A Hybrid Deep Learning Approach for Analyzing and Detecting the Malware in Software Defined Networks

Vasantharaj Karunakaran*, Angelina Geetha
Department of Computer Science and Engineering, Hindustan Institute of Technology and Science, Padur, Chennai, India
E-mail: vasantharajk35@gmail.com, angelinag@hindustanuniv.ac.in
*Corresponding author

Keywords: convolutional neural networks, long short-term memory, multi-layer perceptron, POX, RYU, relu

Recieved: November 7, 2024

The rise of software-defined networking (SDN) has introduced new security challenges, particularly in detecting and mitigating malware threats within network infrastructures. Traditional malware detection techniques often struggle with the dynamic nature of modern cyber threats. This paper presents a hybrid deep learning-based approach for malware detection in SDN environments, leveraging Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Multi-Layer Perceptron (MLP). The proposed CNN-LSTM-MLP model integrates spatial, temporal, and fully connected feature extraction techniques to enhance classification accuracy. The study evaluates multiple LSTM architectures, including Bi-Directional-LSTM, Stacked-LSTM, and LSTM-MLP, demonstrating that the CNN-LSTM-MLP model achieves superior performance. The experimental results, conducted using datasets from the Canadian Institute for Cybersecurity, indicate that our model attains an accuracy of 98%, outperforming existing deep learning-based approaches. Additionally, the study integrates RYU and POX SDN controllers to simulate real-world network environments, ensuring practical applicability. The findings highlight the efficacy of hybrid deep learning models in securing SDN architectures against evolving malware threats.

Povzetek: Za bojšo varnost v programsko definiranih omrežjih in za zaznavanje zlonamerne programske opreme avtorja predlagata hibridni model, ki združuje konvolucijske mreže, dolgi kratkoročni spomin in večslojni perceptron ter ga vgradita v krmilni sloj z RYU in POX, s premišljenim predprocesiranjem, uravnoteženjem razredov in nadzorom prileganja za robustnejše odkrivanje napadov.

1 Introduction

The prevalence of Internet-connected gadgets has led to an increased risk of infection by harmful programs. Once attacked, attackers utilize persistence measures to ensure that affected systems remain compromised for long periods of time. Consequently, the presence of persistence in harmful algorithms poses a challenge for static analysis conducted by simple code inspection [1]. To identify harmful activities in a system, it is necessary to conduct dynamic analyses, execute code, and provide reports on system changes. There is a constant discovery of new harmful programs, and their quantity is growing fast. Therefore, it is challenging to examine and categorise all current harmful scripts utilising debugging and signatures. Developing defensive strategies in response to newly found harmful code is insufficiently rapid to keep pace with the frequency at which dangerous programs arise. To address a novel harmful code, it is imperative to promptly conduct analysis and categorisation of the code. Malicious programs of the same category exhibit resemblances in their utilisation of comparable libraries and APIS, resulting in parallels in program behaviour. Hence, by identifying and categorising novel dangerous codes into pre-existing families of harmful codes, we may ascertain the specific nature of the new malicious code and offer suitable protective strategies.

Software-defined networking (SDN)[2,3] is a new concept in networking that offers hope for overcoming the shortcomings of existing network infrastructures. To begin with, it dissociates the network's logic for controlling the network (the control plane) from the physical hardware responsible for forwarding traffic (the data plane), thereby breaking vertical integration.

The second benefit is that policy enforcement, network (re)configuration, and evolution are made easier with the control logic implemented in a logically centralized controller (or operating system for networks) and the data plane is separated from the network, reducing network switches to simple forwarding devices [4].

With a fine, distinct programming interface between the switches and the SDN controller, the data plane and the control plane may be separated. One or more flow tables contain the rules for packet processing in an OpenFlow switch. Each rule does a certain action (dropping, forwarding, altering, etc.) on a subset of the traffic based on the matching subset. A controller application [5] can configure an OpenFlow switch to act

as a firewall, load balancer, traffic shaper, router, or any number of other functions (think middlebox), depending on the rules imposed by the application.

Disentangling the responsibilities of those involved in defining network policies, implementing those policies in switching hardware, and forwarding traffic is a significant outcome of the SDN principles. By decomposing the network control issue into smaller, more manageable portions [6], this separation allows for the needed flexibility, simplifies network administration, and facilitates network evolution and innovation by making it easier to build and introduce new networking abstractions.

For the SDN-Controller to learn about the connections between the infrastructure layer's forwarding devices, it employs link discovery. So, to find these connections, the OFDP will act as a go-between for the SDN-Controller and the infrastructure-layer network devices. This is accomplished by OFDP using the Link Layer Discovery Protocol (LLDP) message format, and for each active SDN-Switch port in the network, the SDN-Controller sends out a huge number of LLDP advertising at relatively large, set intervals. To advertise LLDP, the SDN-Controller sends out Packet_Out OpenFlow messages to all of the active SDN-Switch ports. The data layer SDN-Switch, on the other hand, encapsulates the LLDP packet in a Packet_In OpenFlow message and sends the link information [7].

By combining spatial, temporal, and fully linked feature extraction techniques, the ultimate objective is to create a hybrid deep learning model (CNN-LSTM-MLP) that improves malware detection accuracy in the SDN Environment.

Here is the structure of the article: In Section 2, provides the literature review. Section 3 details the research methods used, while 4 provides context for the study. The experimental setup and findings are presented in Section 5, while the discussion follows in Section 6. Section 7 concludes the article.

2 Related work

Network virtualisation in the SDN architecture separates the forwarding and controlling activities of the network. This document explains how to set up and configure a control plane to function as an SDN controller. [8] provided a quick overview of the many OpenFlow-enabled controllers built on SDN and available in different programmable languages. The two Open Flow-enabled controllers, POX, a Python-based controller, and Floodlight, a Java-based controller, are the main subjects of this study. Using the effective network simulator Mininet, a performance comparison of both controllers is performed across various network topologies by examining network throughput and round-trip delay.

