Enhancing Network Security Intrusion Detection and Real-Time Response With Long Short-Term Memory Networks

[1]Rukmani Devi S, [2]Manju A, [3]Lakshmi T K, [4]Venkataramanaiah B, [5]Sureshkumar Chandrasekaran and [6]Lakshmi Prasanna
[1]Department of Computer Science, Saveetha College of Liberal Arts and Sciences, SIMATS Deemed to be University, Chennai, Tamil Nadu, India.
[2]Department of Computer Science and Engineering, SRM Institute of Science and Technology, Ramapuram, Chennai, Tamil Nadu, India.
[3]Department of Computer Science and Engineering, Malla Reddy University, Hyderabad, Telangana, India.
[4]Department of Electronics and Communication Engineering, Vel Tech Rangarajan Dr. Sagunthala R & D Institute of Science and Technology, Avadi, Chennai, Tamil Nadu, India.
[5]Department of Artificial Intelligence and Data Science, KCG College of Technology, Chennai, Tamil Nadu, India.
[6]Department of Computer Science and Engineering Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh, India.
[1]rukmanibaveshnambi@gmail.com, [2]manjuappukuttan1985@gmail.com, [3]dr.lakshmitk@gmail.com, [4]bvenkataramanaiah@veltech.edu.in, [5]csureshkumarmca@gmail.com, [6]lakshmiprasannap87@gmail.com

Correspondence should be addressed to Rukmani Devi S : rukmanibaveshnambi@gmail.com

Article Info
Journal of Machine and Computing (https://anapub.co.ke/journals/jmc/jmc.html)
Doi : https://doi.org/10.53759/7669/jmc202505079
Received 11 October 2024; Revised from 26 January 2025; Accepted 02 March 2025.
Available online 05 April 2025.

Abstract – In cybersecurity, network security systems (NSS) are technologies used to protect sensitive data against increasing cyber-attacks. This paper has carried out the process of integrating advanced Machine Learning (ML) techniques such as the Long Short-Term Memory (LSTM) networks along with the Convolutional Neural Networks (CNN) for the task of enhancing the Intrusion Detection Systems (IDS) in the NSS. Usually, the traditional models have faced many challenges, such as high false positive rates (FPR) and the need for real-time processing of extensive data streams, which make these systems insufficient to handle such scenarios. So, to handle these complications, ML has evolved to propose improvements in IDS implementation by adapting to new attacks and detecting complex patterns with greater accuracy. The study introduced a novel deep learning (DL), "GC-SLSTM," that combined models such as Gated CNN with Stacked LSTM to address these challenges. This model includes CNN robust spatial pattern recognition and the LSTM that effectively handles the temporal data analysis. The proposed model was experimented with using the CICIDS2018 dataset, and the results demonstrate that the proposed Gated CNN + Stacked LSTM (GC-SLSTM) had achieved an accuracy of up to 99.59%, precision of 99.58% and a recall of 99.47%, culminating in an F1-score of 99.59%.

Keywords – Network Security Systems, Intrusion Detection Systems, CNN, LSTM, False Positive Rates, Machine Learning.

I. INTRODUCTION

Network Security Systems (NSS) are crucial in ensuring business integrity and security in the digital age. They protect sensitive data from unauthorized access, breaches, and cyber-attacks. As cyber-attacks become more complex and frequent, the need for enhanced network security protocols has increased. Intrusion Detection Systems (IDS) are used to identify potential attacks by monitoring network traffic. However, IDS faces challenges such as high False Positive Rates (FPR), new and unknown attacks, and computational demands for real-time analysis. The increasing complexity of network models and attacking vectors by cyber criminals complicates the development of effective IDS [1-2].

Machine learning (ML) has significantly improved predictive analytics for network security, replacing traditional rule-based approaches that maintain hope in the face of dynamic cyber-attacks. ML's adaptive learning capabilities are crucial for managing new and evolving attacks. As research on ML, particularly in Deep Learning and Neural Networks, advances, sophisticated IDS models have become more accurate and timelier. However, these systems require large datasets for training, which require substantial computational resources, which are critical in a security context [3]. Long-short memory (LSTM) is a Recurrent Neural Network (RNN) type designed to handle time-series data. Its model can remember past data for an extended period, enabling it to identify patterns and predict future events. Applying LSTM to IDS can better understand network traffic flow and anomalies over time. Unlike traditional models that process each data point independently, LSTM considers traffic flow as a sequence, detecting complex attack patterns over an extended period. This reduces the FP and increases attack detection accuracy in real-time [4-5].

Convolutional Neural Networks (CNN) are used in IDS to classify network intrusion patterns. They can process and analyze spatial relationships within data, IDS, and relevant network traffic patterns. Integrating CNN + LSTM can provide a more reliable IDS. This hybrid model uses CNN's spatial pattern recognition capability and LSTM's sequence prediction capabilities [6-10]. CNN analyzes data to detect spatial anomalies, while LSTM processes output over time to understand temporal patterns and anomalies. This integration allows for more effective multi-stage cyber-attack detection and higher chances of reducing the frequency of attacks (FP).

This work proposed a Deep Learning (DL) Advanced Intrusion Detection and Real-Time Response in NSS. The work combines the Gated CNN (GCNN) with Stacked LSTM (S-LSTM) networks for network IDS. The method effectively preprocesses network traffic by segmenting the input data employing time and type of attack. Then, the segmented data is serialized and converted into grayscale images fed as CNN input. The proposed GC-SLSTM processes the preprocessed data by utilizing the G-CNN to filter the essential features that S-LSTM processes to analyze the temporal dependencies. The GC-LSTM experimented with using the CICIDS2018, and it showed better performance than existing models.

The paper is structured as follows: Section 2 presents the literature review, Section 3 provides the methods used in this work, Section 4 presents the proposed IDS, Section 5 examines the performance of the work, and Section 6 concludes the work.

II. LITERATURE REVIEW

Authors [9] invented a Deep Neural Network (DNN) using 28 features from the NSL-KDDt. It included a real-time feature extraction (FE) into an ML pipeline. Their proposed DNN has demonstrated performance with accuracy, precision, recall, and F1-score metrics at 81%, 96%, 70%, and 81%, respectively. Authors have designed a Conditional Deep Belief Network (CDBN)-based IDS for handling data imbalance and redundancy by using a window-based instance selection algorithm, "SamSelect," and by including a Stacked Contractive Encoder (SCAE) for dimension reduction. Their system had shown a detection accuracy rate of 97.4% with an achieved detection time mean of 1.14 *ms*.

The authors have applied machine learning (ML) [10] for IDS, using Signature IDS (S-IDS) and Anomaly IDS (A-IDS) on datasets like KKDDCUP99 and NLS-KDD. They used SVM, Naïve Bayes, and ANN, and their method performed better in real-time networks. They also explored hierarchically distributed IDS for Cyber-physical-based Industrial Systems using the Kalman Filter (KF) and a recursive Gaussian mixture model. Their method efficiently recognized potential and covert cyber-attacks across ICPS links, as demonstrated by several experiments.

