International Journal of Research and Development in Engineering Sciences
ISSN: 2582 - 4201(Online) , http://www.ijrdes.com/
Regular Issue, Vol. 7, Issue. 6 (November – December) , 2025 , Pages: 1 - 19
Received: 24 February 2025 ; *Accepted:* 19 October 2025 ; *Published:* 01 November 2025
https://doi.org/10.63328/IJRDES-V7RI6P1
Original Paper : Applied Research – Compare and Contrast Research Paper

Predicting Botnet Attack and Severity in Fog Computing Networks using Deep Learning with Reinforced Feature Optimization

Surya Pavan Kumar Gudla [1], **Preeti Nutipalli** [2], **Ramesh Yegireddi** [3], **T Chalapathi Rao** [4]

[1, 3, 4] Department of Computer Science and Engineering, Aditya Institute of Technology and Management, K Kotturu, Tekkali 532201, AP, India 532201;
[2] Department of Computer Science and Engineering, Anil Neerukonda Institute of Technology and Sciences, Sangivalasa, Visakhapatnam, AP, India, 531162;

** Corresponding Author: Surya Pavan Kumar Gudla ; pavan1980.mca@gmail.com*

Abstract: As an extension of cloud services to the edge, Fog computing is a current paradigm for latency-sensitive and resource-constrained applications. Nevertheless, its distributed architecture makes it vulnerable to sophisticated cybersecurity threats, especially botnet assaults. To tackle this problem, we propose a reinforcement learning–enhanced deep learning framework for multi-class classification of botnet attacks in a fog computing environment. At the heart of our approach is a Bidirectional Long Short-Term Memory (BiLSTM) network, a well-known sequential data modeling network. Reinforcement learning based feature selection was used to find and retain the best relevant set of attributes such that detection can be maximized. We performed extensive experimentation and evaluated over four cross-validation folds using a real-life dataset. This proposed model consistently proved micro-averaged precision, recall, F-score, and accuracy greater than 95% on all classes — namely High, Moderate, Low, and Normal attack intensities — over three benchmarked datasets, BADD, RLUF, and DIBCA. Particularly, precision and sensitivity scores were relatively close to perfect, ranging from near-perfect scores for higher severity attacks in terms of detection and false alarms. By proving that deep learning can be further improved by incorporating reinforced feature optimization, this study provides evidence that deep learning can be further boosted to assist in fog networks with deep botnet detection. The proposed framework is ready to adapt, efficient, and holds great promise for real-time intrusion detection based on Internet of Things (IoT) backed fog infrastructures.

Keywords: Fog Computing, Botnet Detection, DL, BiLSTM, Reinforcement Learning, IoT Security, DDoS.

1. Introduction

The Fog computing networks are gaining popularity due to their capacity to bring a decentralized computing infrastructure close to end users and devices. Implementing network security is difficult because fog nodes are susceptible to botnet attacks [1]. Cybercriminals run botnets composed of infected devices. Spam, DDoS attacks, and identity theft are common uses [1].

Researchers have proposed machine learning-based botnet identification and mitigation techniques for fog computing[3]. Because it can learn and recognize data patterns, machine learning is popular for botnet detection [3] because it can learn and recognize data patterns. This reveals inconsistencies. Deep learning can identify botnets by automatically extracting relevant features from raw data and improving classification accuracy [4].

1.1. Botnet Attacks

Botnet attacks create many problems to network systems [5]. Controlled by a person called botmaster, these attacks use many infected computers. They send spam, steal data, or do DDoS attacks. Catching them is difficult because like normal traffic, these bots behave. They hide identity and change patterns often, making detection sometimes slow and confusing [6]. Through weak systems, a botnet first spreads. Malware enters using fake emails or harmful websites, and joins the system to a group once it has entered. All bots listen to the attacker's command after joining. In one type, called centralized, all bots are controlled by one server. Without a single point of control, bots talk to each other and follow commands in a decentralized manner. Removing one will not stop the whole system, as shown in Figure 1.

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

It starts attacks when botnet gets full control. Sending too many requests to shut down websites or stealing user data silently, the bots do. Some just sit and collect information in the background. To hide, in many times, bots change their IP addresses or communication path. Bots keep spreading and avoid detection because the controller updates them regularly. Money is earned by some botnets through different ways. Cryptocurrencies are mined without permission, stolen personal data is sold, or ads are clicked by them. If caught by police or cybersecurity teams, the botnet sometimes shuts down. The attacker before getting caught, sometimes stops it.

Fake emails are sent by botnets that trick people into giving bank passwords. Unwanted ads are shown or private data collected by them. Passwords are guessed or accounts broken into during brute force attacks.

Data from websites is taken or pirated files spread in many cases. Botnets are dangerous because their structure is smart and actions hidden. Problems for online systems safety is created seriously by botnets. Early detection using behavior and traffic patterns is one possible way to stop them.

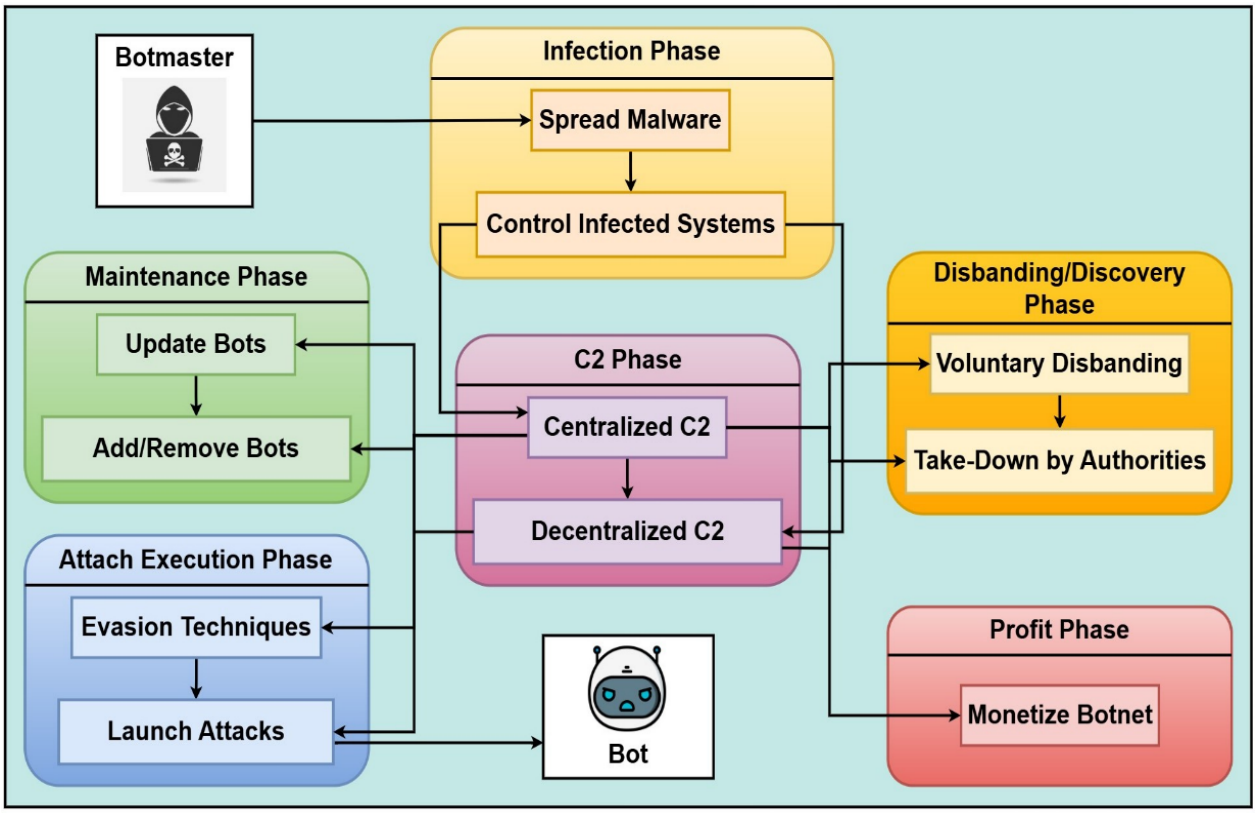

Figure.1 Botnet attack model

1.2. Fog Computing Model

Fog computing shifts computation and services from centralized data centers to the network edge, enabling low-latency processing and efficient bandwidth usage. Its architecture diagram shown in Figure 2 includes end devices, fog nodes, and cloud centers, where fog nodes act as intermediaries to process data closer to its source [7]. These nodes, embedded in devices like routers or micro data centers, interface with edge devices such as sensors and mobile units.

Fog nodes can also connect to traditional cloud infrastructure or local data centers. They may send aggregated, filtered, or processed data further up the chain, relieving the need to send all raw data directly to the central cloud [8]. The networking layer enables communication between edge devices, fog nodes, and the cloud. It can include various types of connections, like LAN, WAN, cellular, or Wi-Fi. The fog orchestrator is responsible for managing and orchestrating the resources within the fog computing environment. It handles tasks like resource allocation, monitoring, security, and ensuring the right data gets to the right place at the right time. Fog nodes often include capabilities for real-time analytics and data processing. This allows immediate insights and action on the data without having to send it all to a central cloud or data center. Machine learning can identify botnets by recognizing data patterns and anomalies. By extracting relevant characteristics from raw data, deep learning (DL) can improve classification accuracy for botnet detection. In order to identify botnets, cloud computing, IoT, and mobile networks employ "LSTM (Long Short-Term Memory)" and "CNNs (Convolutional Neural Networks)" networks.

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

Deep learning requires a large quantity of training data and is computationally expensive. Moreover, selecting relevant features from a large dataset can be difficult and have a negative effect on model performance. To improve the precision of deep learning models, researchers have developed a variety of feature selection strategies. These methods utilize filter, wrapper, and embedded methods. Reinforcement learning is another machine learning technique for feature selection in deep learning models. Reinforcement learning promotes the development of decision-making skills by rewarding or punishing choices [9]. To maximize the accuracy of deep learning models while minimizing computational cost, reinforcement learning selects features.

1.3. Contributions

Through this manuscript, we aim for the following key contributions: We developed a BiLSTM-based multi-class classification model for detecting various botnet attacks in fog computing networks, which will effectively analyze sequential network traffic data.

(a). We integrated reinforcement learning for feature optimization, allowing the model to autonomously select the most informative and minimal set of botnet features.

(b). We introduced novel heuristic regression-enhanced reinforcement framework, which further improved accuracy and other metrics, setting a new benchmark for intelligent botnet detection in fog computing environments.