Either a single controller or several controllers can be used to deploy the SDN architecture. The latter faces a controller placement problem (CPP) in a large-scale network setting, whereas the first is not appropriate for large-scale networks. In order to achieve specific performance objectives, such as dependability, load balancing, latency, energy efficiency, and computation time, CPP entails the issue of installing the ideal amount of controllers inside a network. Over the years, a number of CPP approaches have been put forth, in which each has its own specific goals, advantages, and disadvantages. The results of [9] showed a number of current approaches and algorithms as well as a number of difficulties, including the requirement for an effective algorithm and for attack-aware, cost-aware, and energy-aware CPP schemes while ensuring a high Quality of Service.

The control and data planes are divided by SDN, and they are later synchronized using a control protocol like OpenFlow. The control and data planes of an SDN deployment called in-band control use the same physical network. It presents a number of difficulties, including data loss, network congestion, and security flaws. Despite these difficulties, in-band control offers a number of advantages, such as increased network flexibility and programmability, lower costs, and more dependability. The proposed methods that have been put forth by [10] thus far to improve in-band SDN control, which are divided into four primary groupings: quick failure recovery, automatic routing, distributed control network and bootstrapping. Apart from the above, the authors also provided elaborated analysis of Control plane summary tables and poses challenges in the field.

Over the past few decades, numerous network techniques have been suggested to enhance user performance. Software-defined networks (SDN) play a vital role in different network topologies and their efficient management. SDNs are categorized into commercial and open-source controllers since they are widely utilized in the present networking landscape. Many companies utilize both proprietary and open-source controllers. There exists a substantial amount of literature on these controllers; however, it does not provide an analysis or evaluation of the controllers' performance in different network configurations. [11] conducted a comparative analysis on the efficiency of two Python based open-source controllers, The initial evaluation involves applying Shortest Path algorithm to determine the most efficient route from the starting point to the destination. The second evaluation involves designing a customized network configuration using the Mininet simulator. Subsequently, the two end hosts in each network calculate the quality of service (QoS) measurements, including Jitter, throughput, packet loss, and packet delivery ratio. According to the examination results, it has been determined that POX surpasses RYU and is highly suitable for deployment in any scenario.

A unique method for detecting malware by combining Convolutional Neural Networks (CNN) with Long Short-Term Memory (LSTM) networks have been illustrated by [12] and the study highlights the constraints of conventional signature-based detection techniques when

dealing with advancing and intricate malware, especially zero-day threats. The proposed hybrid model efficiently detects fraudulent activity in API call sequences by using the spatial feature extraction power of CNNs and the temporal dependency capturing ability of LSTMs. The research exhibits a validation accuracy of 96%, highlighting the model's capacity to enhance cybersecurity measures by identifying and categorizing malware, even in situations when traditional approaches are inadequate.

A new method that combines Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN) to improve the accuracy of malware detection presented by [13]. This approach employs both static and dynamic characteristics of malware by transforming malware binaries into grayscale pictures for analysis, and by collecting temporal relationships using LSTM and parallel feature extraction using CNN. The model also utilizes Principal Component Analysis (PCA) to identify features, hence enhancing efficiency. When tested on a publicly available dataset of malicious software, this method shows superior precision, accuracy, and F1 scores compared to previous approaches. It has the potential to greatly improve system security.

A malware classification technique that employs the VGG16 model was optimized by [14] using picture-related datasets and the VGG19, ResNet-50, and InceptionV3 models were pre-trained to extract features from malware pictures. Subsequently, these characteristics were employed to categorize malware lineages utilizing six machine learning classifiers.

A technique to identify and categorize photos of malware was designed by [15]. This method employs image-based stacking ensemble techniques and extracts vital information from the images using local binary pattern (LBP) and grey-level spatial dependence matrices (GLCM). Subsequently, the system utilizes a CNN ensemble model to convert the high-dimensional characteristics into low-dimensional ones. Ultimately, a total of six machine learning classifiers are employed to identify and categorize the malicious software.

The author [16] illustrated a machine learning technique for detecting and classifying Android malware using a stacking ensemble-based convolutional neural network (CNN). The method entails training a pre-trained model known as EfficientNetB0 by fine-tuning it with virus pictures. The generated model is subsequently fed into a logistic lapse model that incorporates Machine Learning algorithms.

The author [17] has performed a comparative study on machine learning methods for identifying malware in Android apps, emphasising the importance of accurate and efficient detection. Models such as Random Forest, Extra Trees, and Logistic Regression were assessed, with Logistic Regression emerging as the top performer with a 97.31% accuracy rate. This research underscores the potential of machine learning in bolstering Android security and provides a foundation for future improvements in mobile malware detection.

Malware analysis is also promoted by Windows. Malware, a dangerous online threat, is currently the main focus of the research community due to the rapid emergence of new types of malware. Unfortunately, no matter how many approaches have been taken, no malware has yet been detected. The suggested approach accomplishes this goal by integrating machine learning with dynamic malware analysis techniques to detect and categorize Windows malware. To execute the executable in a restricted environment with few exposed resources, you may use the Cuckoo Sandbox Tool. After the execution is complete, you can analyze the behaviour patterns and statistics. The JSON report provided by [18] was used to choose the characteristics and their count frequencies.

Deep Learning techniques are employed to safeguard the controller by implementing robust security measures, which are crucial for ensuring uninterrupted availability and connection within the network. Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) , and Gated Recurrent Units (GRU) are suggested by [19] to identify and thwart intrusion threats. [20] assessed the aforementioned models using a recently published dataset (InSDN dataset). Ultimately, all the models demonstrate exceptional precision in identifying and detecting malware. Therefore, there is a notable enhancement in the detection of attacks when compared to one of the leading state-of-the-art methodologies. Malicious software poses a hazard to several software-intensive systems, leading to the development of various malware detection methods, frequently relying on sequential data processing. Long short-term memory (LSTM) is a type of artificial recurrent neural network (RNN) structure that is particularly useful for analyzing sequential data. However, there has been no research conducted to examine the effectiveness of various LSTM designs specifically for the purpose of detecting malware.