The authors have developed an Artificial Intelligence System (A-IDS) [11] based on the human Immune System features, incorporating innate and adaptive layers. They used statistical and adaptive Immune models to mimic the immune system's response mechanisms. The system achieved high True Positive Rates (TPR) and effective IDS. The authors also investigated the integration of network profiling, ML, and game theory to secure IoT environments against cyber-attacks. Their A-IDS dynamically profiles and monitors IoT devices, identifying suspicious transactions. Tested on the Cyber-Trust testbed, the model achieved a high overall accuracy of 98.35% and a low FRP of 0.98%.

The authors [12] developed an LSTM-based IDS to detect attacks on vehicles' Controller Area Network (CAN) bus networks. They generated a unique dataset using attack simulations on an experimental car and trained and tested their model. The system demonstrated a 99.9% detection accuracy. They designed a novel II-stage DL that combined LSTM with Auto-Encoders (AE) and used their model for attack detection. The model performed better in CICIDS2017 and CSE-CIC-IDS2018 datasets.

The authors have developed a Network Intrusion Detection System (NIDS) using a Recurrent Neural Network (RNN) [13]. The system integrates multiple modules, including a management center, knowledge database, data acquisition, risk analysis, BiLSTM + DNN for sequential data relevance, and FE. An attention mechanism enhances the importance of features for NN efficiency. The authors also proposed a distributed DL-IDS using Apache Spark to tackle challenges related to the Internet of Vehicles (IoV) under 5G. The model achieved fast IDS speeds and a high accuracy of 99.7%, proving superiority over existing models. They also developed a novel IDS to detect botnet activities by analyzing the input flow of network node features using RL2TM. This model improves network efficiency and eliminates redundant activities [14-15].

III. METHODS

Convolution Neural Network

A CNN is a type of ANN designed to process data with a grid-like topology; such data usually encompass images or videos [16]. These datasets hold and exhibit complex patterns effectively processed and analyzed by CNN. The model of a CNN **Fig 1** includes multiple layers, such as an input layer, one or more convolutional layers, pooling layers, and fully connected layers at the end. Each layer has its role to play

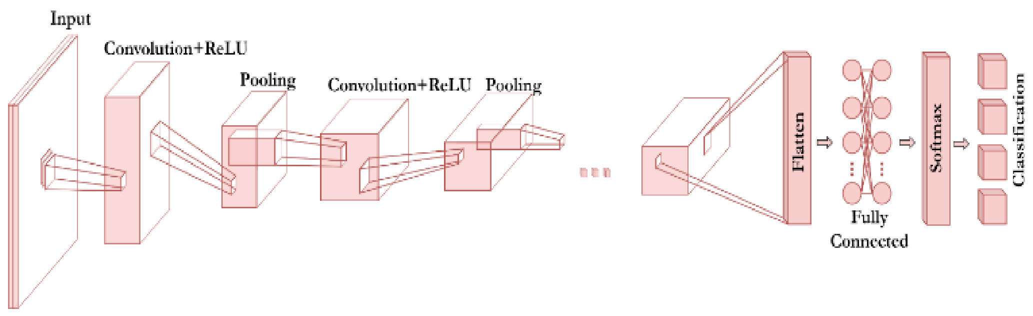


Fig 1. Basic CNN.

Input Layers
In a CNN, the typical inputs are predominantly images or videos. This layer receives the image's raw data, which has dimensions of 32×32×3 and width 'w' height 'h' depth corresponding to the color channels.

Convolutional Layers
The convolutional layer extracts the features from the input data from input layers by applying filters called kernels to the input images. These kernels are typically matrices of 2×2, 3×3, or 5×5 in size. They compute the dot product between the kernel weights and the corresponding patches of the image. The output from this layer is known as feature maps that highlight essential features in the input.

Activation Layer
After the convolutional layer, this layer introduces non-linear into the network by applying an activation function to the outputs from the previous layer. Commonly used activation functions include RELU, which is defined as *max (0, x*), *tanh*, and Leaky RELU, among others.

Pooling Layer
The pooling layer is placed between convolutional layers to downsize the volume of data, speed up computation, decrease memory usage, and prevent overfitting. The two most common forms of pooling are max pooling and average pooling.

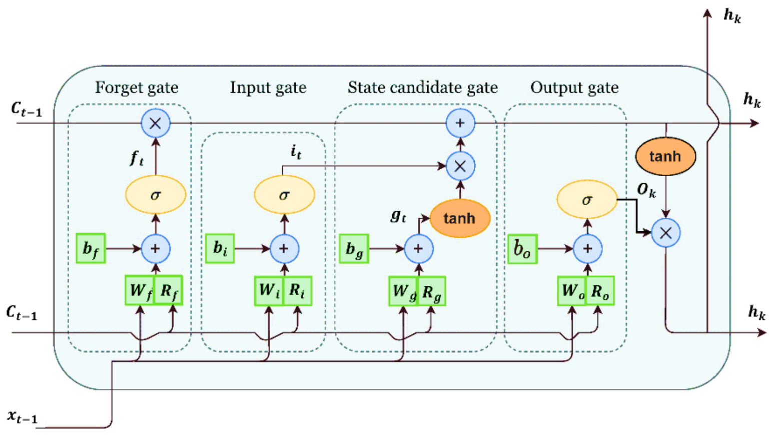


Fig 2. LSTM.

LSTM
LSTM is a type of RNN designed to capture long and short-term dependencies. An LSTM **Fig 2** typically consists of four layers called gates, EQU (1) to (2).

Input Gate (IG) (i_t)
The IG decides how much of the new data to allow into the cell state:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (1)$$

Here, σ is the sigmoid activation function, which outputs values between 0 and 1, effectively controlling the extent to which new data is allowed into the cell. W_i is the weight matrix for the IG, h_{t-1} is the previous output, x_t is the current input, and b_i is the bias [17].

Forget Gate (FG) (f_t)
The FG determines the amount of the previous cell state (c_{t-1}) to retain:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{2}$$

This gate filters out insignificant parts of the previous state by selectively letting only valuable parts based on the current input and previous output.

Output Gate (OG) (O_t)
The OG controls the output from the cell state to the rest of the network:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3}$$

This EQU (3) fixes which parts of the cell state are output based on the current and previous inputs.

Cell State Candidates ($\tilde{c}_t$)
This EQU (4) represents a candidate version of the new cell state, combining new input data with the previous output:

$$\tilde{c}_t = \tanh\,(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{4}$$

The tanh function helps regulate the network by scaling the output between -1 and 1, providing a normalized form of new data to be added to the cell state [18].

Cell State Update (c_t)
The cell state is updated using the FG, IG, and the new candidate cell state, EQU (5).