(c). We conducted rigorous 4-fold cross-validation using real-world dataset to validate the robustness and reliability of the approach.

1.4. Organization of the manuscript

This paper's structure includes a discussion on introducing botnet attacks and severity in fog computing networks.

Section 2 explores related research and several models of the proposed approach, using conventional botnet attack features and traffic flow features to train the BiLSTM associated with the reinforcement learning approach and materials connected to the proposed approach, which have been explored in Section 3. Section 4 shows the experimental setup and the data used.

The proposed model has been tested experimentally and contrasted with existing models in Section 5. The explanation of this study's conclusion and a list of sources may be found in Section 6.

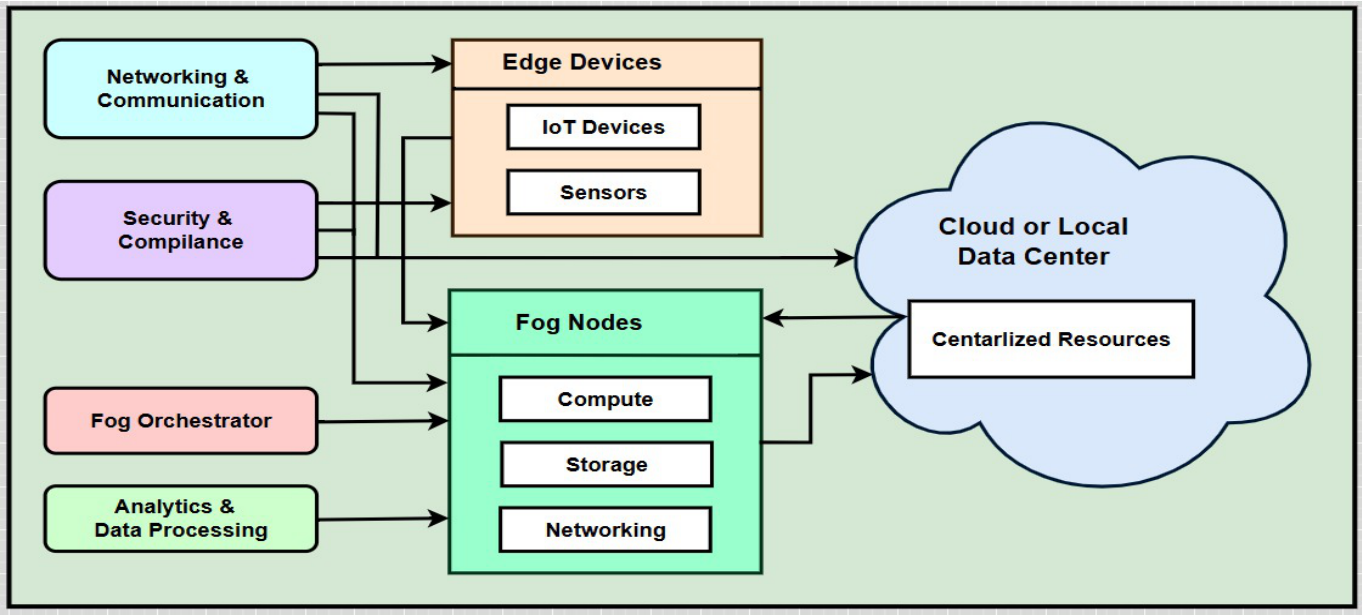

Figure.2 Fog Computing Model

2. Related Research

The focus of the reviewed articles is the application of deep learning models as well as machine learning techniques to botnet detection. Multiple factors, including the quality and quantity of data utilised for model training, the complexity of the approach, the costs associated with implementation, and the susceptibility to cyber threats, influence the effectiveness of the suggested methodologies. Both benefits and drawbacks are endemic to these methods.

Gopinath, Mohana et al. [10] extensively surveyed deep learning models for identifying threats within IoT environments. The work outlines the capacity of DL models in malware detection.

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

Rawat, Romil et al., [11] proposes a model that decentralizes threat detection using fog-based deep learning deployments. The reviewed model highlights distributed execution, but it does not look into the level of detail in threat assessment, which is a key contribution of the current work.

Syed, Naeem Firdous et al., [12] explores a deep learning approach for fog and cloud infrastructure. Feature selection is emphasized as a critical step in improving classification accuracy.

Balasubramaniam, S. et al., [13] focuses on hyperparameters tuning to enhance deep learning-based DDoS detection. The study demonstrates improved accuracy but lacks implementation of multi-class classification or attack severity differentiation.

Nandanwar, Himanshu et al., [14] introduces AttackNet, a model optimized for dynamic and complex traffic scenarios, to support the adoption of BiLSTM for fog environments. Ali, Mudasir et al., [15] contributes a layered architecture that improves detection accuracy but omits severity classification and advanced feature selection, reinforcing the uniqueness of the proposed framework.

Finally, Li, Yiran et al., [16] proposes CROSSBEAM, which strengthens distributed learning security and complements the decentralized learning context of fog computing.

Ibitoye et al., scholarly work [17] focuses on the examination of adversarial attacks aimed at deep learning algorithms employed in the context of intrusion detection in IoT networks. This study shows SNN (Self-normalizing Neural Networks) superiority over FNN (Feedforward Neural Networks) in classifying the attacks.

Ge, Mengmeng, et al., [18] proposed a deep learning-based intrusion detection method for IoT networks with limitation in detecting unfamiliar forms of attacks.

Ferrag and Maglaras [19] proposed the DeepCoin framework. This work combines blockchain and deep learning technologies to improve the efficiency and security of energy transactions in Smart Grids.

Sari and Susilo [20] focused on using deep learning to identify DoS attacks and improve security effectiveness in IoT networks.
Muhammad Ghulam et al., [21] proposed a Stacked Autoencoder-based IDS that employs machine learning (ML) techniques to combat financial misconduct. This system exhibits versatility in identifying various types of fraudulent activity but its ability to identify novel or unfamiliar forms of fraudulent behaviour is limited, and it may produce erroneous results in the form of false positives or false negatives.

Zhou et al., [22] presented a technique Hierarchical Adversarial Attacks (HAA) that can effectively compromise NIDS (Network Intrusion Detection Systems) than Graph Neural Networks (GNN).

Mudassir, Mohammed et al., [23] presented a multilayer deep learning technique for identifying botnets in IIoT networks.

Popoola Segun et al., [24] focused on optimising the hyperparameters of deep learning models to detect botnets in IoT networks. Catillo Marta, et al., [25] proposed a lightweight and cross-device method for intrusion detection in IoT networks using a semi-supervised approach.

The employed technique is scalable and adaptable to the ever-changing characteristics of modern IoT networks.

Kirubavathi, G., and U. K. Sridevi [26] studied ML and DL methodologies in identifying the cyber intrusions in IoT environments.

Elsayed,et al. [27] presented an economic deep learning model to recognize botnet attacks. The advantages and disadvantages of these reviewed articles are presented in Table 1.

Nevertheless, deep learning models can be computationally intensive and require a lot of data for efficient training.

In addition, selecting pertinent features from a large dataset can be difficult, and selecting the incorrect features can result in poor model performance.

Using a deep learning-based multi-class classification strategy, this study proposes a novel method for addressing these problems in fog computing networks for predicting botnet attacks.

This study proposes using BiLSTM as the deep learning model and a heuristic regression technique for reinforcement learning to optimise feature selection.

The proposed method seeks to increase the precision of botnet attack prediction in fog computing networks and lower the computational cost of training deep learning models. Table 1 highlights the strengths and gap in existing research on botnet detection. From the above it is evident

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

Table.1 The checklist of the articles reviewed in the context of the proposed model

Reference	IDS	Botnet Attack Prediction	Botnet Prediction	Botnet Attack Severity Prediction	Machine Learning	Deep Learning	Feature Optimization Addressed	Reinforcement Learning
[17]	✓	✓	×	×	✓	✓	×	×
[18]	×	✓	×	×	×	✓	×	×
[19]	✓	×	×	✓	×	✓	×	×
[20]	✓	×	×	×	✓	✓	×	×
[21]	✓	×	×	×	✓	×	×	×
[22]	×	×	✓	×	×	✓	×	×
[23]	×	✓	×	×	×	✓	×	×
[24]	×	×	✓	×	×	✓	✓	×
[25]	✓	×	✓	×	✓	×	×	×
[26]	×	✓	×	×	✓	✓	×	×
[27]	×	✓	×	×	×	✓	×	×

that none of the surveyed studies have incorporated reinforcement learning, which represents a significant research gap. The proposed study advances the field by not only addressing botnet attack prediction and severity analysis but also integrates feature optimization and reinforcement learning, thereby providing a more comprehensive and adaptive defense framework. The capacity of deep learning models to automatically recognize intricate features and patterns improves the performance and precision of botnet detection systems. It is possible to improve the accuracy and efficiency of botnet detection systems by selecting and representing features with care. This article proposes using reinforcement learning techniques to improve feature selection and botnet attack prediction. Botnet detection can be improved by using reinforcement learning to automate system behavior and decision-making.

3. Model Narrative and Methods Used

The objective of the proposed method is to estimate the botnet attack severity in four different categories listed as high, medium, low, and normal (not-an-attack). A deep reinforcement learning, which is based on the deep learning technique BiLSTM [29], has been portrayed to achieve the target objective.

The proposed approach is using conventional botnet attack features and traffic flow features to train the BiLSTM associated with a reinforcement learning approach. The phases involved in this approach are data sampling, revising the data samples used for training by the reinforcement learning approach, feature extraction, optimal feature selection, and portraying fitness functions for the training phase, as well as defining the prediction phase. The overall process of the suggested approach is outlined in the following Figure 3.

The objective of the proposed method is to estimate the botnet attack severity in four different categories listed as high, medium, low, and normal (not-an-attack). A deep reinforcement learning, which is based on the deep learning technique BiLSTM [29], has been portrayed to achieve the target objective. The proposed approach is using conventional botnet attack features and traffic flow features to train the BiLSTM associated with a reinforcement learning approach. The phases involved in this approach are data sampling, revising the data samples used for training by the reinforcement learning approach, feature extraction, optimal feature selection, and portraying fitness functions for the training phase, as well as defining the prediction phase. The overall process of the suggested approach is outlined in the following Figure 3.

3.1. The Features

The features preferred to use in this contribution are the combination of the conventional botnet features, as well as traffic flow features. Several conventional botnet attack features [29] can be used in botnet attack prediction using deep learning. These features can be categorized into four groups: network traffic-based features, system event-based features, bot behavior-based features, and botnet topology-based features, which are listed in Table 2.