A Distributed Denial of Service (DDoS) attack, sometimes referred to as an internet services attack, assesses the impact of both traffic flow and throughput reductions to identify anomalies. This type of assault has a substantial influence on the entire Software-Defined Networking (SDN) system. A Deep Learning technique projected by [21] to enhance the efficiency of the Software-Defined Networking (SDN) by categorizing network switches as either trusted or malicious devices. This study presents a way for detecting attacks on Internet services using Software Defined Networking (SDN). The SDN controller has the capability to assess the movement of data, identify irregularities, and limit the flow of both incoming and outgoing data, as well as the origin of the data. The SDN recommends the utilization of a Convolutional Neural Network (CNN) for the purpose of detecting attacks, specifically targeting the identification of nodes that exhibit hostile behaviour.

3 Background

The purpose of malicious software, or malware, is to organizations and systems. Because malware can jeopardise operating systems, infrastructure, and sensitive user data, its detection and analysis are essential to cybersecurity. Deep learning algorithms have been used more and more in malware analysis to increase classification accuracy. However, feature selection and high-quality data preparation are essential for successful virus[22] detection.

Dataset Pre-processing Steps: The Canadian Institute for Cybersecurity provided the dataset used in this investigation, which includes over 17,341 malware samples divided into five categories: banking malware, SMS malware, adware, riskware, and benign samples. Both static and dynamic features that were taken from system logs, network activity, and API calls are included in the collection.

Data cleaning
Corrupted and duplicate entries were eliminated. Removed columns that weren't needed (such as timestamps and hash values, which aren't informative).

Feature extraction
- Selected essential characteristics, like memory use, system behaviour, and API call patterns, that are crucial to malware categorization.
- Reduced dimensionality while maintaining crucial information by using feature selection methods such as Principal Component Analysis (PCA) and Mutual Information.
- To provide consistent scaling, StandardScaler (zero mean, unit variance) was used to standardize numerical features.
- Label encoding was used to encode categorical features for model compatibility.

Data augmentation & splitting
Arrange the dataset into three sets: test (10%), validation (10%), and training (80%).
Oversampling techniques (SMOTE) were used for underrepresented malware classes in order to equalize the distribution of classes.

Reason for feature selection
Empirical analysis and domain expertise served as the main sources of feature selection. System logs and API calls are crucial components because they offer important insights into malware activities. By removing superfluous features, PCA increased computing effectiveness and improved model performance by lowering the possibility of overfitting.

Optimization of Hyperparameters
Bayesian Optimization was utilized to refine hyperparameters for the proposed CNN-LSTM-MLP model to enhance its performance,including:

Parameters of CNN
Kernel dimensions: (3x3, 5x5, 7x7)
Quantity of filters: 32, 64, 128
Activation functions: ReLU, Leaky ReLU

Parameters of LSTM
Quantity of concealed units: (32, 64, 128)
Quantity of superimposed layers: (1, 2, 3)
Dropout rate: (20%, 30%, 40%)

Parameters of the MLP
Quantity of dense layers: (2, 3, 4)
Number of neurons per layer: (64, 128, 256)
Activation functions: ReLU, Sigmoid

Parameters for training
Learning rates: (0.001, 0.0005, 0.0001)
Batch sizes: 32, 64, 128
Optimization Algorithms: Adam, RMSprop

Strategies for preventing overfitting
A number of regularization strategies were used to reduce overfitting:

Layers of dropout
0.5 dropouts are incorporated in fully connected layers and 0.3 dropouts in LSTM layers to randomly deactivate neurons in order to avoid becoming overly dependent on particular patterns.

L2 Weight Decay regularization
To penalize big weights, reduce model complexity, and improve generalization, L2 weight regularization (λ = 0.001) was added to the thick layers.

Early termination
To avoid needless overfitting, validation loss was tracked using an early stopping patience of 10 epochs, and training was stopped if no improvement was seen.

3.1 RYU and POX controller

In an SDN context, combining RYU and POX [23] controllers can help make the most of each controller's specific capabilities.

RYU
1. A software-defined networking framework based on components. Programmed using the Python language.
2. Offers software components with clear APIs to assist developers in building new apps for network control and administration.

POX:
1. A second-generation Python SDN controller.
2. Perfect for educational and prototyping applications due to its simplicity and ease of understanding.

3.2 LSTM

The well-known deep learning frameworks Keras and TensorFlow are compatible with the implementation and training of the model. For the purpose of feedback learning, the LSTM stores the kernel function and the scores of the neurons that make up the neural network in a memory unit[24]. An LSTM's ability to process all data points, rather than just one, is its main benefit over more conventional fully connected layers. This gives it more strength. We utilized a Long Short-Term Memory (LSTM) model developed using the Keras package and Tensorflow in our experiment. Dimensions of our hidden state neurons layer are [(64,32), (32,16), (64,32,32), (64,32, 32),] respectively.

3.3 Multi-Layer perceptron

An artificial neural network (ANN) with several layers of interconnected neurons is called a Multi-Layer Perceptron (MLP). By adding hidden layers between the input and output layers, an MLP can simulate intricate, non-linear interactions, in contrast to a standard perceptron, which is only capable of handling linearly separable problems.

MLP structure:
The feature vectors from the preprocessed dataset are received by the input layer.
Multiple fully linked neurons make up hidden layers, which use activation functions (like ReLU) to identify non-linear patterns.
Using a softmax activation function for multi-class classification, the output layer generates classification probabilities.

4 Proposed methodology

The objective of this study is to provide a sophisticated deep learning-based approach for identifying and preventing threats in software-defined networking (SDN) settings. This section focuses on the technique employed in our work, namely the hybrid threat-detection framework, dataset preparation, suggested network model, and dataset description.