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{5}$$

This equation ensures that the cell state is a mixture of old data (FG) and new data (IG).

Output from the LSTM cell (h_t)
Finally, the output of the LSTM cell, EQU (6)

$$h_t = o_t \cdot \tanh\,(c_t) \tag{6}$$

The OG decides how much of the cell state to output and the *tanh* of the cell state c_t helps to scale the output values.

Stacked LSTM
A Stacked LSTM is a modified single-layer LSTM comprised of multiple hidden LSTM layers, each with several memory cells. Stacked LSTMs are particularly effective for complex sequence prediction problems that other models can handle. They effectively use their model parameters, rapid convergence, and better parameter efficiency in learning **Fig 3** [19].

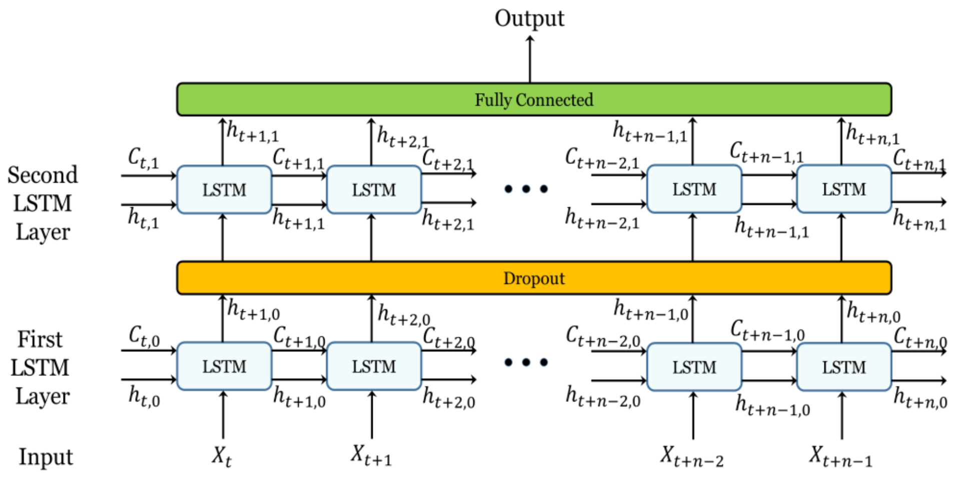


Fig 3. Stacked LSTM.

IV. PROPOSED MODEL

Data Preprocessing

Data preprocessing is done in numerous formats, such as *snoop, pcap, pppdump, btsnoop, i4btrace, LANalyzer, and pcapng*. These formats organize the data for input into the proposed IDS. The preprocessing involves several steps: segmenting the traffic by time, serializing the data into formats like *pkl* or *jso*n, depending on the programming language used, labeling serialized files, and generating a *greyscale* image of the traffic data.

Step 1 (Time Division)

Time division involves partitioning the incoming data stream based on the timing and type of potential attacks. Let $D = \{(t_i, x_i)\}$ represent the data stream, where t_i is the timestamp of the i-th data point, and x_i is the matching data value (or set of data values). Assume we have a set of known attack types A and corresponding time intervals during which these attacks are likely to occur. Each attack type $a \in A$ is associated with a time interval $[t_{\text{start}}, t_{\text{end}}]$. A segmentation function S is defined as those partitions D based on the specified attack time and type, EQU (7)

$$S(D, a, t_{\text{start}}, t_{\text{end}}) = \{(t_i, x_i) \in D: t_{\text{start}} \leq t_i \leq t_{\text{end}} \text{ and type } (x_i) = a\} \tag{7}$$

Here, type (x_i) determines whether the data point x_i matches to the type of attack a. The output of this function S is a subset of D containing only those data points that fall within the specified time interval and match the attack type. This subset is then used for further analysis or processing in the IDS.

Step 2 (Traffic Segmentation)

Traffic segmentation involves further dividing the dataset obtained from Step 1, which has already been segmented by time, into more discrete sessions. This is achieved by sharding the data based on the IP addresses of the attacking host and the victim host for each corresponding time.

From Step 1, we have a subset of data $S(D, a, t_{\text{start}}, t_{\text{end}})$ that has been segmented by attack type and time. Let H_{attack} and H_{victim} represent the sets of IP addresses for the attacking hosts and victim hosts. A function T is defined that partitions the subset S into sessions based on the IP addresses of the attack and victim hosts:

$$T(S, H_{\text{attack}}, H_{\text{victim}}) = \{S_k \subseteq S: \text{IP}_{\text{attack}}\ (x_i) \in H_{\text{attack}} \text{ and } \text{IP}_{\text{victim}}\ (x_i) \in H_{\text{victim}} \text{ for all } (t_i, x_i) \in S_k\}. \tag{8}$$

In this function, S_k represents a session, ip $\text{p}_{\text{attack}}\ (x_i)$ and $\text{IP}_{\text{victim}}\ (x_i)$ are functions that extract the attacking and victim IP addresses from each data point x_i. The output of this segmentation function T is a collection of sessions $\{S_k\}$, each session contains data points that share the same attacking and victim IP addresses within the specified time frame.

Step 3 (Serializing Data)

Serializing or flattening is the process where denormalized data, resulting from joining tables in a "one to many" (1:M) relationship, is compacted into repeating groups within a primary identity table. Let us consider a primary table P and a secondary table S with a one-to-many relationship. The data in P is then joined with S based on a shared key k, EQU (8).

$$J = \{(p, s_1, s_2, \dots, s_n): p \in P, s_i \in S \text{ and } s_i \text{ is related to } p \text{ through } k\} \tag{9}$$

Here, each p represents a record in the primary table, and $s_1, s_2, \dots, s_n$ are the related records from S. The process of serializing involves restructuring J such that the data from S is embedded into P as repeating groups, EQU (9)

$$F = \{(p, \{s_1, s_2, \dots, s_n\}): (p, s_1, s_2, \dots, s_n) \in J\} \tag{10}$$

In this structure, F, each primary record p is associated with a set of related secondary records $\{s_1, s_2, \dots, s_n\}$, which are serialized into a single row or record in the identity table.
The serialized dataset F encapsulates the primary and its related secondary data in a compact form, which simplifies and accelerates search operations by reducing the need to perform multiple joins during queries

Step 4 (Tag the Serialized File)

This step involves labeling the serialized data to facilitate more efficient data extraction, addressing the challenge of large file sizes. Assume from Step 2 that we have a set of traffic sessions $\{S_k\}$, where each S_k matches to a specific interaction between hosts. For each session S_k, predominant attack type a_k is identified as being associated with the session. The labeling function L is defined to assign a label to each session based on its attack type, EQU (10)

$$L(S_k) = a_k \tag{11}$$

where a_k is the attack type determined from the data characteristics of S_k. Then, a labeled package P_k is created for each session, EQU (11)

$$P_k = (S_k, L(S_k)) \tag{12}$$

In this packaging, each session S_k is paired with its corresponding label $L(S_k)$, which describes the type of attack the session data represents. The result is a collection of labeled sessions $\{P_k\}$, where each P_k contains the session data S_k and its associated label $L(S_k)$.