(a). Traffic-based features for networks: These features are extracted from the network traffic generated by botnets.
(b). System event-based features: These features are extracted from the system events generated by botnets.
(c). Behavior-based features for bot: These features are extracted from the behavior of bots in the network.
(d). Botnet topology-based features: These features are taken from the design of botnets.

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

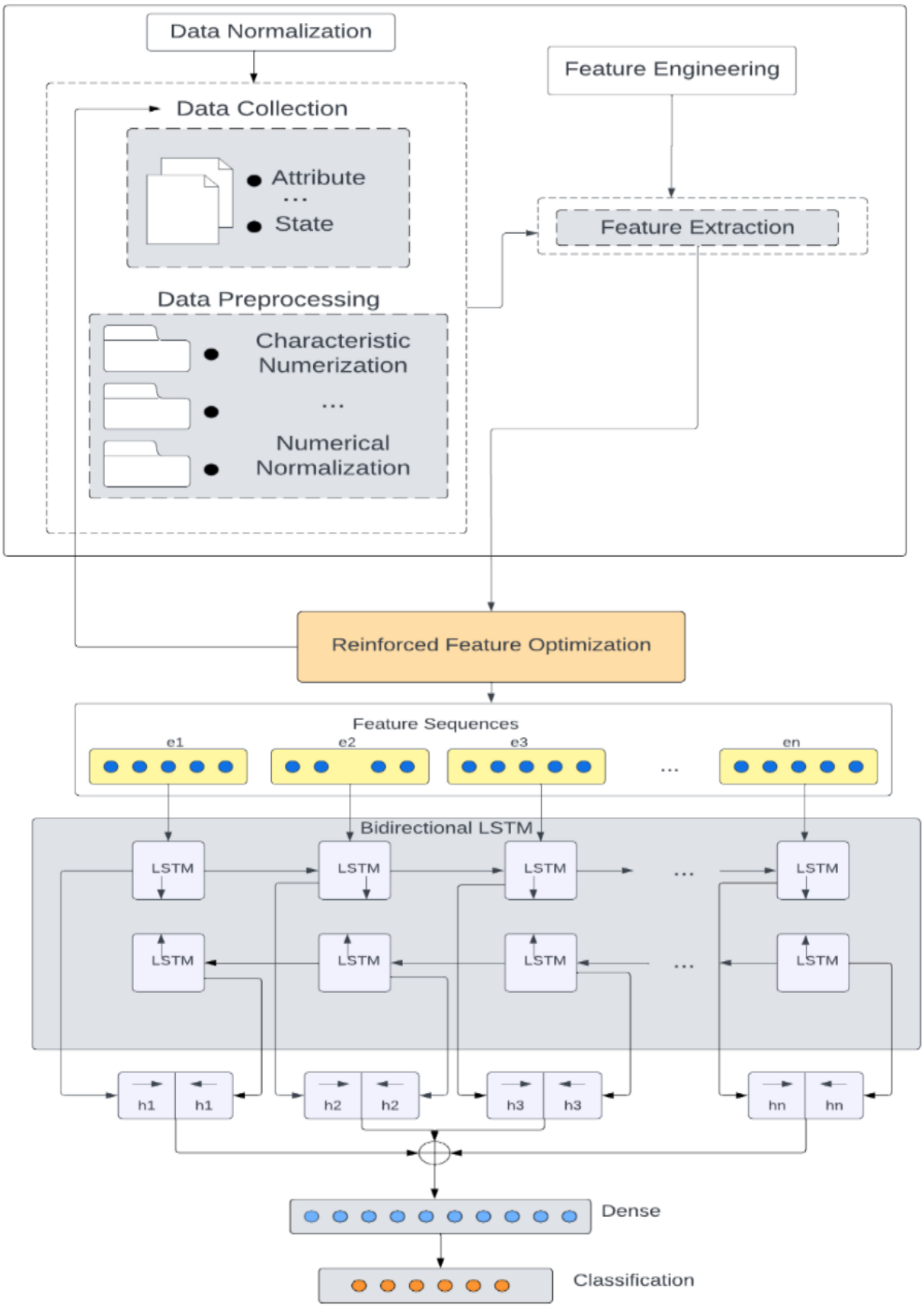

Figure.3 Architecture to the suggested model

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

Table.2 Conventional Botnet Attack Features

Feature Category	Specific Features
Network traffic-based features	Number of packets sent/received, Packet size distribution, Protocol distribution, Port distribution, Traffic volume
System event-based features	Number of system calls, Type of system calls, Time duration of system calls, File system activities, Registry activities
Bot behavior-based features	Number of bots in the network, Command and control (C&C) communication patterns, Malware propagation patterns, Malware injection patterns, Malware evasion techniques
Botnet topology-based features	Botnet size, Botnet density, Botnet hierarchy, Botnet communication patterns, Botnet geographical distribution

The raw pcap files shall be explored to extract the feature values. These extracted features shall be arranged in two dimensional matrix format such that each column represents the values obtained for each feature. And each row represents the values of all features of each attack or normal record.

3.2. *Feature Optimization*

This step examines the most distinctive attributes within different classes {high, medium, low, and normal}. The Kruskal-Wallis Test is used to find the best features. It is also be used to find the correlation between different sets of values. The Kruskal-Wallis Test (KWS-Test) [30] is an extension of the Mann-Whitney U (MWU) Test and used to compare more than two independent samples. It ranks all the observations from all the groups and tests whether the the medians of the groups are very different from each other. The KWS-Test is a non-parametric statistical method used when the assumptions of normality and equal variances between the groups are not met.

3.3. *Kruskal-Wallis Test*

The test procedure begins by ranking all data from all groups together, from smallest to largest. Then, it

calculates the sum of ranks for each group. The test statistic H is computed using the following Eq. 1.

$$H = \left(\frac{12}{n(n+1)}\right)\left[\sum\left(\frac{R_i^2}{n_i}\right)\right] - 3(n+1) \quad (1)$$

Where:

- n is the total number of observations (sum of the sizes of the four input vectors),
- R_i is the sum of the ranks for group i,
- n_i is the number of observations in group i,

$\sum\left(\frac{R_i^2}{n_i}\right)$ is the sum over each group i of the squared ranks divided by the number of observations in group i.

Given four vectors and , representing the values observed for a feature in each set of records of the four labels (high, moderate, low, and normal respectively), the Kruskal-Wallis test is applied to identify differences among these vectors. Initially, all values in the vectors are ranked together. Then, the sum of ranks for each group is calculated. Using these, the test statistic is computed as shown above.

The null hypothesis of the Kruskal-Wallis test assumes that all groups originate from identical populations. The alternative hypothesis states that at least one population differs. The decision to reject or not reject the null hypothesis is based on the p-value derived from the statistic. A small p-value (typically) indicates strong evidence to reject the null hypothesis.

3.4. *Fitness Function*

The fitness function identifies the low, mean, and high values {low, mean, high} of each feature $f_k \in ivF_i$ across all labels

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

$$l \in \{\text{high(h)}, \text{medium(m)}, \text{low(l)}, \text{normal(n)}\}$$

. These values are used to compute status-measure coefficients for each feature corresponding to each label, as denoted in Eq. 2.

$$\{l \in \{\text{high(h)}, \text{medium(m)}, \text{low(l)}, \text{normal(n)}\}\} \quad (2)$$

This process follows a functional method described in the pseudocode to compute the status-measure coefficients.

Fitness Coefficient Computation:

Let the feature values be denoted as a vector $fv = \{e_1, e_2, ..., e_n\}$. The coefficients are calculated as:

Mean of feature values: $\mu = \frac{1}{|fv|}\sum_{i=1}^{|fv|} e_i$

Standard deviation of feature values:

$$\delta = \frac{1}{|fv|}\sum_{i=1}^{|fv|}\sqrt{\mu - e_i)^2}$$

Central pivot: $P = \frac{1}{3}(e_{\min} + e_{\max} + \mu) - \delta$

Lower resistance: $R_{\min} = \log(100) - e_{\min}$

Higher resistance: $R_{\max} = \log(10) + (e_{\max} - e_{\min})$

Lower support: $S_{\min} = \log_{10}(100) - e_{\max}$

Higher support: $S_{\max} = \log(10) - (e_{\max} - e_{\min})$

The final output is the status-measure coefficient set: $smc = \{R_{\min}, R_{\max}, S_{\min}, S_{\max}\}$

These coefficients are returned for each feature-label combination and can be used in further supervised learning tasks to link features with severity labels of botnet attacks.

3.5. Botnet Attack Confidence Calculation

In this step, the goal is to find how much a record matches known severity types. The best features selected are only used. Each test record is compared with label-wise feature patterns saved during training. The record is checked against patterns from training to decide if the attack is high, medium, low, or normal.

Let the test feature vector be $tfv = \{e_1, e_2, ..., e_n\}$. The confidence values are computed as follows:

$$\mu = \frac{1}{|tfv|}\sum_{i=1}^{|tfv|} e_i$$

values:

Standard deviation of the values:

$$\delta = \frac{1}{|tfv|}\sum_{i=1}^{|tfv|}\sqrt{\mu - e_i)^2}$$

$$P = (e_{\min} + e_{\max} + \mu - \delta) \cdot \left(\frac{\log 2}{\log 6}\right)$$

Central Pivot:

Support Confidence:

$$sc = \text{round}(\log 3,1) \cdot (e_{\min} + e_{\max}) - \delta$$

Range Confidence: $rc = \log 100 \cdot P - sc$

The result is the pair of confidence values: $\{sc, rc\}$

$sc = \text{round}(\log 3,1) \cdot (e_{\min} + e_{\max}) - \delta$

Range Confidence: $rc = \log 100 \cdot P - sc$

The result is the pair of confidence values: $\{sc, rc\}$

These metrics indicate the support and range of confidence for each test feature and help classify the incoming network record based on the botnet's predicted severity.

3.6. Reinforcement Learning-Based Sample Selection

The training phase in DL influences the decision accuracy of the model. In the proposed model, reinforcement learning is integrated to curate the input data for the BiLSTM model suitable for multi-class classification tasks. The steps involved in the proposed framework are depicted in Figure 4.

3.6.1. Overview of Process

A labeled samples are prepared to reflect botnet scope and severity. For each label, mean and deviation (standard deviation) are calculated for all features. The process involves:

Computing the absolute difference between the mean and standard deviation for each feature.

(a). Selecting records where the feature value is greater than or equal to this absolute difference.
(b). Ranking the features based on the number of such qualifying records.
(c). Collecting unique records from top-ranked features, ensuring the total count does not exceed the predefined sampling size coefficient.
(d). Using the selected records as training corpus.
(e). Predicting labels of the remaining records.
(f). Adding accurately predicted records to the training set.
(g). If no correct predictions are made, initiate a new cycle of sample selection.