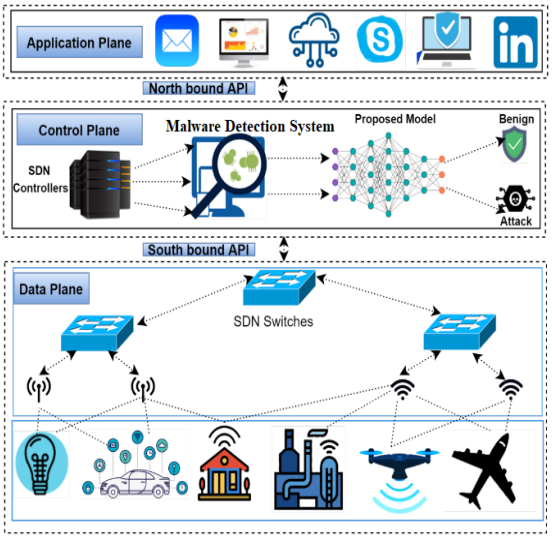

Figure 2: SDN-DMAD architecture

Fig 2 depicts the structure of a Software-Defined Networking (SDN) system that has been augmented with a framework for detecting malware known as Software Defined Networking-Dynamic Malware Analysis and Detection(SDN-DMAD) Architecture that has been inspired from[24]. The architecture is segmented into three main planes: the Application Plane, Control Plane and Data Plane.

The application plane is the uppermost layer that comprises several applications, including email, cloud services, and communication tools like Skype and LinkedIn. These apps communicate with the underlying network infrastructure via the Northbound API. These apps provide the transmission and reception of data, which is efficiently controlled and directed through the SDN infrastructure.

The Control Plane, located centrally, contains the Malware Detection System that is incorporated into the SDN Controllers. The malware detection system integrates a Proposed model (CNN+LSTM+MLP) [29,30,31] that is specifically intended to categorize dataset as either Benign or Attack. The model examines data flows and interacts with the Application Plane to adapt network policies and counteract risks.
Data Plane: The primary layer comprises of SDN switches [32] that oversee and guide the tangible data flow inside the network. These switches regulate the transmission of data to a wide range of IoT devices, industrial systems, smart homes, cars, and other similar entities. Real-time modifications are performed depending on judgments made by the malware system.

This design demonstrates the incorporation of sophisticated malware detection into SDN settings, allowing enhanced and intelligent control of network traffic across a wide range of applications and devices, resulting in increased security.

4.2 CNN-LSTM-MLP model

The deep learning model for attack detection starts by importing and preparing datasets that contain characteristics and labels associated with Trojan and ransomware assaults. Upon eliminating superfluous columns, the datasets are arranged in a coordinated manner, and supplementary characteristics are created to capture intricate connections. Next, the features are standardized using StandardScaler, which guarantees that each feature has a mean of null and a standard deviation of one. This step is essential for optimizing the performance of various Supervised Learning algorithms. The category labels are transformed into a numerical representation appropriate for model training using the Label Encoder. The dataset is subsequently merged and divided into separate training and testing sets using the parameters.

The Proposed Architecture has three branches: as shown in Fig 3. A CNN branch for capturing local patterns, an LSTM branch for capturing temporal dependencies, and an MLP branch for capturing relationships in the flattened feature space. The results of these branches are combined and then sent through other layers. The softmax function converts the output into a probability distribution across all classes, ensuring that the sum of probabilities equals one. This enables the model to make a classification decision by assigning the input to the class with the highest probability. The model is created via the Adam optimizer, and trained using Cross-Entropy (CE), which is suitable for classification tasks by penalizing incorrect predictions based on probability outputs.

The training process consists of 30 epochs, during which performance is monitored using a validation split. Following the training process, the model's accuracy and loss are assessed on the test set. Additionally, predictions are examined using classification reports, confusion matrices, ROC curves, and precision-recall curves. Ultimately, the model that has been trained is stored for future utilization, enabling its application to novel data for the purpose of detecting attacks.

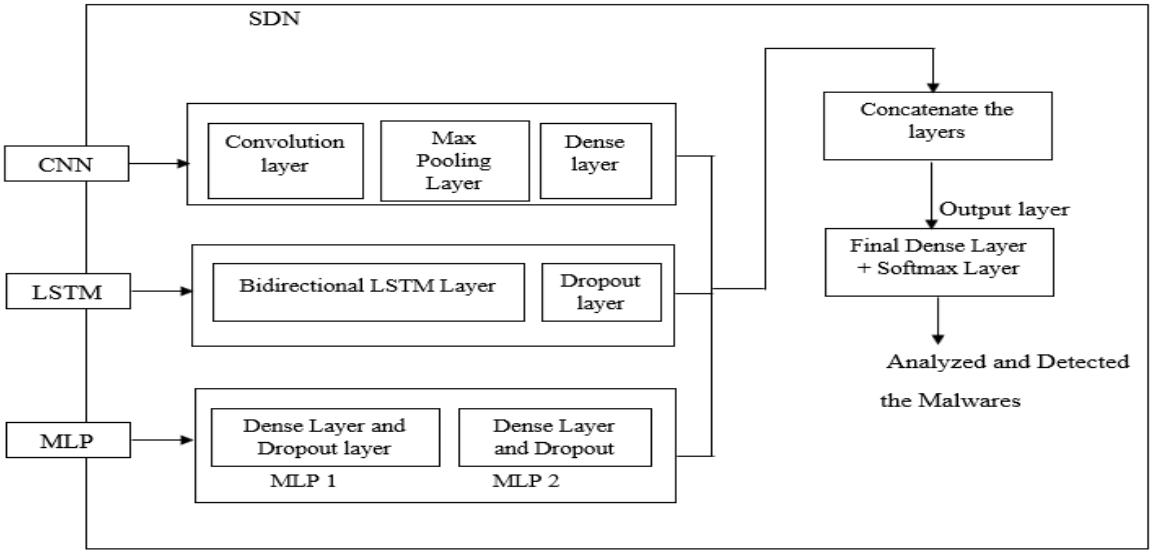

Figure 3: Proposed malware analysis and detection

The pseudocode of the projected model is also proposed as algorithm – 1

Input:

- dataset: Trojan_Banker_before_reboot_Cat.csv, Riskware_before_reboot_Cat.csv
- learning_rate(p): learning rate
- optimizer: adam
- training rounds: epochs (30 in this case)

Output:

Step - 1: **Load and Clean Data**: Load Trojan_Banker_before_reboot_Cat.csv and Riskware_before_reboot_Cat.csv, dropping unnecessary columns (Hash, Category, Family).