Step 5 (Sample Gray Image Conversion)
This final step in data preprocessing involves converting statical data into a format suitable for CNN input, specifically into 2-D matrices representing gray-scale images. The outcome from the preceding preprocessing steps is a set of labeled sessions, $\{P_k\}$. Each session P_k contains numerical data S_k which needs to be normalized to ensure all feature values are on the same scale, typically [0,1], EQU (11).

$$x'_{jk} = \frac{x_{jk} - \min(x_{jk})}{\max(x_{jk}) - \min(x_{jk})} \tag{13}$$

Here, x_{jk} is the $'\text{j}'$ feature of the $'\text{k}'$ session, and x'_{jk} is its normalized value. Each normalized session S'_k is converted into a 2D matrix M_k. Assume each session S_k comprises of a flattened array of features, reshape this array into a matrix, EQU (13)

$$M_k = \text{reshape}\ (S'_k, m, n) \tag{14}$$

where m and n are the dimensions that form the matrix representation suitable for image processing. Each element in the matrix M_k is then interpreted as a pixel in a gray-scale image. The pixel intensity is determined by EQU (14)

$$\text{Pixel}_{jk} = 255 \times (1 - x'_k) \tag{15}$$

In this model, a pixel's intensity is inversely proportional to the normalized feature value, with higher feature values resulting in darker pixels. The final output from this step for each session k is a gray-scale image represented by the matrix M_k.

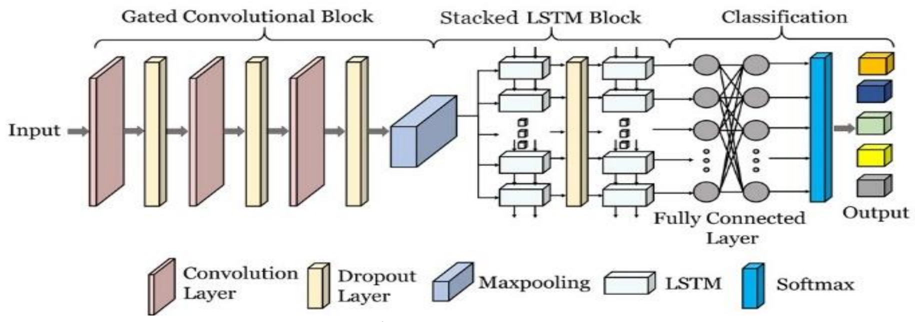


Fig 4. GC-LSTM.

Gated CNN + Stacked LSTM (GC-SLSTM)-IDS
The model of the proposed IDS is depicted in the **Fig 4.** the details of each layer and its function are explained here

Input
The input layer receives preprocessed network data as a gray-scale image, which is then directly inputted into the input layer.

Convolutional S
The convolutional layer employs a Gated Convolutional Neural Network (GCNN). This approach utilizes a gating mechanism inspired by RNN to filter the data selectively by discarding the less relevant data. The process begins by computing C as a linear transformation of F using weights θ_1 and bias d_1. Simultaneously, D is calculated as another linear transformation of F, this time using θ_2 and d_2, and then passed through the ReLU function to present non-linearity. The final output $h(F)$ of the gated convolutional operation is obtained by performing an element-wise multiplication of C and the transformed D, EQU (15) to EQU (17).

$$C = F \cdot \theta_1 + d_1 \tag{16}$$

$$D = F \cdot \theta_2 + d_2 \tag{17}$$

$$h(F) = C \circ \text{ReLU}\ (D), \tag{18}$$

Here, F denotes the output from the preceding layer. The terms θ_1 and θ_2 are weight matrices, while d_1 and d_2 serve as biased terms. The activation function used is ReLU. The symbol $\circ$ indicates element-wise multiplication between matrices. The model includes three convolutional layers of configuration, as shown in the following **Table 1.**

Table 1. Kernel Size of Convolutional Layer

Layer Number	Kernel Size	Number of Kernels
2	1×3	16
4	1×2	32
6	1×1	64

Dropout

The Dropout Layer is implemented to prevent overfitting and increase the simplification of the model by randomly turning off a subset of feature detectors during each training iteration. Each layer's neuron has a probability p of being deactivated, meaning its output is set to '0'. If the dropout rate is $p = 0.5$, it indicates a 50% chance that each neuron's output will be '0' during training. Due to issues like data set label imbalance, which can lead to overfitting, the dropout layers are included in the 3rd, 5th, and 7th in the CNN block and the 10th layer in the LSTM stack with probabilities of 0.6,0.5, 0.4, and 0.4.

Let x_i be the output from the i-th neuron. During training, with a dropout rate p, the output x_i is transformed as follows: EQU (18)

$$x_i' = \begin{cases} 0 & \text{with probability } p \\ \frac{x_i}{1-p} & \text{with probability } (1-p) \end{cases} \tag{19}$$

Max-Pooling

The Max-pooling in a CNN compresses features and removes redundancy while reducing the computational load of the model. The Max-pooling in this architecture is designed with a stride set to 2. Mathematically, if M represents the input matrix to the Max-pooling layer and S is the size of the pooling filter, the output matrix N at position (i, j) is calculated as follows:

$$N_{i,j} = \underset{k,l \in [1,S]}{\text{Max}} M_{2i+k,2j+l} \tag{20}$$

Here, k and l iterate over the matrix region covered by the pooling filter, selecting the maximum value within each pooling window as the output for that window.

The LSTM layer is initialized with a hidden vector size 128 for layers 9 and 11. The 11th layer outputs a vector h_i, which is input to the next layer. The LSTM units in layers 9 and 11 process data by gates and a cell state, each managing a hidden vector of 128 dimensions. The hidden state h_t at any time, step t in these LSTM layers is updated based on the current input x_t, the previous hidden state h_{t-1}, and the previous cell state c_{t-1} : $h_t = \text{LSTM}\ (x_t, h_{t-1}, c_{t-1})$

Fully Connected (FC)

The proposed model incorporates 2-FC layers, with the 1st layer containing 512 neurons and the 2nd containing 128 neurons. Let L_k represent the k-th fully connected layer in the model. The neurons in these layers are interconnected with all activations from the previous layer. The first FC layer, L_1, has 512 neurons. Each neuron in L_1 is connected to all outputs from the previous layer or network section. If x represents the input vector to L_1, the output y_1 from this layer can be represented by the EQU (21):

$$y_1 = f(W_1 \cdot x + b_1) \tag{21}$$

The 2nd FC layer, L_2, follows L_1 and contains 128 neutrons. It takes y_1 as input and produces the output y_2, calculated as EQU (22):

$$y_2 = f(W_2 \cdot y_1 + b_2) \tag{22}$$

Output

The output layer used a SoftMax function for multiclass classification that converts the logits from the previous fully connected layer into a set of probabilities that collectively sum to one, providing a distribution across the various classes. Given a vector of logits z from the preceding layer, the output probabilities for each class j are calculated using the EQU (23):

$$p_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{23}$$

where e^{z_j} is the exponential function applied to the logit of class j, and the denominator is the sum of the exponentials of all logits within the vector z, with K representing the total number of class options in the model.