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

3.6.2. Initialization

Let be the labeled dataset, where is the feature vector and is the corresponding label representing botnet scope and severity. Define:

S : an initially empty set to store selected samples.
n : total number of records in D.
m : number of features in x_i.
k : predefined sampling size coefficient.

3.6.3. Feature Analysis

For each label y and feature $j = 1, 2, ..., m$, compute the mean μ_j and standard deviation σ_j in Eq. 3.

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij} \quad \sigma_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \mu_j)^2} \tag{3}$$

Compute the absolute difference for each feature:

$$\delta_j = \left|\mu_j - \sigma_j\right|$$

3.6.4. Sample Selection

Select records for each feature j where $x_{ij} \geq \delta_j$, and denote the selected record set as D_j in Eq. 4.

$$D_j = \left\{ i : x_{ij} \geq \delta_j \right\} \tag{4}$$

Rank features based on the size of D_j in Eq. 5.

$$\text{Rank} : \left|D_1\right|, \left|D_2\right|, ..., \left|D_m\right| \tag{5}$$

Aggregate record sets starting from the smallest D_j, continuing until the combined number of unique records does not exceed k. Denote the final selected set as:

$$S = \bigcup D_j \quad \text{suchthat} \quad \left|S\right| \leq k$$

Remove the selected records in S from dataset D.

3.7. Model Training and Prediction

Train the model using the selected sample set S. Use the model to predict the labels of the remaining records in D. Sample Update: Transfer correctly predicted records from D to S. Repeat the sample selection and model training steps until no new records are predicted correctly, or D becomes empty.

This sample selection algorithm adjusts the training data dynamically, enabling the model to optimize feature representation and improve overall performance.

Model Training and Prediction: The model is trained on a labeled sample set with the goal of identifying the corresponding labels during prediction accurately. This process integrates optimal feature selection and the computation of feature fitness coefficients. The sequence of operations is as follows:

3.7.1. Optimal Feature Selection

Let $F = \left\{ f_1, f_2, ..., f_{|F|} \right\}$ be the set of all features, and let $Fo = \{\}$ be the set to store optimal features. Let the set of labels be $B = \{h, m, l, n\}$ representing high, medium, low, and normal.For each $f_i \in F$, perform the Kruskal-Wallis H test using the vectors $f_{i_h}, f_{i_m}, f_{i_l}, f_{i_n}$ which contain values of feature f_i in records labeled high, moderate, low, and normal respectively. If the p-value from the H test is less than 0.05, indicating statistical significance, then include f_i in the optimal feature set Fo.

3.7.2. Computing Feature Fitness Coefficients

For each label, and for each feature, compute the fitness coefficient as:

$$fc_b(f_i) = \frac{\mu_b}{\sigma_b}$$

Where μ_b is the mean and σ_b is the standard deviation of feature for label.

Botnet Attack Confidence Calculation: Let be the total record corpus and the selected records, with denoting a leftover test record.

For each feature:

- Let e_i be the feature value used as the mean of a normal distribution $\mathrm{N}\left(e_i, \sigma^2\right)$, where σ is the standard deviation of f_i across the dataset.
- Generate a vector vf_i of size

$$\max\left(\left|tRC_h\right|, \left|tRC_m\right|, \left|tRC_l\right|, \left|tRC_n\right|\right)$$

from this normal distribution.

- Compute the Botnet Attack Confidence of f_i as:

$$ac(f_i) = \sum\left(fc_b(f_i) \cdot P(vf_i \mid b)\right)$$

Iterating Over Labels and Optimal Features: For each label $l \in \{h, m, l, n\}$ and each optimal feature $f_j^{tr} \in oF$,

calculate the botnet attack impact score using the following rules:

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

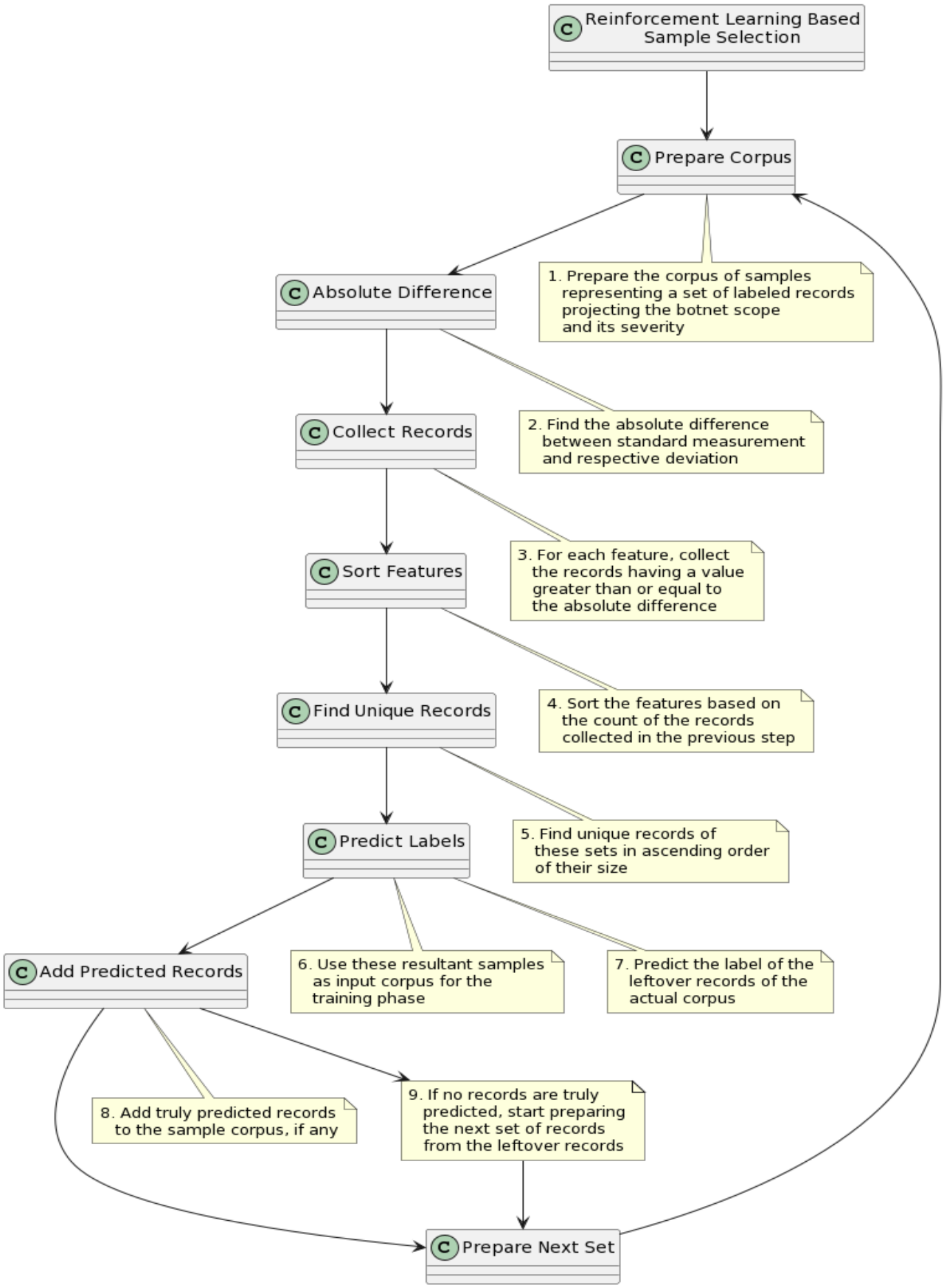

Figure.4 Sample Selection by Reinforcement Learning

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

3.7.3. Botnet Attack Impact Score for the Record:

Compute the average impact score for label in Eq. 6.

$$\langle tr_h \rangle = \frac{1}{|oF_h|}\sum_{i=1}^{|oF_h|} bass_h\left(f_i^{tr}\right) \quad (6)$$

Calculate the deviation in Eq. 7.

$$\delta\left(tr_h\right) = \frac{1}{|oF_h|}\sum_{i=1}^{|oF_h|}\sqrt{\left(\langle tr_h \rangle - bass_h\left(f_i^{tr}\right)\right)^2} \quad (7)$$

Final impact score is calculated in Eq. 8.

$$bass_h\left(tr\right) = \langle tr_h \rangle - \delta\left(tr_h\right) \quad (8)$$

Repeat the above steps for labels m , l , and n .

$$bass_l\left(f_j^{tr}\right) = \begin{cases} 0.25 & \text{if} f_j\left(R_{\min}^{+}\right) < f_j^{tr}\left(rc\right) < P_{+}\left(f_i\right) \\ 0.5 & \text{if} f_j\left(R_{\min}^{+}\right) < f_j^{tr}\left(rc\right) < f_j\left(R_{\max}^{+}\right) \\ 1 & \text{if} f_j^{tr}\left(rc\right) > f_j\left(R_{\max}^{+}\right) \\ 0.125 & \text{if} f_j\left(S_{\min}^{+}\right) > f_j^{tr}\left(sc\right) > P_{+}\left(f_j\right) \\ 0.0625 & \text{if} f_j\left(S_{\min}^{+}\right) < f_i^{tr}\left(sc\right) < f\left(S_{\max}^{+}\right) \\ 0 & \text{if} f_j^{tr}\left(sc\right) > f_j\left(S_{\max}^{+}\right) \end{cases}$$

3.7.4. Predicted Label for the Test Record

Assign the label pl corresponding to the highest score among all labels in Eq. 9.

$$pl = \begin{cases} h & \text{if} \max\left(bass_h, bass_m, bass_l, bass_n\right) = bass_h \\ m & \text{if} \max\left(bass_h, bass_m, bass_l, bass_n\right) = bass_m \\ l & \text{if} \max\left(bass_h, bass_m, bass_l, bass_n\right) = bass_l \\ n & \text{if} \max\left(bass_h, bass_m, bass_l, bass_n\right) = bass_n \end{cases} \quad (9)$$

Validation of the Prediction:

If pl is the true label of record tr , then add tr to the truly predicted set tPR .

Return: Return the final set tPR of truly predicted records.

3.8. The Classification Process

Botnet attacks on fog computing networks exhibit sequential and temporal dependencies, making Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) [22] networks, suitable for their analysis. LSTM models can capture long-term dependencies and temporal patterns in sequential data, making them useful for detecting and classifying botnet attacks by severity. For multi-label classification where each input may belong to multiple severity categories, a Bidirectional LSTM (BiLSTM) [25] is preferred, as it processes input sequences in both forward and backward directions to retain contextual information.