Step - 2: **Align and Transform Features**:

- Align columns of Riskware data to match Trojan data.
- Create additional features and standardize with StandardScaler.

Step - 3: **Prepare Labels and Combine Data**:

- Encode labels using LabelEncoder.
- Concatenate scaled features and encoded labels.

Step - 4: **Reshape and Split Data**:

- Reshape features for LSTM input.
- Convert labels to categorical and split data for training/testing.

Step - 5: **Define Model Architecture**:

- **CNN Branch**: Input, convolutional layers, pooling, flattening, and dropout layers.
- **LSTM Branch**: Input, bidirectional LSTM layers, and dropout layers.
- **MLP Branch**: Input, dense layers, and dropout layers.

Step - 6: **Combine and Compile**:

- Concatenate CNN, LSTM, and MLP outputs, add final dense layers, and compile the model.

The deep learning model may be represented by a set of fundamental equations that correspond to the CNN, LSTM, MLP branches, and the final output layer. The deep learning model mentioned may be represented by various fundamental equations corresponding to the CNN, LSTM, MLP branches, and the final output layer. The following equations are pertinent: The pertinent equations:

4.2.1 Convolution operation

$$\text{Conv1D}(x) = \sigma\left(\sum_{k=1}^{K} w_k \cdot x_{t+k-1} + b\right) \quad (1)$$

Where:

- x is the input to the convolutional layer.
- w_k represents the convolutional filter weights.
- b is the bias term.
- σ is the activation function (ReLU in this case).

Max pooling operation

$$\text{MaxPooling}(x) = \max_{i \in \text{window}} x_i \quad (2)$$

Where x_i are the elements within the Pooling Window

Flattening

$$\text{Flatten}(x) = \text{reshape}(x, [-1]) \quad (3)$$

Where x is the output from the final MaxPooling layer

The convolution process in Convolutional Neural Networks (CNNs) involves the application of filters to input data in order to extract important features. This is done by mixing the input with filter weights and a bias term, and then applying an activation function such as ReLU, as seen in equation 1. Next, a max pooling operation is performed to decrease the spatial dimensions of the data. This operation selects the largest value inside a pooling window, thus preserving the most important information as outlined in equation 2. Equation 3 involves taking the output from the max pooling layers and converting it into a one-dimensional vector. This allows the data to be easily processed by fully connected layers, enabling additional analysis and decision-making within the network.

4.2.2 LSTM branch equations

LSTM Cell Computation:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (4)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (5)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (6)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (7)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (8)$$

$$h_t = o_t \cdot \tanh(C_t) \quad (9)$$

where

f_t, i_t, o_t are the forget, input and output gates respectively

C_t is the Cell state and H_t is the hidden state

W_f, W_i, Wc and W_0 are the weight matrices and b_f, b_i, b_c, b_o are the bias Vectors

X_t is the input at the time step t

Several important gates that control the input flow organise calculations in an LSTM cell. First, the input gate regulates the amount of new data that is added to the cell state from the prior hidden state h_{t-1} and the current input xt. Which portion of the prior cell state C_{t-1} to be kept is decided by the forget gate ft . Using a tanh activation, the cell state update C_t creates a candidate for the current cell state. Ultimately, the new cell state C_t balances memory retention and new information incorporation by combining retained information from C_{t-1} with the candidate cell state. The LSTM is very efficient for sequential data processing because of these calculations, which enable it to retain pertinent data over time in a selected manner. The forget gate filters out outdated or irrelevant network behaviours while retaining critical attack patterns, ensuring the model

focuses on persistent threats. The input gate updates the cell state by incorporating new suspicious activity, such as sudden spikes in network traffic, to enhance malware detection. The output gate determines how much of the learned attack pattern influences the final decision, ensuring accurate classification of malicious behaviour.

4.2.3 MLP branch equations

Dense Layer:

$$\text{Dense}(x) = \sigma(W \cdot x + b) \quad (10)$$

The output in the dense layer is calculated by multiplying the input x with a weight matrix W and adding a bias vector b. This is then sent through an activation function σ, which is usually the rectified linear unit (ReLU). This technique enables the model to acquire intricate connections within the data by converting the input into a representation at a higher level.

5 Experimental setup and results

We do our experiment in the following manner to undertake malware analysis and detection:

1. Setting up the environment for SDN development
2. Results.

5.1 Setting up the environment for SDN development

The setup specified in Table 1 is used for these investigations. Each virtual computer in the emulated network runs its own Linux kernel, allowing for the configuration of connection characteristics like bandwidth. Mininet [18] was used to run this network. The SDN controllers utilized were RYU and POX.

Table 1: Specification

System	Specification
CPU	Intel Core i3-15, CPU 2.7 GHz
RAM	8GB – DDR4
GPU	2GB
OS	Ubuntu – 20.0
Simulator	Mininet 3.0
Controller	RYU and POX
Software Tool	Jupyter

5.1.1 RYU and POX controller

Process: In order to administer the network, switches talk to both the RYU and POX controllers. RYU Controller may detect topologies and then either publish messages to a broker or disclose pertinent data using a REST API, at the same time, the POX Controller uses REST API calls or message broker subscriptions to get data from the RYU controller. The SDN environment is managed by both controllers, who complement each other by playing to their strengths. The response time with respect to the packet count when the Controllers are getting executed [33].