Cross-Entropy Loss Function

For the IDS employing a SoftMax output layer for multiclass classification, the appropriate loss function to use is the Cross-Entropy Loss, also known as the SoftMax Loss. Given a set of true class labels y and the predicted probability distributions p from the SoftMax layer, the cross-entropy loss for a single data example can be expressed as EQU (24)

$$L = -\sum_{j=1}^{K} y_j \log\left(p_j\right) \tag{24}$$

where y_j is the binary indicator (0 or 1). The sum runs over all K classes. When calculating the loss over a batch of data points, the sum or average cross-entropy loss can be computed by sum or average of the individual losses over all data points in the batch, EQU (25)

$$L_{\text{batch}} = -\frac{1}{N}\sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \log\left(p_{ij}\right) \tag{25}$$

Here, N is the number of data points in the batch, y_{ij} Indicates whether class j is the correct class for the i-th data point and p_{ij} is the model's predicted probability that the i-th data point belongs to class j.

V. EXPERIMENT ANALYSIS

In this study, the CICIDS2018 (Canadian Institute for Cybersecurity) is employed, and it includes different attack scenarios. The datasets cover ten days of network traffic, four of which have DoS and DDoS attacks. This work uses data from four traffic days: Thursday, Friday, Tuesday, and Wednesday. This study evaluated the proposed model by training it individually and testing it on the same data sets. In addition, we also experimented by training on Thursday data and testing using Friday data, training using Tuesday data, and testing using Wednesday data. The proposed GC-SLSTM undergoes training and validation in comparison with other models, such as Model 1, Model 2, and Model 3, using a set of performance metrics including EQU (26) to EQU (29) results shown in **Table 2.**

Accuracy (Acc.)

This metric measures the overall correctness of the model and is defined as the ratio of correctly predicted observations to the total observations:

$$\text{Accuracy} \; \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{26}$$

Precision (P)

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It is a measure of a classifier's exactness:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \tag{27}$$

Recall (R)

Recall is the ratio of correctly predicted positive observations to all observations in the actual class. It is a measure of a classifier's completeness:

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{28}$$

F1-score (F1)

The F1 Score is the weighted average of Precision and Recall. This score takes both FP and FN into account:

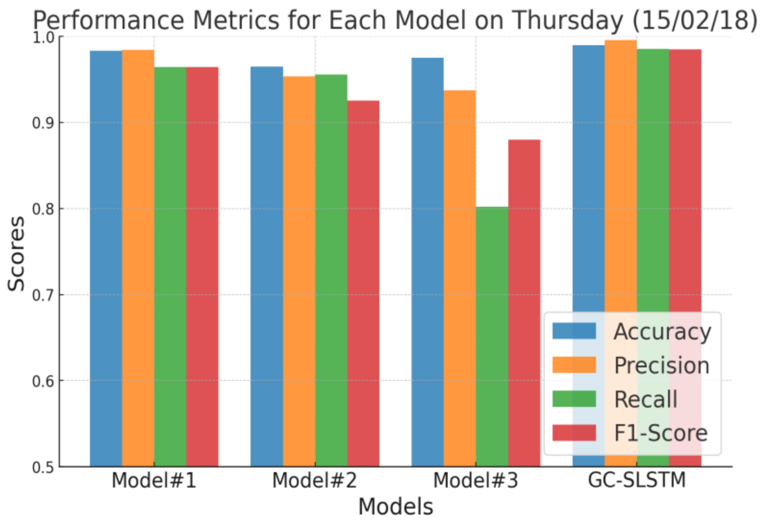
$$\text{F1} - \text{Score} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{29}$$

Experiment 1

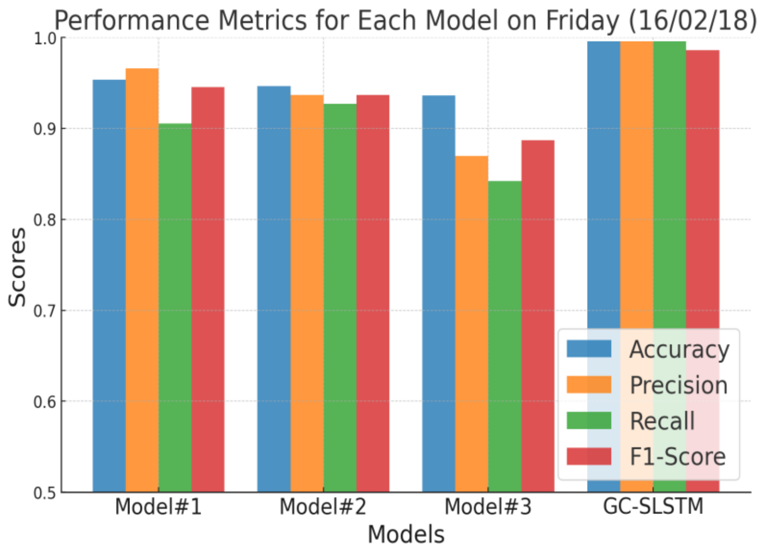
On Thursday, Friday, Tuesday, and Wednesday data sets, we train the models individually and then test them on the same data sets.

Table 2. Results (Experiment 1)