Let's take a closer look at how a BiLSTM works. Like a standard LSTM, a BiLSTM consists of a sequence of LSTM cells, where each cell takes an input, updates its hidden state, and produces an output. The key difference is that a BiLSTM Each time step, the input sequence is processed in both the left-to-right and right-to-left directions, concatenating the outputs from the backward and forward LSTMs. The process flow of the BiLSTM is outlined in Figure 5.

Let the input sequence be represented as:

$$\mathbf{X} = \left\{x_1, x_2, \ldots, x_T\right\}$$

where $x_t \in \mathrm{R}^d$ is the feature vector at time step t , representing botnet-related traffic features. Let the output label matrix be:

$$\mathbf{Y} \in \{0,1\}^{N \times C}$$

where N is the number of samples and $C = 4$ denotes the number of severity classes.

The BiLSTM processes the input as:

$$\overleftarrow{h_t} = \mathrm{LSTM}_{\mathrm{fwd}}\left(x_t\right), \quad \overrightarrow{h_t} = \mathrm{LSTM}_{\mathrm{bwd}}\left(x_t\right)$$

$$h_t = \left[\overleftarrow{h_t}; \overrightarrow{h_t}\right]$$

The concatenated hidden states h_t are passed through a fully connected layer FC to produce an output vector $\mathbf{Z} \in \mathrm{R}^C$, followed by a softmax activation to obtain class probabilities:

$$\mathbf{Z} = \mathrm{FC}\left(h_T\right), \quad \mathbf{P} = \mathrm{Softmax}\left(\mathbf{Z}\right)$$

The model is trained using the binary cross-entropy loss:

$$\mathrm{L} = -\sum_{i=1}^{N}\sum_{j=1}^{C}\left[y_{ij}\log(p_{ij}) + (1-y_{ij})\log(1-p_{ij})\right]$$

Optimization is performed using the Adam optimizer with a learning rate $\alpha = 0.001$ and batch size $B = 32$. The model is evaluated using metrics such as accuracy, precision, recall, and F1-score on an unseen test set.

$$\mathbf{Y} = \mathrm{BiLSTM}\left(\mathbf{X}\right)$$

$$\mathbf{Z} = \mathrm{FC}\left(\mathbf{Y}\right)$$

$$\mathbf{P} = \mathrm{Softmax}\left(\mathbf{Z}\right)$$

$$\mathrm{L} = \mathrm{BinaryCrossEntropy}\left(\mathbf{Y}, \mathbf{P}\right)$$

$$\theta = \mathrm{AdamOptimizer}\left(\mathrm{L}\right)$$

$$\mathrm{Performance} = \mathrm{EvaluateModel}\left(\theta, \mathrm{TestSet}\right)$$

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

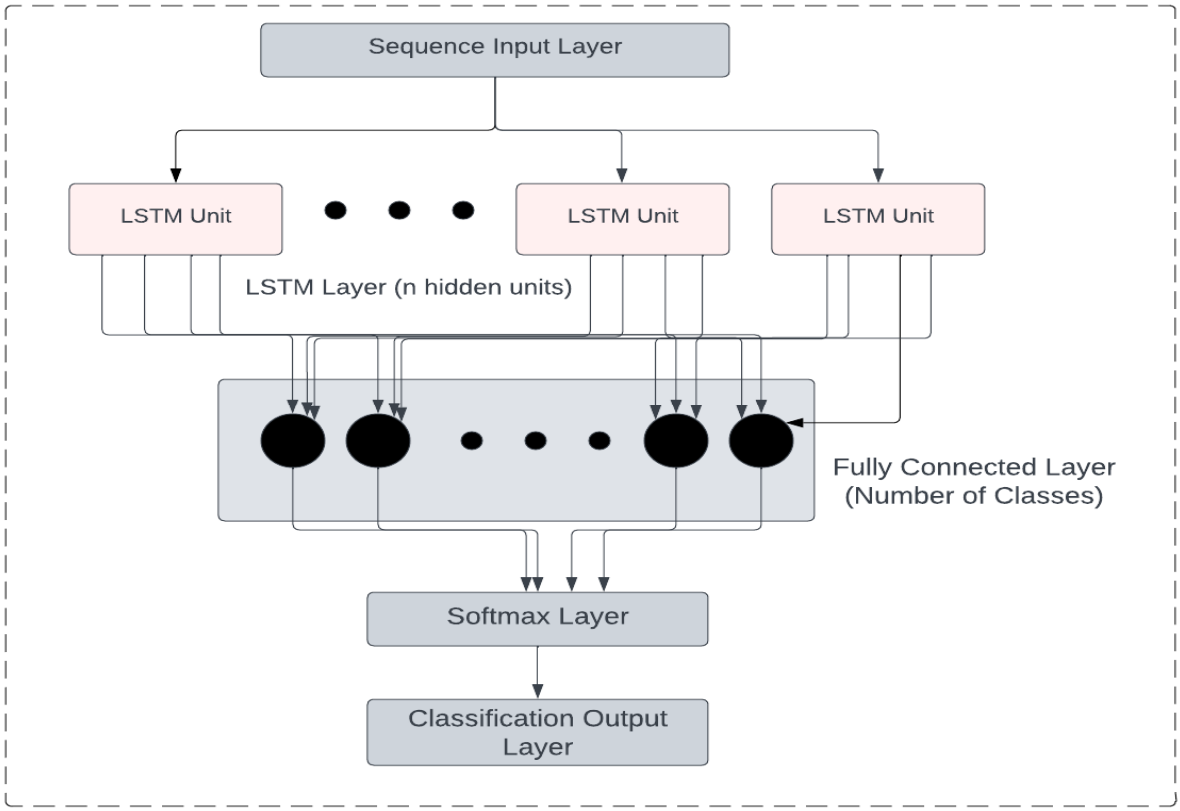

Figure.5 Process flow of the BiLSTM

The BiLSTM captures bidirectional context to enhance the identification of botnet attack severity. The classification output provides multi-label probabilities, helping in prioritizing and mitigating attacks in real-time fog environments

3.8.1 Forward LSTM Computation in BiLSTM

In a Bidirectional LSTM (BiLSTM), the forward LSTM processes the input sequence $X = \{x_1, x_2, \ldots, x_T\}$ from left to right. At each time step t, the LSTM updates its internal states using the following set of equations, where h_t is the hidden state and c_t is the cell state.

(a). **Input Gate ig_t :** Determines how much of the new input x_t should be added to the current cell state c_t. It is computed as in Eq. 10.

$$ig_t = \sigma\left(W_{ig} x_t + U_{ig} h_{t-1} + b_{ig}\right) \quad (10)$$

(b). **Forget Gate fg_t :** Controls the extent to which the previous cell state c_{t-1} should be retained as in Eq. 11.

$$fg_t = \sigma\left(W_{fg} x_t + U_{fg} h_{t-1} + b_{fg}\right) \quad (11)$$

(c). **Output Gate og_t :** Decides how much of the current cell state contributes to the output hidden state through Eq. 12.

$$og_t = \sigma\left(W_{og} x_t + U_{og} h_{t-1} + b_{og}\right) \quad (12)$$

(d). **Input Modulation Gate img_t :** Generates candidate values for updating the cell state, scaled between -1 and 1, as in Eq. 13.

$$img_t = \tanh\left(W_{img} x_t + U_{img} h_{t-1} + b_{img}\right) \quad (13)$$

(e). **Cell State Update c_t :** Combines the previous cell state and new modulated input using element-wise operations, as in Eq. 14.

$$c_t = fg_t \odot c_{t-1} + ig_t \odot img_t \quad (14)$$

Each gate uses the current input x_t, the previous hidden state h_{t-1}, corresponding weight matrices W_*, U_*, and bias vectors b_*.

The sigmoid function σ and the hyperbolic tangent $\tanh$ regulate and modulate the information flow, ensuring stable learning of long-term dependencies.

3.8.2 Backward LSTM

In BiLSTM, the backward LSTM processes the input

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

sequence $X=\{x_1,x_2,...,x_T\}$ from x_T to x_1. At each time step t, it computes as in Eq. 15.

Input Gate ig_t^{bl} :

$$ig_t^{bl}=\sigma\left(W_{ig}^{bl}x_t^{bl}+U_{ig}^{bl}h_{t+1}^{bl}+b_{ig}^{bl}\right) \quad (15)$$

Forget Gate fg_t^{bl} : as in Eq. 16.

$$fg_t^{bl}=\sigma\left(W_{fg}^{bl}x_t^{bl}+U_{fg}^{bl}h_{t+1}^{bl}+b_{fg}^{bl}\right) \quad (16)$$

Output Gate og_t^{bl} : as in Eq. 17.

$$og_t^{bl}=\sigma\left(W_{og}^{bl}x_t^{bl}+U_{og}^{bl}h_{t+1}^{bl}+b_{og}^{bl}\right) \quad (17)$$

Input Modulation Gate img_t^{bl} : as in Eq. 18.

$$img_t^{bl}=\tanh\left(W_{img}^{bl}x_t^{bl}+U_{img}^{bl}h_{t+1}^{bl}+b_{img}^{bl}\right) \quad (18)$$

Cell State cs_t^{bl} : as in Eq. 19.

$$cs_t^{bl}-fg_t^{bl}\odot cs_{t+1}^{bl}+ig_t^{bl}\odot img_t^{bl} \quad (19)$$

Hidden State h_t^{bl} : as in Eq. 20.

$$h_t^{bl}=og_t^{bl}\ \tanh\left(cs_t^{bl}\right) \quad (20)$$

The BiLSTM output at each time step t is:

$$Y_t=\left[h_t;h_t^{bl}\right]$$

All outputs Y_t are collected into final output sequence Y.

Fully Connected Layer $FC(Y)$:

Given BiLSTM output $Y=[y_1,y_2,...,y_m]$, each element z_j of the FC output vector is shown in Eq. 21.

$$z_j=\sigma\left(W_j^T y_m+b_j\right) \quad (21)$$

Where σ is the activation function, W_j is the weight vector, and b_j is the bias.

Softmax Function:

Transforms the FC output $Z=[z_1,z_2,...,z_n]$ into probabilities through Eq. 22.

$$p_j=\frac{e^{z_j}}{\sum_{i=1}^{n}e^{z_i}} \quad (22)$$

This ensures $(\sum_j p_j=1)$, making it suitable for multi-class probability distribution.