5.2 Results

The files “Trojan_Banker_before_Cat.csv and "Riskware_before_reboot_Cat.csv" are likely to include data that is associated with the categories of Trojan Banker and Riskware malware, respectively. The datasets would consist of characteristics and classifications that are utilized to build a machine learning model for the identification of malicious software. The files include extracted characteristics that represent different parts of the malware's activity. The labels show if the virus belongs to the Trojan Banker or Riskware category.

Regarding the confusion matrix mentioned before, these datasets would have undergone preprocessing and then been used to train the model. The confusion matrix is a useful tool for assessing the model's ability to differentiate between instances of Trojan_Banker and Riskware, based on the datasets it was trained on.

The below Fig-4 represents the Confusion Matrix

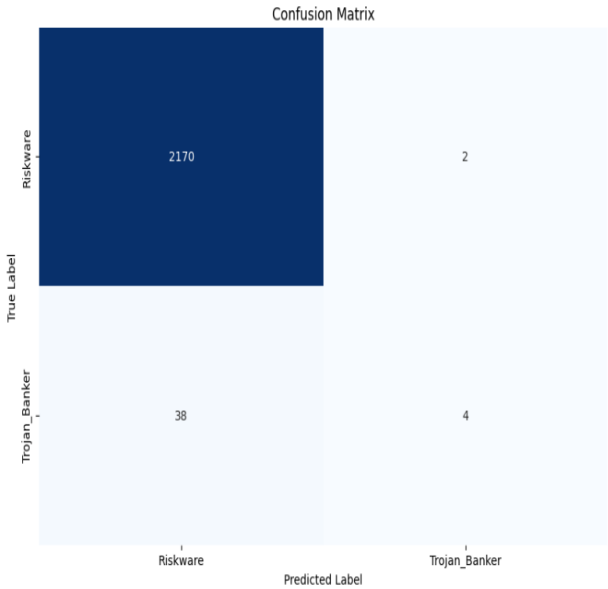

Figure 4 : Confusion matrix

True Positive Rate (TPR) and False Positive Rate (FPR) are crucial measures for assessing the effectiveness of a classification model, especially in binary classification applications like malware detection. From the Confusion matrix we took True Positive Rate (TPR) and False Positive Rate (FPR) for computation of the result and the comparison of Precision vs Recall is shown in Fig 5 and Fig 6 respectively. The provided Receiver-operating

characteristic curve (ROC curve) demonstrates the effectiveness of a binary classification

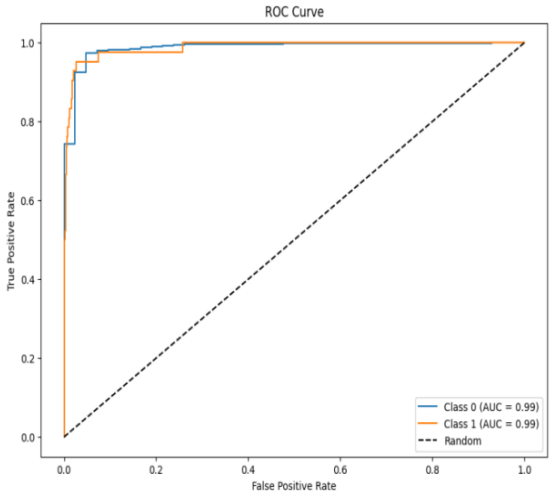

Figure 5: ROC Curve between TPR and FPR

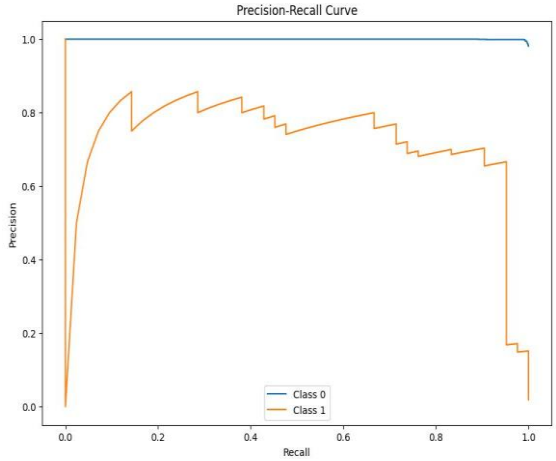

Figure 6: Precision Vs Recall

The above Precision-Recall (PR) curve depicts the effectiveness of a binary classification model in distinguishing between Class 0 and Class 1. It plots precision versus recall. Our dataset initially suffered from a significant class imbalance, where Class 0 had a much larger representation compared to Class 1. This imbalance led the model to favour Class 0, resulting in high precision and recall for it, while Class 1 exhibited fluctuating precision-recall values due to insufficient training examples. To address this issue, we applied Synthetic Minority Over-sampling Technique (SMOTE) to generate synthetic samples for Class 1, thereby balancing the dataset. As a result, the model's performance on Class 1 improved, leading to a more stable precision-recall curve, as observed in the updated PR graph. Receiver Operating Characteristic (ROC) curve illustrates the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR) at various threshold values [34]. The curve's position near the top-left corner indicates a model with a high TPR and a low FPR, indicating its ability to maximize accurate positive predictions while minimizing incorrect positive classifications.

The performance of the work can be predicted by Accuracy gain and Model Loss as depicted in the Fig 7 and Fig 8 respectively.