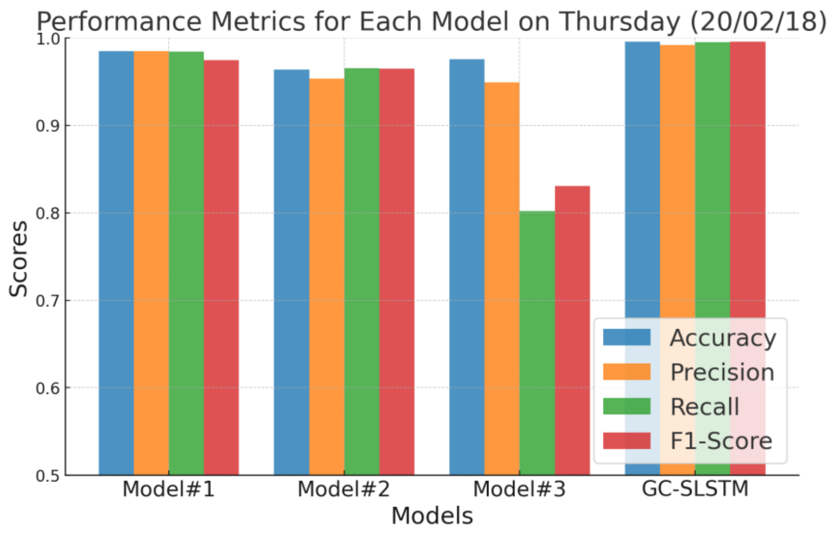
	Thursday				**Friday**			
Models	Acc.	P	R	F1	Acc	P	R	F1
Model#1	0.9835	0.9846	0.9646	0.9646	0.9536	0.9659	0.9055	0.9457
Model#2	0.9650	0.9536	0.9557	0.9254	0.9465	0.9366	0.9270	0.9367
Model#3	0.9753	0.9372	0.8021	0.8800	0.9365	0.8695	0.8423	0.8871
GC-S$	0.9898	0.9955	0.9857	0.9851	0.9959	0.9958	0.9959	0.9859
	Tuesday				**Wednesday**			
Models	Acc	P	R	F1	Acc	P	R	F1
Model#1	0.9848	0.9849	0.9845	0.9749	0.9836	0.9858	0.9752	0.9756
Model#2	0.9639	0.9536	0.9652	0.9650	0.9547	0.9561	0.9465	0.9464
Model#3	0.9759	0.9490	0.8021	0.8307	0.8423	0.8695	0.8423	0.8871
GC-SLSTM	0.9958	0.9919	0.9951	0.9957	0.9959	0.9958	0.9947	0.9959



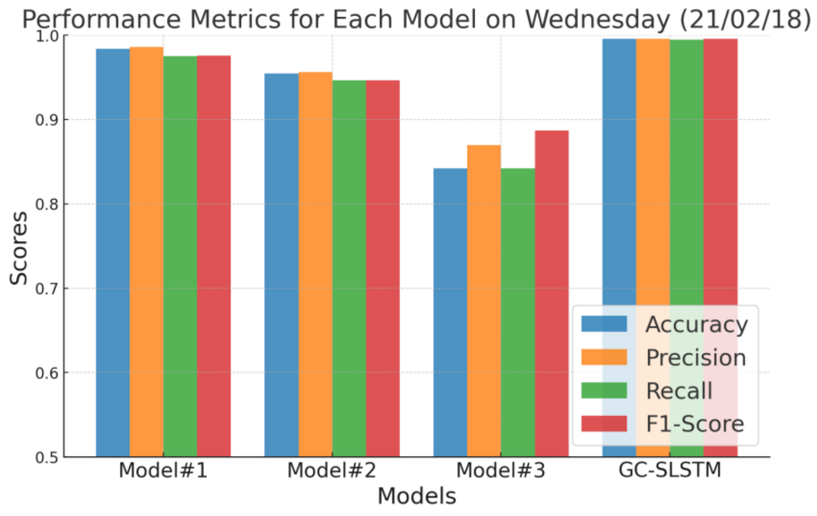
(a)



(b)



(c)



(d)

Fig 5. Performance Analysis Of The Proposed Model Compared To Other Models For A) Thursday, B) Friday, C) Tuesday, And D) Wednesday Dataset.

In the performance comparison shown in **Table 2** and **Fig 5.** (a) to (d), the GC-SLSTM has shown better results than other models. For instance, on Thursday and Friday datasets, the proposed model showed a performance of 0.9898 (Acc), 0.9955 (P), 0.9857 (R), and 0.9851 (F1) for Thursday dataset and 0.9959 (Acc), 0.9958 (P), 0.9959 (R) and 0.9859 (F1) for Friday dataset. Model#1 showed performance with an Acc of 0.9835, an F1-score of 0.9646 on Thursday, and a slight drop in performance on Friday to an Acc of 0.9536 and an F1-score of 0.9457. Model#2 scored lower than Model#1, with an F1 of 0.9254 on Thursday and 0.9367 on Friday. Model#3 scored with the lowest recall of 0.8021 on Thursday and slightly improved to 0.8423 on Friday. The trends continued with data from Tuesday and Wednesday, in which the proposed model had shown a performance of 0.9958 (Acc), 0.9919 (P), 0.9951(R) and 0.9957 (F1) for Tuesday and

0.9959 (Acc), 0.9958 (P), 0.9947 (R) and 0.9959 (F1) for Wednesday dataset. Model#1 on Tuesday has an Acc of 0.9848 and an F1 of 0.9749; on Wednesday, it has an Acc of 0.9836 and an F1 of 0.9756. Model#2 has accuracy scores of 0.9639 on Tuesday and 0.9547 on Wednesday and similar F1 of 0.9650 and 0.9464, respectively. Model#3's performance was variable, with a low recall rate.

Experiment 2

Thursday Data as Training and Friday Data for Testing

Analyzing the performance metrics using Thursday's data for training and Friday's data for testing **Fig 6**. The GC-SLSTM stands out with the highest metrics across the board—accuracy at 0.9486, precision at 0.9878, recall at 0.9747, and an F1-score of 0.8816. following the proposed model, Model#1 has high accuracy at 0.9356; however, its recall at 0.6233 is considerably lower and has an F1-score of 0.7671. Model#2 scores slightly lower in accuracy at 0.9154 and even lower in precision at 0.8640, and the recall rate drops further to 0.4874 and its F1-score down to 0.6469. Model#3 scored the lowest accuracy at 0.9056, precision, and recall, with values at 0.7681 and 0.7547 and a moderately balanced F1-score of 0.7413.

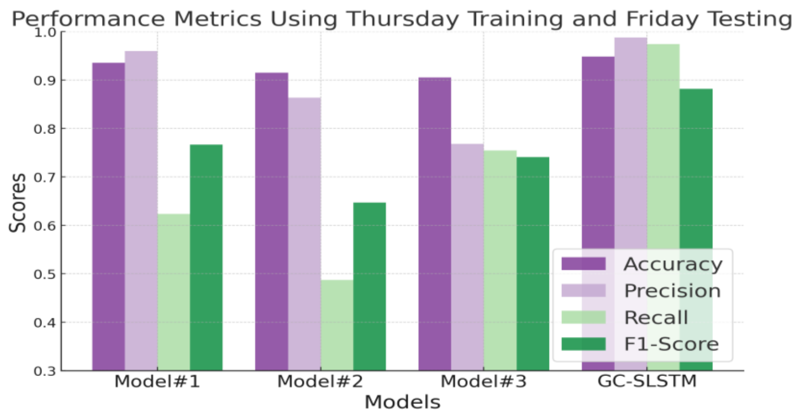


Fig 6. Performance Comparison for Thursday Data as Training and Friday Data for Testing.

