabled Asynchronous Federated Learning-Based Anomaly Detection Framework for Ubiquitous IoT." arXiv preprint arXiv:2303.02948 (2023).

[5] Sánchez, Pedro Miguel Sánchez, Alberto Huertas Celdrán, Timo Schenk, Adrian Lars Benjamin Iten, Gérôme Bovet, Gregorio Martínez Pérez, and Burkhard Stiller. "Studying the robustness of anti-adversarial federated learning models detecting cyberattacks in iot spectrum sensors." IEEE Transactions on Dependable and Secure Computing (2022).

[6] Gkillas, Alexandros, and Aris Lalos. "Resource Efficient Federated Learning for Deep Anomaly Detection in Industrial IoT applications." In 2023 24th International Conference on Digital Signal Processing (DSP), pp. 1-5. IEEE, 2023.

[7] Gkillas, Alexandros, and Aris Lalos. "Resource Efficient Federated Learning for Deep Anomaly Detection in Industrial IoT applications." In 2023 24th International Conference on Digital Signal Processing (DSP), pp. 1-5. IEEE, 2023.

[8] Gkillas, Alexandros, and Aris Lalos. "Resource Efficient Federated Learning for Deep Anomaly Detection in Industrial IoT applications." In 2023 24th International Conference on Digital Signal Processing (DSP), pp. 1-5. IEEE, 2023.

[9] Rey, Valerian, Pedro Miguel Sánchez Sánchez, Alberto Huertas Celdrán, and Gérôme Bovet. "Federated learning for malware detection in IoT devices." Computer Networks 204 (2022): 108693.

[10] Weinger, Brett, Jinoh Kim, Alex Sim, Makiya Nakashima, Nour Moustafa, and K. John Wu. "Enhancing IoT anomaly detection performance for federated learning." Digital Communications and Networks 8, no. 3 (2022): 314-323.

[11] Lian, Zhuotao, and Chunhua Su. "Decentralized federated learning for Internet of Things anomaly detection." In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security, pp. 1249-1251. 2022.

[12] Huong, Truong Thu, Ta Phuong Bac, Kieu Ngan Ha, Nguyen Viet Hoang, Nguyen Xuan Hoang, Nguyen Tai Hung, and Kim Phuc Tran. "Federated learning-based explainable anomaly detection for industrial control systems." IEEE Access 10 (2022): 53854-53872.

[13] Halder, Subir, and Thomas Newe. "Radio fingerprinting for anomaly detection using federated learning in LoRa-enabled Industrial Internet of Things." Future Generation Computer Systems 143 (2023): 322-336.

[14] Sáez-de-Cámara, Xabier, Jose Luis Flores, Cristóbal Arellano, Aitor Urbieta, and Urko Zurutuza. "Clustered federated learning architecture for network anomaly detection in large scale heterogeneous IoT networks." Computers & Security 131 (2023): 103299.

[15] Truong, Huong Thu, Bac Phuong Ta, Quang Anh Le, Dan Minh Nguyen, Cong Thanh Le, Hoang Xuan Nguyen, Ha Thu Do, Hung Tai Nguyen, and Kim Phuc Tran. "Light-weight federated learning-based anomaly detection for time-series data in industrial control systems." Computers in Industry 140 (2022): 103692.

[16] Fan, Jiamin, Guoming Tang, Kui Wu, Zhengan Zhao, Yang Zhou, and Shengqiang Huang. "Score-VAE: Root Cause Analysis for Federated Learning-based IoT Anomaly Detection." IEEE Internet of Things Journal (2023).

[17] Raza, Ali, Kim Phuc Tran, Ludovic Koehl, and Shujun Li. "AnoFed: Adaptive anomaly detection for digital health using transformer-based federated learning and support vector data description." Engineering Applications of Artificial Intelligence 121 (2023): 106051.

[18] Jithish, J., Bithin Alangot, Nagarajan Mahalingam, and Kiat Seng Yeo. "Distributed Anomaly Detection in Smart Grids: A Federated Learning-Based Approach." IEEE Access 11 (2023): 7157-7179.

[19] Wang, Xiaofeng, Yonghong Wang, Zahra Javaheri, Laila Almutairi, Navid Moghadamnejad, and Osama S. Younes. "Federated deep learning for anomaly detection in the internet of things." Computers and Electrical Engineering 108 (2023): 108651.

[20] Wang, Xiaoding, Wenxin Liu, Hui Lin, Jia Hu, Kuljeet Kaur, and M. Shamim Hossain. "AI-empowered trajectory anomaly detection for intelligent transportation systems: A hierarchical federated learning approach." IEEE Transactions on Intelligent Transportation Systems 24, no. 4 (2022): 4631-4640.

[21] Shubyn, Bohdan, Dariusz Mrozek, Taras Maksymyuk, Vaidy Sunderam, Daniel Kostrzewa, Piotr Grzesik, and Paweł Benecki. "Federated learning for anomaly detection in industrial IoT-enabled production environment supported by autonomous guided vehicles." In International Conference on Computational Science, pp. 409-421. Cham: Springer International Publishing, 2022.

[22] Toldinas, Jevgenijus, Algimantas Venčkauskas, Agnius Liutkevičius, and Nerijus Morkevičius. "Framing Network Flow for Anomaly Detection Using Image Recognition and Federated Learning." Electronics 11, no. 19 (2022): 3138.

[23] Wu, Dongmin, Yi Deng, and Mingyong Li. "FL-MGVN: Federated learning for anomaly detection using mixed gaussian variational self-encoding network." Information processing & management 59, no. 2 (2022): 102839.

[24] Fedorchenko, Elena, Evgenia Novikova, and Anton Shulepov. "Comparative review of the intrusion detection systems based on federated learning: Advantages and open challenges." Algorithms 15, no. 7 (2022): 247.