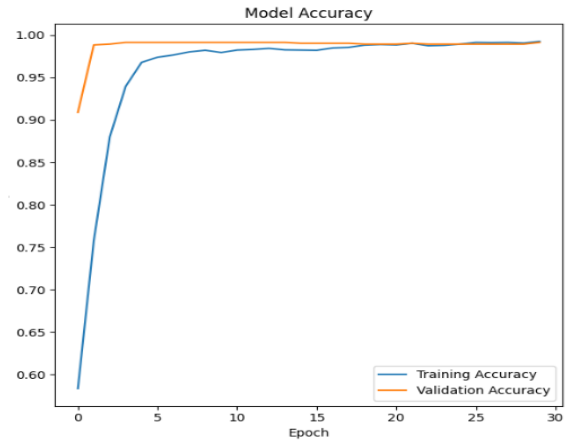

Figure 7: Training and validation accuracy

The above Fig 7 displays the accuracy of both the training and validation processes across a fixed number of epochs. The training accuracy (in blue) and validation accuracy (in orange) both see fast increases over the initial epochs, with the training accuracy finally approaching 100%. The validation accuracy likewise reaches a steady state at a high value, closely tracking the training accuracy. The model demonstrates a good level of accuracy on both the training and validation datasets.

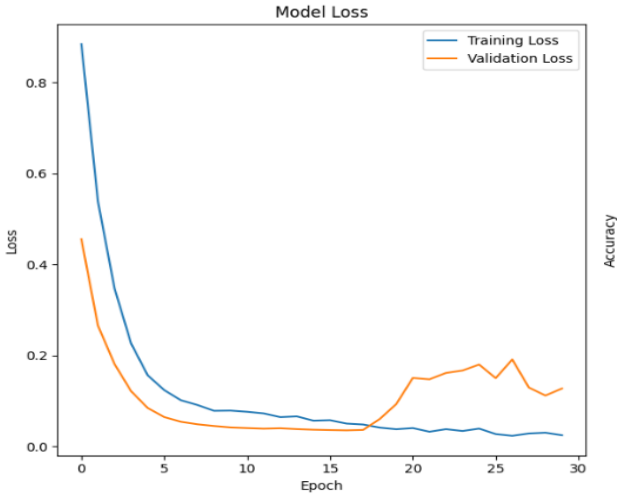

Figure 8: Model loss

The Fig 8 illustrates the training and validation loss as they vary with the number of
epochs. The training loss, represented by the blue line, consistently reduces as the model learns, eventually achieving a minimal value as the training continues. The validation loss (in orange) initially exhibits a similar decreasing trend, suggesting that the model is making

progress in terms of its performance on unknown data. Nevertheless, after around 10-15 epochs, the validation loss begins to rise while the training loss continues to decline, indicating that the model is starting to exhibit overfitting to the training data. We have executed the Malware Analysis and Detection using various methods of LSTM, CNN and MLP. This is shown in below Table – 2. Some of the methods are compared with the existing works as shown below in Table – 3.

Table 2: Comparison of our works

S. No	Method	Accuracy	Precision	Recall	F-Score
1	Bidirectional-LSTM	0.96	0.95	0.98	0.96
2	Stacked-LSTM	0.97	0.98	0.99	0.97
3	Multi-Layer Perceptron	0.97	0.99	0.99	0.99
4	LSTM-MLP	0.975	0.98	1.0	0.97
5	CNN-LSTM-MLP (Proposed)	0.98	0.98	1.0	0.98

Table 3: Comparison with existing works

Reference	Method	Dataset used	Detection Accuracy
Avci et al.[34]	Vanilla LSTM, Bi-Directional LSTM, Stacked-LSTM and CNN-LSTM	Malware Dataset (Canadian Institute of Cybersecurity)	0.88
Mustafa, O., Ali, K., & Naqash, T. [27]	LSTM & Self-Attention Architectures	CSE-CIC-IDS2018	0.97
Akhtar MS, Feng T [29]	CNN-LSTM	Malware Dataset from Kaggle	0.97
Marek Amanowicz and and Damian Jankowski [24]	MADMAS (SVM, KNN)	KDD Cup 1999	0.97
Proposed work	CNN-LSTM-MLP	Trojan Bank and Riskware (Canadian Institute of Cybersecurity)	0.98

6 Conclusion

The proposed technique efficiently combines Convolutional Neural Networks (CNNS), Long Short-Term Memory networks (LSTMS), and Multilayer Perceptrons (MLPS) to enhance malware detection in SDN environments. CNNS are employed to extract spatial features such as patterns in API usage, LSTMS capture sequential and temporal dependencies in malware behaviour over time, and MLPS integrate these features to perform accurate classification. This architectural synergy enables the model to detect complex and subtle malware

patterns more effectively than single-model approaches. However, the model's performance depends on the quality and diversity of the training dataset, and synthetic oversampling (SMOTE) may not fully reflect real-world complexity. Additionally, the hybrid model's computational intensity may limit deployment in resource-constrained or real-time settings. Future work will aim to improve generalizability across diverse malware families, reduce computational overhead, and further enhance adaptability for a dynamic SDN environment

References

[1] Hu, J. (2025). Online Criminal Behavior Recognition Based on CNNH and MCNN-LSTM. *Informatica*, *49*(12).

[2] Parol, P., & Pawłowski, M. (2014). Future-proof access networks for B2B applications. *Informatica*, *38*(3).

[3] Seun, E., Adebayo, A. O., & Osisanwo, F. Y. (2016). Introduction to software-defined networks (SDN). *International Journal of Applied Information Systems*, *11*(7), 10-14.

[4] Hussain, M., Shah, N., Amin, R., Alshamrani, S. S., Alotaibi, A., & Raza, S. M. (2022). Software-defined networking: Categories, analysis, and future directions. *Sensors*, *22*(15), 5551.

[5] Masoudi, R., & Ghaffari, A. (2016). Software-defined networks: A survey. *Journal of Network and Computer Applications*, *67*, 1-25.

[6] Lee, C., Yoon, C., Shin, S., & Cha, S. K. (2018). INDAGO: A new framework for detecting malicious SDN applications. *Proceedings of the IEEE 26th International Conference on Network Protocols (ICNP)*, 220–230. IEEE.