Tuesday Data as Training and Wednesday Data for Testing

The analysis of the model performances using data from Tuesday for training and Wednesday for testing is shown in **Fig 7.** The proposed **GC-SLSTM** excels with the highest scores across all metrics: accuracy at 0.9691, precision at 0.8950, recall at 0.8255, and an F1-score at 0.9124. **Model#1** shows a moderate level of accuracy at 0.9126, struggles with precision at 0.8277, and recall at 0.4922, and its F1-score is only 0.5005, reflecting a significant imbalance between precision and recall. **Model#2** has lower accuracy at 0.9002 but improves precision at 0.8528 and recall at 0.7759 compared to Model#1 and a higher F1-score of 0.7477. **Model#3** reports the lowest accuracy at 0.8963 and precision at 0.6973. However, its recall at 0.6312 is higher than that of Model#1.

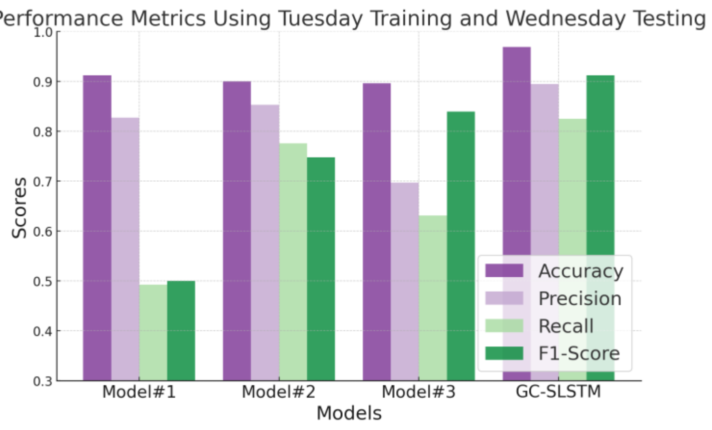
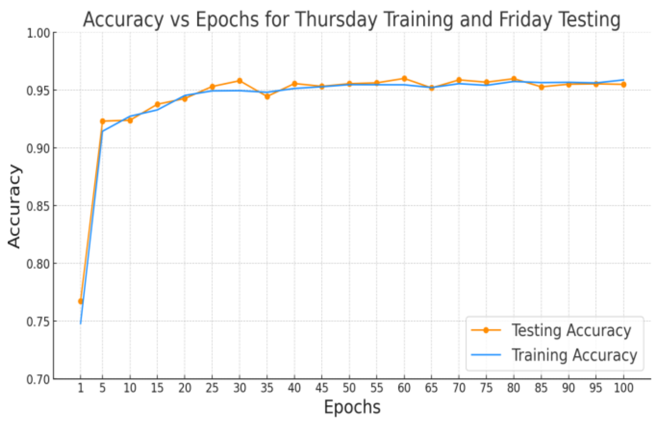
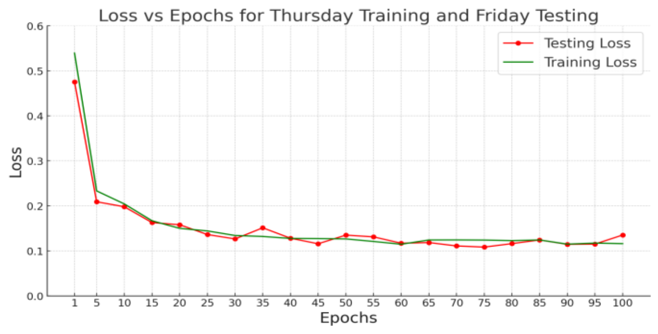


Fig 7. Performance Comparison for Tuesday Data as Training and Wednesday Data for Testing.

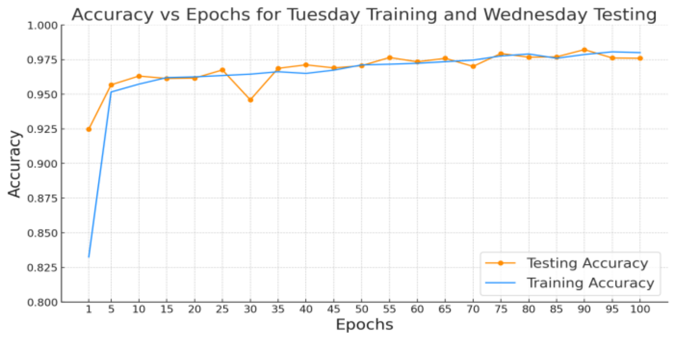
Experiment on Accuracy and Loss Over Epochs



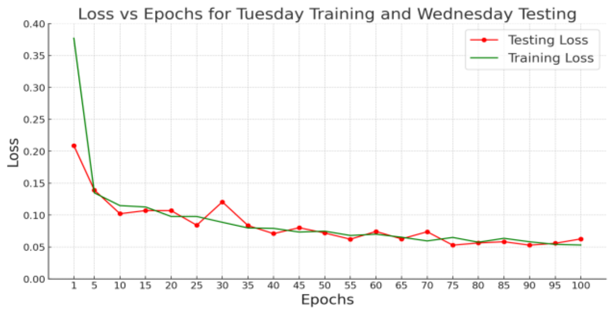
(a)



(b)



(c)



(d)

Fig 8. Analysis of Accuracy and Loss Against Epochs for The Two Testing Scenarios.

Fig 8 a & b displays the classification accuracy and loss function of the GC-SLSTM, using Thursday's data for training and Friday's data for testing. After 100 epochs, the training loss stabilizes at 0.116, while the validation loss levels off at 0.135. Although the loss function shows some fluctuations, it consistently decreases over the 100-epoch period, confirming the convergence of the proposed model. The classification accuracy on the training set converges relatively quickly, achieving stability after approximately 40 epochs. In contrast, while displaying minor fluctuations, the validation accuracy maintains a high level throughout the epochs. After 100 epochs, the training accuracy reaches 95.87%, and the validation accuracy stands at 95.49%. A similar trend is observed for the scenario using Tuesday's data for training and Wednesday's data for testing **(Fig 8 c & d),** where the model's loss stabilizes at 0.062 for testing and 0.053 for the training set after 100 epochs. The corresponding accuracies for this dataset reach 98% for training and 97.6% for testing after 100 epochs.

VI. CONCLUSION AND FUTURE WORK

In the field of cybersecurity, the integration of ML, such as LSTM + CNN, for the task of IDS could provide better prediction capability. This work attempted this integration by proposing GC-LSTM, which combines gated convolutional NN with the stacked LSTM. This work aims to capture the spatial and temporal features of the network data for effective IDS. For better NN training, this work incorporates an effective data processing pipeline that includes segmenting and converting the network data to an image for CNN processing. This model addresses the constant challenges of high FP in standard IDS and the limitations of such models that challenge them to adapt swiftly to new and evolving attacks. The proposed research was tested using the CICIDS 2018, focusing on four days, and different evaluation scenarios were examined. The proposed GC-LSTM proved higher accuracy, precision, recall, and F1-scores in each experiment than traditional models.

As cyber-attacks evolve, future work will focus on refining and developing such models, vital for maintaining robust NSS in an increasingly interconnected world.