Binary Cross-Entropy Loss:

For ground truth $Y=[y_1,...,y_m]$ and predictions $P=[p_1,...,p_m]$, the loss is shown in Eq. 23.

$$L=-\sum_{i=1}^{m}\sum_{j=1}^{n}\left(y_{ij}\ \log\left(p_{ij}\right)+\left(1-y_{ij}\right)\log\left(1-p_{ij}\right)\right) \quad (23)$$

Adam Optimizer:

The parameters θ are updated using:

Moment Estimates:

$$m_t=\beta_1 m_{t-1}+\left(1-\beta_1\right)g_t,\quad v_t=\beta_2 v_{t-1}+\left(1-\beta_2\right)g_t^2$$

Bias Correction:

$$\hat{m}_t=\frac{m_t}{1-\beta_1^t},\quad \hat{v}_t=\frac{v_t}{1-\beta_2^t}$$

Parameter Update:

$$\theta_t=\theta_{t-1}-\alpha\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\grave{o}}$$

Where α is the learning rate, and $\grave{o}$ is a small constant to prevent division by zero. This end-to-end mechanism enables effective training of BiLSTM for multi-label classification in botnet severity prediction tasks.

3.9. Experimental Setup

Python was used to conduct the experiments, and the code was created using the Python editor PyCharm. External Python libraries, such as pypcapkit and scikit-learn, were investigated for processing pcap files of the input corpus and performing 4-fold cross-validation in respective order. Regarding this, the hardware requirements include a 7th-generation Intel processor-based CPU with the Windows 10 operating system, 16 GB of RAM, and a 2TB hard drive.

3.9.1. Dataset

The data used in this study were derived from the traffic sessions of the eight Internet of Things devices [31]. Eight IoT devices' pcap files were analysed to ascertain the observed sessions. The devices logged 48320, 38169, 174025, 60950, 56845, 7588, 45410, and 18480 sessions, respectively. Each data session's multivariable sequential representation includes twenty features that depict botnet and traffic flow characteristics. The total number of sessions is 449786. Based on their observed impact, specific sessions out of 3,534,11 have been categorised as high, medium, low, or not an attack. This categorization was the result of the annotation procedure. Table 3 demonstrates the cross-validation input statistics. The records are organised by device ID and class label; for each device ID,

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

the total and number of records for each class label are provided. The efficacy of the proposed model is evaluated using the dataset and 4-fold cross-validation.

4. Results and Discussions

4.1. Comparative Study

The objective of the empirical investigation was to assess the performance of BiLSTM in multiclass classification using four class labels: high, moderate, low, and normal. Fourfold cross-validation was used to ensure the accuracy of the experiment's results. The BiLSTM model was trained using three distinct optimizers: ADAM, ADAGRAD, and SGD. BPTT, a well-known algorithm for training RNN models, was utilised as the backpropagation technique. Two regularisation procedures, L1 and L2, were employed during training to prevent overfitting. Hyperparameter adjustment improved model performance. To find the best hyperparameters. Precision, sensitivity/recall, F-score, specificity, and accuracy assessed the BiLSTM model's performance. Researchers could compare BiLSTM models trained with different optimizers and regularization techniques based on experimental results. Hyperparameter optimisation helps find the best hyperparameters. The study's experimental design was carefully crafted to ensure reliability and validity.These classifications represent the severity of botnet attacks on networks utilising fog computing. For each fold, precision values are calculated separately for each class. The decision accuracy values (see Figure 6 (a)) representing the overall accuracy of the algorithm were highest for ADAM across all four phases, followed by ADAGRAD. SGD had the lowest decision accuracy values in all four folds and categories. In terms of sensitivity/recall (see Figure 6 (b)), all three algorithms achieved values in the range of 0.960-0.990 for all four folds and categories. In the high category, however, ADAM and ADAGRAD consistently outperformed SGD, achieving values of 0.990-0.996 versus 0.960-0.980 for SGD.

Observing the precision values (see Figure 6 (c)), we can see that ADAM consistently outperformed the other two algorithms, particularly in the high and low categories. ADAM achieved precision values of 0.995-0.996 in the high category and 0.992-0.996 in the low category, whereas ADAGRAD and SGD achieved 0.960-0.994 and 0.820-0.995, respectively, in the same categories.Specificity values (see Figure 6 (d)) were comparable for all three algorithms, ranging from 0.961 to 0.987. In some instances, ADAM and ADAGRAD attained marginally higher specificity values than SGD.

F-score values (see Figure 6 (e)) were also generally greater for ADAM and ADAGRAD than for SGD, with ADAM attaining the highest F-score values in the majority of categories and folds.

The statistics indicate that ADAM and ADAGRAD outperform SGD for precision, F-score, and decision accuracy. All three algorithms performed identically regarding sensitivity/recall and specificity, whereas ADAM consistently demonstrated superior performance in the high and low categories. Table 3 displays the mean and standard deviation of various cross-validation metrics for the ADAM, ADAGRAD, and SGD optimisation algorithms.

Table.3 Mean and Deviation of Various Cross-Validation Metrics for ADAM, ADAGRAD, and SGD

Metric	Model	Average	Deviation
Precision	ADAM	0.9685	0.0036
	SGD	0.9449	0.0072
	ADAGRAD	0.9609	0.0019
Sensitivity	ADAM	0.9825	0.0018
	SGD	0.9656	0.0045
	ADAGRAD	0.9756	0.0011
F-Score	ADAM	0.9751	0.0041
	SGD	0.9546	0.0046
	ADAGRAD	0.9678	0.0058
Decision Accuracy	ADAM	0.9825	0.0018
	SGD	0.9646	0.0031
	ADAGRAD	0.976	0.0011
Specificity	ADAM	0.9481	0.0017
	SGD	0.88	0.0041
	ADAGRAD	0.9046	0.0005

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

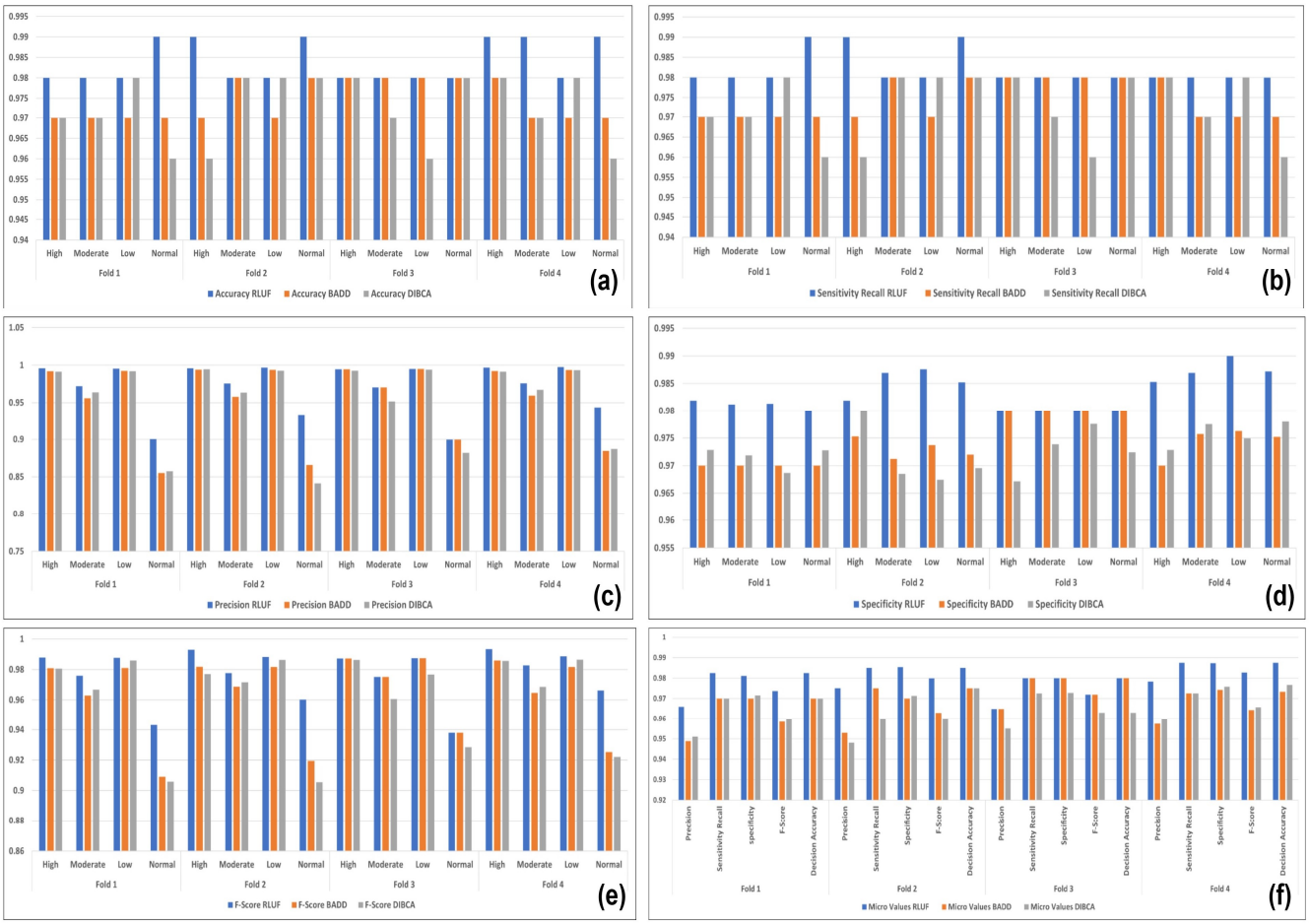

Figure.6 The bar plots in Figure (a) through (f) present a comparative analysis of the performance metrics—Accuracy, Sensitivity, Precision, Specificity, F-Score, and various P-values—achieved by three feature selection methods (RLRF, BADD, and DMCA) across four distinct folds of the dataset. Each subplot illustrates the performance of these methods under different data complexity levels: High, Moderate, Low, and Normal. The results are presented fold-wise to assess the consistency and robustness of each feature selection technique across different data partitions.

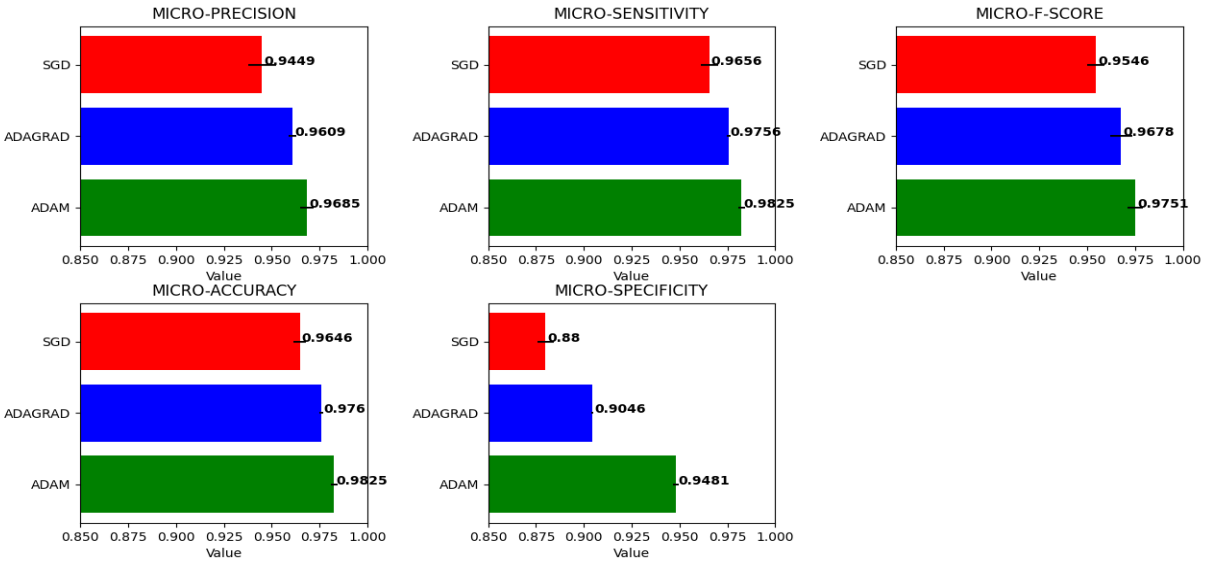

Figure.7 Comparative Analysis of Micro Metrics for Model Performance

Figure 7 shows the micro values of five metrics. These include precision, recall, F-score, accuracy, and specificity. Each metric measures a unique facet of the model's effectiveness. In Table 3, values for each model are given. For example, ADAM shows a micro precision of 0.9685 with a standard deviation of 0.0036. The ADAM optimizer

ISSN: 2582 - 4201(Online), http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 6, 2025 , https://doi.org/10.63328/IJRDES-V7RI6P1

shows a micro recall (sensitivity) of 0.9825 with a deviation of 0.0018. Specificity means how well the negatives are predicted. It tells the correct detection of negative observations. ADAM shows a micro specificity of 0.9481 with a deviation of 0.0017. F-score gives a balance between precision and recall. It is the harmonic mean of both. ADAM shows a micro F-score of 0.9751 with a deviation of 0.0041. Accuracy tells how many total predictions are correct. ADAM shows a micro decision accuracy of 0.9825 with a deviation of 0.0018. The micro values shown in Table 3 combine the results across all severity labels high, medium, low and normal. In this part, ADAM outperforms the other optimizers and proves the most effective with Bi-LSTM model in botnet prediction. ADAGRAD offers reasonable stable results but SGD lags behind.

4.2. Attack Detection Time

The simulation model is intended to generate botnet attacks on a decentralised fog computing network. The network consists of 240 IoT devices and 8 fog nodes, each with a processing capacity of 1.5 GHz. Each set of 30 IoT devices is connected to a single fog node. The simulation uses HTTP as the botnet payload type, and the propagation range is set to 200 metres. The simulation is intended to operate for 60 minutes. The inventory of simulation model parameters was presented in Table 4.

The simulation begins with a single compromised node which starts the spread of botnet in the fog environment. Using the HTTP payload, the infection propagates to other nodes within its range. This simulation's propagation parameter is set to 0.3, means that 30% of the nodes within the botnet's range become infected at each time step. Once a node is compromised, it executs its assigned commands. The attack intensity parameter is set to 0.7 of the assault will be determined by the attack parameter, which in this simulation is set to 0.7, indicates that the 70% of each nodes processing and memory capabilities are used to launch attacks.

The simulation model also measures various metrics, includes number of infected nodes, the number of attacks launched, and the network throughput. These metrics helps to evaluate, the efficacy of various countermeasures against botnet attacks on fog computing networks.

Table.4 Parameters Used in the Simulation Model to Generate Botnet Attacks on a Fog Computing Network Using a Decentralized Topology

Parameter	Description	Ideal Value
Network Topology	Decentralized topology for a fog computing network	Decentralized
Number of Fog Nodes	Number of fog nodes in the network	8
Number of Devices	Number of devices in the network	240
Botnet Payload	Type of payload used for the attack	HTTP Flood
Payload Strength	Strength of the botnet payload in terms of request rate	100 to 1000 requests per second
Propagation Range	Range of the botnet propagation within the network	8 fog nodes
Attack Duration	Duration of the botnet attack	1 hour

As illustrated in Figure 8, the detection time increases with the payload ratefor all three optimization algorithms (ADAM, ADAGRAD, and SGD) but the rate of increase varies across them. At payload of 100 requests per second, ADAM detects the attack in 0.03 seconds, ADAGRAD in 0.043 seconds, and SGD requires 0.026 seconds. As the payload per second increases, the detection time increases for all three optimizers, ADAM consistently maintain lower and stable latency compares to ADAGRAD, but SGD is slightly faster but less consistent. On average the mean detection times for ADAM, ADAGRAD and SGD 0.036 seconds, 0.049 seconds, and 0.033 seconds, respectively. The standard deviation values 0.0033 seconds, 0.0054 seconds, and 0.0086 indicates that ADAM offers most stable performance and SGD shows greater fluctuation across varying payload levels.

4.3. Statistical Analysis

Statistical analyses showed apparent differences between DIBCA, RLUF, and BADD models. Values for precision and other measures were checked. The two main tests used were ANOVA and descriptive statistics. The values of descriptive statistics in Table 5 are shown. The table shows deviations, medians, and means for all severity levels. For RLUF, the precision average of high severity was about 0.987, while around 0.980 was given by BADD and DIBCA. Precision for moderate severity was around 0.980 in RLUF, while slightly lower, near 0.970, in both DIBCA and BADD. Around 0.986 precision was shown for RLUF in low severity, with slightly lower values at around 0.979 for DIBCA and BADD.

ISSN: 2582 - 4201(Online) , http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 2, 2025 , https://doi.org/10.63328/IJRDES-V7RI2P4

Normal severity had larger variations, where around 0.965 precision was recorded in RLUF, and the lowest was around 0.939 in DIBCA.

Depending on severity, these values suggest model differences, yet practical scenarios might vary in these differences.

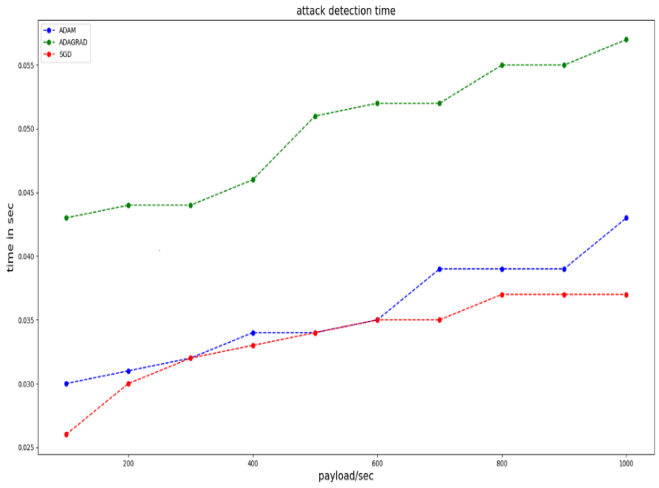

Figure.8 Payload versus detection time in seconds

Table.5 Parameters Used in the Simulation Model to Generate Botnet Attacks on a Fog Computing Network Using a Decentralized Topology

Severity Level	Model	Mean	Std Dev
High	RLUF	0.987	0.006
	BADD	0.980	0.009
	DIBCA	0.979	0.010
Moderate	RLUF	0.980	0.005
	BADD	0.971	0.008
	DIBCA	0.969	0.006
Low	RLUF	0.986	0.006
	BADD	0.979	0.009
	DIBCA	0.980	0.010
Normal	RLUF	0.965	0.029
	BADD	0.945	0.042
	DIBCA	0.939	0.045

ANOVA analysis was used to statistically confirm differences in model performance. Differences significant in High severity (F=5.75, p=0.005) and very substantial for Moderate severity (F=15.13, p<0.0001) were found.

Differences for Low severity were also substantial (F=3.44, p=0.039), but no significant difference was shown in Table 6 for Normal severity (F=2.47, p=0.094).

Tests showed apparent differences in precision and sensitivity, with p-values around 0.007 and 0.004, respectively.

Decision accuracy and F-score differences were also significant, with about 0.011 and 0.003 p-values. Differences were shown in specificity too, having a p-value of 0.026, suggesting that, depending on the case examined, differences possibly exist.

The box plot clearly shows Figure 9 (a) precision variations among different models visibly different across severity levels are models RLUF, BADD, and DIBCA.

More minor variations among these models are generally displayed in the High and Moderate severity groups, while greater differences are shown in the

Normal severity group. The complexity of dataset gives slight inconsistency in precision for normal severity group.

These observed trends are match statistical tests from ANOVA. The bar graph shown in Figure 9 (b) compares the mean precision across the severity levels, showing the RLUF model obtains slight higher averages in the in High, Moderate, and Low severity groups compared to BADD and DIBCA.

The difference between High and Low groups are minor yet visible, while larger variations appear in Normal severity.

Overall, RLUF generally showed stronger higher precision in severe cases. Further research might focus on real-world network checking of these models to better understand their practical usefulness in cases.