CRediT Author Statement

The authors confirm contribution to the paper as follows:
Conceptualization: Rukmani Devi S, Manju A, Lakshmi T K, Venkataramanaiah B, Sureshkumar Chandrasekaran and Lakshmi Prasanna P; **Methodology:** Rukmani Devi S, Manju A, Lakshmi T K and Venkataramanaiah B; **Software:** Sureshkumar Chandrasekaran and Lakshmi Prasanna P; **Data Curation:** Rukmani Devi S, Manju A, Lakshmi T K and Venkataramanaiah B; **Writing- Original Draft Preparation:** Rukmani Devi S, Manju A, Lakshmi T K, Venkataramanaiah B, Sureshkumar Chandrasekaran and Lakshmi Prasanna P; **Supervision:** Rukmani Devi S, Manju A and Lakshmi T K; **Validation:** Sureshkumar Chandrasekaran and Lakshmi Prasanna P; **Writing- Reviewing and Editing:** Rukmani Devi S, Manju A, Lakshmi T K, Venkataramanaiah B, Sureshkumar Chandrasekaran and Lakshmi Prasanna P; All authors reviewed the results and approved the final version of the manuscript.

Data Availability

No data was used to support this study.

Conflicts of Interests

The author(s) declare(s) that they have no conflicts of interest.

Funding

No funding agency is associated with this research.

Competing Interests

There are no competing interests

References

[1]. S. P. Thirimanne, L. Jayawardana, L. Yasakethu, P. Liyanaarachchi, and C. Hewage, "Deep Neural Network Based Real-Time Intrusion Detection System," SN Computer Science, vol. 3, no. 2, Jan. 2022, doi: 10.1007/s42979-022-01031-1.
[2]. L. Yang, J. Li, L. Yin, Z. Sun, Y. Zhao, and Z. Li, "Real-Time Intrusion Detection in Wireless Network: A Deep Learning-Based Intelligent Mechanism," IEEE Access, vol. 8, pp. 170128–170139, 2020, doi: 10.1109/access.2020.3019973.
[3]. J. Liu et al., "Toward security monitoring of industrial Cyber-Physical systems via hierarchically distributed intrusion detection," Expert Systems with Applications, vol. 158, p. 113578, Nov. 2020, doi: 10.1016/j.eswa.2020.113578.
[4]. I. Dutt, S. Borah, and I. K. Maitra, "Immune System Based Intrusion Detection System (IS-IDS): A Proposed Model," IEEE Access, vol. 8, pp. 34929–34941, 2020, doi: 10.1109/access.2020.2973608.
[5]. N. Krishnadoss and L. Kumar Ramasamy, "A study on high dimensional big data using predictive data analytics model," Indonesian Journal of Electrical Engineering and Computer Science, vol. 30, no. 1, p. 174, Apr. 2023, doi: 10.11591/ijeecs. v30.i1. pp174-182.
[6]. N. Krishnadoss and L. K. Ramasamy, "Crop yield prediction with environmental and chemical variables using optimized ensemble predictive model in machine learning," Environmental Research Communications, vol. 6, no. 10, p. 101001, Oct. 2024, doi: 10.1088/2515-7620/ad7e81.
[7]. "An E-Commerce Based Personalized Health Product Recommendation System Using CNN-Bi-LSTM Model," International Journal of Intelligent Engineering and Systems, vol. 16, no. 6, pp. 398–410, Dec. 2023, doi: 10.22266/ijies2023.1231.33.
[8]. A. Alferaidi et al., "Distributed Deep CNN-LSTM Model for Intrusion Detection Method in IoT-Based Vehicles," Mathematical Problems in Engineering, vol. 2022, pp. 1–8, Mar. 2022, doi: 10.1155/2022/3424819.
[9]. B. Padmavathi and B. Muthukumar, "A deep recursively learning LSTM model to improve cyber security botnet attack intrusion detection," International Journal of Modeling, Simulation, and Scientific Computing, vol. 14, no. 02, May 2022, doi: 10.1142/s1793962323410180.
[10]. P. Krishnamoorthy et al., "Effective Scheduling of Multi-Load Automated Guided Vehicle in Spinning Mill: A Case Study," IEEE Access, vol. 11, pp. 9389–9402, 2023, doi: 10.1109/access.2023.3236843.
[11]. S. Sengupta et al., "A review of deep learning with special emphasis on architectures, applications and recent trends," Knowledge-Based Systems, vol. 194, p. 105596, Apr. 2020, doi: 10.1016/j.knosys.2020.105596.
[12]. Mahalakshmi, R. L. Kumar, K. S. Ranjini, S. Sindhu, and R. Udhayakumar, "Efficient authenticated key establishment protocol for telecare medicine information systems," Industrial, Mechanical And Electrical Engineering, vol. 2676, p. 020006, 2022, doi: 10.1063/5.0117522.
[13]. U. Chadha et al., "Powder Bed Fusion via Machine Learning-Enabled Approaches," Complexity, vol. 2023, pp. 1–25, Apr. 2023, doi: 10.1155/2023/9481790.
[14]. S. Kunjiappan, L. K. Ramasamy, S. Kannan, P. Pavadai, P. Theivendren, and P. Palanisamy, "Optimization of ultrasound-aided extraction of bioactive ingredients from Vitis vinifera seeds using RSM and ANFIS modeling with machine learning algorithm," Scientific Reports, vol. 14, no. 1, Jan. 2024, doi: 10.1038/s41598-023-49839-y.
[15]. S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, and N. F. Samatova, "Anomaly detection in dynamic networks: a survey," WIREs Computational Statistics, vol. 7, no. 3, pp. 223–247, Mar. 2015, doi: 10.1002/wics.1347.
[16]. R. K. Poluru and R. Lokeshkumar, "Meta-Heuristic MOALO Algorithm for Energy-Aware Clustering in the Internet of Things," International Journal of Swarm Intelligence Research, vol. 12, no. 2, pp. 74–93, Apr. 2021, doi: 10.4018/ijsir.2021040105.
[17]. B. R. R. Reddy and R. L. Kumar, "A Fusion Model for Personalized Adaptive Multi-Product Recommendation System Using Transfer [
[18]. S. Sengan, S. Vairavasundaram, L. Ravi, A. Q. M. AlHamad, H. A. Alkhazaleh, and M. Alharbi, "Fake News Detection Using Stance Extracted Multimodal Fusion-Based Hybrid Neural Network," IEEE Transactions on Computational Social Systems, vol. 11, no. 4, pp. 5146–5157, Aug. 2024, doi: 10.1109/tcss.2023.3269087.
[19]. S. Panneerselvam, S. K. Thangavel, V. S. Ponnam, and S. Sengan, "Federated learning-based fire detection method using local MobileNet," Scientific Reports, vol. 14, no. 1, Dec. 2024, doi: 10.1038/s41598-024-82001-w.