Table.6 Statistical Analysis of Metrics

Severity / Metric	F-Statistic	p-value	Significance (p < 0.05)
High	5.75	0.005	Yes
Moderate	15.13	<0.0001	Yes
Low	3.44	0.039	Yes
Normal	2.47	0.094	No
Precision	9.25	0.007	Yes
Sensitivity Recall	10.78	0.004	Yes
F-Score	11.83	0.003	Yes

ISSN: 2582 - 4201(Online) , http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 2, 2025 , https://doi.org/10.63328/IJRDES-V7RI2P4

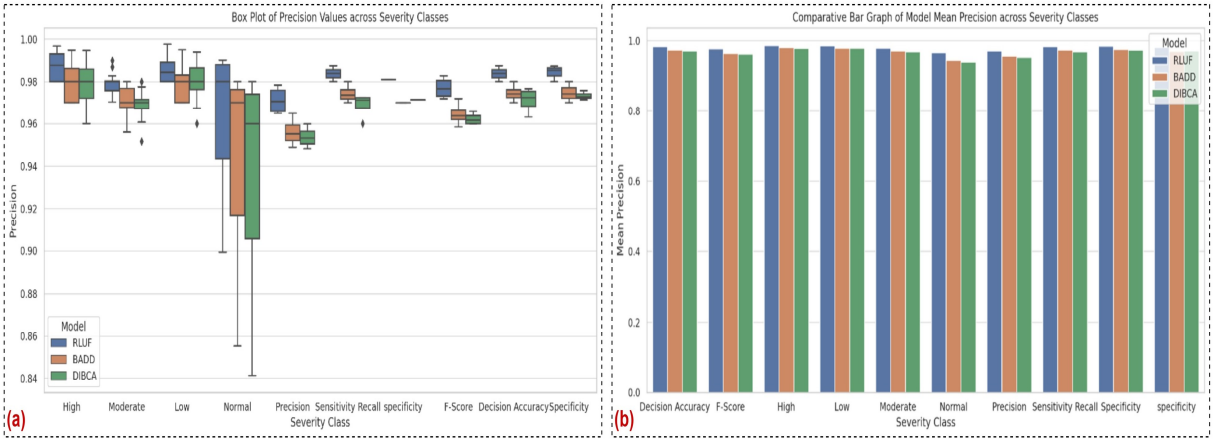

Figure.9 (a) shows the box Plot of Precision Values across Severity Classes` and (b) shows comparative Bar Graph of Model Performance Metrics

5. Conclusion

This study explored botnet predictionin fog computing environments using Bi-LSTM model optimized with ADAM, ADAGRAD and SGD includes Bi-LSTM multi-class classification strategy with reinforced feature optimisation. The results indicate that the proposed method outperforms with ADAM optimizer with its competitors in terms of all performance metrics. ADAGRD performed with slight lower accuracy while SGD showed faster but at lower consistency. The reinforcement feature optimization enables the model continuously refine the parameters based on the feedback, allowing to respond efficently to changing network behaviours. Simulation shows this strategy reduces detection time and improve resilence under different payload conditions. Overall, the proposed Bi-LSTM RL approach demonstrates strong performance for intelligent and self optimizing botnet attack detection in fog based IoT environments. Future work may focus on real world deployment and integration of hybrid meta learning to enhance the robustness and scalability.

References

[1]. Y. Labiod, A. Amara Korba, and N. Ghoualmi, “Fog computing-based intrusion detection architecture to protect iot networks,” Wireless Personal Communications, vol. 125, no. 1, pp. 231–259, 2022.

[2]. P. Kumar, G. P. Gupta, and R. Tripathi, “Design of anomaly-based intrusion detection system using fog computing for iot network,” Automatic Control and Computer Sciences, vol. 55, no. 2, pp. 137–147, 2021.

[3]. S. Miller and C. Busby-Earle, “The role of machine learning in botnet detection,” in 2016 11th international conference for internet technology and secured transactions (icitst). 1em plus 0.5em minus 0.4em IEEE, 2016, pp. 359–364.

[4]. S. Sriram, R. Vinayakumar, M. Alazab, and K. Soman, “Network flow based iot botnet attack detection using deep learning,” in IEEE INFOCOM 2020-IEEE conference on computer communications workshops (INFOCOM WKSHPS). 1em plus 0.5em minus 0.4em IEEE, 2020, pp. 189–194.

[5]. Y. Xing, H. Shu, H. Zhao, D. Li, and L. Guo, “Survey on botnet detection techniques: Classification, methods, and evaluation,” Mathematical Problems in Engineering, vol. 2021, no. 1, p. 6640499, 2021.

[6]. S. Gaonkar, N. F. Dessai, J. Costa, A. Borkar, S. Aswale, and P. Shetgaonkar, “A survey on botnet detection techniques,” in 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE). 1em plus 0.5em minus 0.4em IEEE, 2020, pp. 1–6.

[7]. H. Sabireen and V. Neelanarayanan, “A review on fog computing: Architecture, fog with iot, algorithms and research challenges,” Ict Express, vol. 7, no. 2, pp. 162–176, 2021.

[8]. B. Costa, J. Bachiega Jr, L. R. De Carvalho, and A. P. Araujo, “Orchestration in fog computing: A comprehensive survey,” ACM Computing Surveys (CSUR), vol. 55, no. 2, pp. 1–34, 2022.

[9]. Y. Yang, W. Wang, H. Fu, C.-C. J. Kuo et al., “On supervised feature selection from high dimensional feature spaces,” APSIPA Transactions on Signal and Information Processing, vol. 11, no. 1, 2022.

[10]. M. Gopinath and S. C. Sethuraman, “A comprehensive survey on deep learning based malware detection techniques,” Computer Science Review, vol. 47, p. 100529, 2023.

[11]. R. Rawat, R. K. Chakrawarti, P. Vyas, J. L. A. Gonzáles, R. Sikarwar, and R. Bhardwaj, “Intelligent fog computing surveillance system for crime and vulnerability identification and tracing,” International Journal of Information Security and Privacy (IJISP), vol. 17, no. 1, pp. 1–25, 2023.

[12]. N. F. Syed, M. Ge, and Z. Baig, “Fog-cloud based intrusion detection system using recurrent neural networks and feature selection for iot networks,” Computer Networks, vol.

ISSN: 2582 - 4201(Online) , http://www.ijrdes.com/ , IJRDES, Vol. 7, Issue 2, 2025 , https://doi.org/10.63328/IJRDES-V7RI2P4

225, p. 109662, 2023.

[13]. S. Balasubramaniam, C. Vijesh Joe, T. Sivakumar, A. Prasanth, K. Satheesh Kumar, V. Kavitha, and R. K. Dhanaraj, "Optimization enabled deep learning-based ddos attack detection in cloud computing," International Journal of Intelligent Systems, vol. 2023, no. 1, p. 2039217, 2023.

[14]. H. Nandanwar and R. Katarya, "Deep learning enabled intrusion detection system for industrial iot environment," Expert Systems with Applications, vol. 249, p. 123808, 2024.

[15]. M. Ali, M. Shahroz, M. F. Mushtaq, S. Alfarhood, M. Safran, and I. Ashraf, "Hybrid machine learning model for efficient botnet attack detection in iot environment," IEEE Access, 2024.

[16]. Y. Li, S. Zhang, Y. Chang, G. Xu, and H. Li, "Privacy-preserving and poisoning-defending federated learning in fog computing," IEEE Internet of Things Journal, vol. 11, no. 3, pp. 5063–5077, 2023.

[17]. O. Ibitoye, O. Shafiq, and A. Matrawy, "Analyzing adversarial attacks against deep learning for intrusion detection in iot networks," in 2019 IEEE global communications conference (GLOBECOM). 1em plus 0.5em minus 0.4em IEEE, 2019, pp. 1–6.

[18]. M. Ge, X. Fu, N. Syed, Z. Baig, G. Teo, and A. Robles-Kelly, "Deep learning-based intrusion detection for iot networks," in 2019 IEEE 24th pacific rim international symposium on dependable computing (PRDC). 1em plus 0.5em minus 0.4em IEEE, 2019, pp. 256–25 609.

[19]. M. A. Ferrag and L. Maglaras, "Deepcoin: A novel deep learning and blockchain-based energy exchange framework for smart grids," IEEE Transactions on Engineering Management, vol. 67, no. 4, pp. 1285–1297, 2019.

[20]. B. Susilo and R. F. Sari, "Intrusion detection in iot networks using deep learning algorithm," Information, vol. 11, no. 5, p. 279, 2020.

[21]. G. Muhammad, M. S. Hossain, and S. Garg, "Stacked autoencoder-based intrusion detection system to combat financial fraudulent," IEEE Internet of Things Journal, vol. 10, no. 3, pp. 2071–2078, 2020.

[22]. X. Zhou, W. Liang, W. Li, K. Yan, S. Shimizu, and K. I.-K. Wang, "Hierarchical adversarial attacks against graph-neural-network-based iot network intrusion detection system," IEEE Internet of Things Journal, vol. 9, no. 12, pp. 9310–9319, 2021.

[23]. M. Mudassir, D. Unal, M. Hammoudeh, and F. Azzedin, "Detection of botnet attacks against industrial iot systems by multilayer deep learning approaches," Wireless Communications and Mobile Computing, vol. 2022, no. 1, p. 2845446, 2022.

[24]. S. Popoola, B. Adebisi, G. Gui, M. Hammoudeh, H. Gacanin, and D. Dancey, "Optimizing deep learning model hyperparameters for botnet attack detection in iot networks," Authorea Preprints, 2022.

[25]. M. Catillo, A. Pecchia, and U. Villano, "A deep learning method for lightweight and cross-device iot botnet detection," Applied Sciences, vol. 13, no. 2, p. 837, 2023.

[26]. G. Kirubavathi, U. Sridevi et al., "Detection of iot botnet using machine learning and deep learning techniques," 2023.

[27]. N. Elsayed, Z. ElSayed, and M. Bayoumi, "Iot botnet detection using an economic deep learning model," in 2023 IEEE World AI IoT Congress (AIIoT). 1em plus 0.5em minus 0.4em IEEE, 2023, pp. 0134–0142.

[28]. S. Siami-Namini, N. Tavakoli, and A. S. Namin, "The performance of lstm and bilstm in forecasting time series," in 2019 IEEE International conference on big data (Big Data). 1em plus 0.5em minus 0.4em IEEE, 2019, pp. 3285–3292.

[29]. F. Hussain, S. G. Abbas, U. U. Fayyaz, G. A. Shah, A. Toqeer, and A. Ali, "Towards a universal features set for iot botnet attacks detection," in 2020 IEEE 23rd international multitopic conference (INMIC). 1em plus 0.5em minus 0.4em IEEE, 2020, pp. 1–6.

[30]. Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiotâ€"network-based detection of iot botnet attacks using deep autoencoders," IEEE Pervasive Computing, vol. 17, no. 3, pp. 12–22, 2018.

Author Contribution

SPKG: Conceptualization, drafting, data collection and analysis, Methodology, and Supervision;

PN: Methodology, Writing and Data Collection, and Editing

RY: Interpretation and Supervision;

TC: Data Collection and Interpretation;

Declaration

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

Author Contribution: All authors wrote the main manuscript text and also consent to the submission.

Ethical approval: Not applicable.

Consent to Participate: All authors consent to participate.

Funding: Not applicable, and No funding was received

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Personal Statement: We declare with our best of knowledge that this research work is purely Original Work and No third party material used in this article drafting. If any such kind material found in further online publication, we are responsible only for any judicial and copyright issues.

Acknowledgements

We thank everyone who inspired our work.