[7] Buzura, S., Peculea, A., Iancu, B., Cebuc, E., Dadarlat, V., & Kovacs, R. (2023). A hybrid software and hardware SDN simulation testbed. *Sensors*, *23*(1), 490.

[8] Bholebawa, I. Z., & Dalal, U. D. (2018). Performance analysis of SDN/OpenFlow controllers: POX versus floodlight. *Wireless Personal Communications*, *98*, 1679-1699.

[9] Isong, B., Molose, R. R. S., Abu-Mahfouz, A. M., & Dladlu, N. (2020). Comprehensive review of SDN controller placement strategies. *IEEE Access*, *8*, 170070-170092

[10] Carrascal, D., Rojas, E., Arco, J. M., Lopez-Pajares, D., Alvarez-Horcajo, J., & Carral, J. A. (2023). A Comprehensive Survey of In-Band Control in SDN: Challenges and Opportunities. *Electronics*, *12*(6), 1265.

[11] Naim, N., Imad, M., Hassan, M. A., Afzal, M. B., Khan, S., & Khan, A. U. (2023). POX and RYU Controller Performance Analysis on Software Defined Network. *EAI Endorsed Transactions on Internet of Things*, *9*(3).

[12] Gautam Karat, Jinesh M. Kannimoola, Namrata Nair, Anu Vazhayil, Sujadevi V G, Prabaharan Poornachandran, CNN-LSTM Hybrid Model for Enhanced Malware Analysis and Detection, Procedia Computer Science, Volume 233,2024, https://doi.org/10.1016/j.procs.2024.03.239.

[13] Thakur, P., Kansal, V. & Rishiwal, V. Hybrid Deep Learning Approach Based on LSTM and CNN for Malware Detection. *Wireless Pers Commun* **136**, 1879–1901 (2024). https://doi.org/10.1007/s11277-024-11366-y

[14] Kumar, S.; Panda, K. SDIF-CNN: Stacking deep image features using fine-tuned convolution neural network models for real-world malware detection and classification. Appl. Soft Comput. **2023**, 146, 110676.

[15] Naeem, H.; Dong, S.; Falana, O.J.; Ullah, F. Development of a deep stacked ensemble with process based volatile memory forensics for platform independent malware detection and classification. Expert Syst. Appl. 2023, 223, 119952.

[16] Yadava, P.; Menonb, N.; Ravic, V.; Vishvanathand, S.; Phame, D.T. A two-stage deep learning framework for image-based android malware detection and variant classification. Comput. Intell. 2022, 38, 1748–1771.

[17] Albazar, I., & Hassan, M. (2023). *A Model for Android Platform Malware Detection Utilising Multiple Machine Learning Algorithms*. Informatica, 48(17). https://doi.org/10.31449/inf.v48i17.6543

[18] Dutta, N., Jadav, N., Tanwar, S., Sarma, H.K.D., Pricop, E. (2022). Introduction to Malware Analysis. In: Cyber Security: Issues and Current Trends. Studies in Computational Intelligence, vol 995. Springer, Singapore. https://doi.org/10.1007/978-981-16-6597-4_7

[19] Karunakaran, V. ., & Geetha, A. (2023). Performing Dynamic Malware Analysis in Software-Defined Network using LSTM Technique. *International Journal of Intelligent Systems and Applications in Engineering*, *12*(2s), 411–419. https://ijisae.org/index.php/IJISAE/article/view/3641

[20] Islam, M. T., Islam, N., & Refat, M. A. (2020). Node-to-node performance evaluation through RYU SDN controller. *Wireless Personal Communications*, *112*, 555-570.

[21] Ahuja, N., Mukhopadhyay, D., Singh, L., Kumar, R., & Gupta, C. (2022). Attack Detection in SDN Using RNN. In *International Conference on Advances in Data-driven Computing and Intelligent Systems* (pp. 585-596). Singapore: Springer Nature Singapore.

[22] Ahmed, N., Ngadi, A. B., Sharif, J. M., Hussain, S., Uddin, M., Rathore, M. S., ... & Zuhra, F. T. (2022). Network threat detection using machine/deep learning in sdn-based platforms: a comprehensive analysis of state-of-the-art solutions, discussion, challenges, and future research direction. *Sensors*, *22*(20), 7896.

[23] Cabarkapa, D., & Rancic, D. (2021). Performance Analysis of Ryu-POX Controller in Different Tree-Based SDN Topologies. *Advances in Electrical & Computer Engineering*, *21*(3).

[24] Amanowicz, M.; Jankowski, D. Detection and Classification of Malicious Flows in Software-Defined Networks Using Data Mining Techniques. Sensors 2021, 21, 2972. https://doi.org/10.3390/s21092972

[25] Urooj, U., Al-rimy, B. A. S., Zainal, A., Ghaleb, F. A., & Rassam, M. A. (2021). Ransomware detection using dynamic analysis and machine learning: A survey and research directions. *Applied Sciences*, *12*(1), 172.

[26] Li, C., Lv, Q., Li, N., Wang, Y., Sun, D., & Qiao, Y. (2022). A novel deep framework for dynamic malware detection based on API sequence intrinsic features. *Computers & Security*, *116*, 102686.

[27] Mustafa, O., Ali, K., & Naqash, T. (2023, May). C-RADAR: A Centralised Deep Learning System for Intrusion Detection in Software-Defined Networks. In *2023 International Conference on Communication, Computing and Digital Systems (C-CODE)* (pp. 1-6). IEEE.

[28] Alshaibi, A., Al-Ani, M., Al-Azzawi, A., Konev, A., & Shelupanov, A. (2022). The comparison of cybersecurity datasets. *Data*, *7*(2), 22.

[29] Akhtar MS, Feng T. Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time. *Symmetry*. 2022; 14(11):2308. https://doi.org/10.3390/sym14112308

[30] Liu, J., Huang, X., Li, Q., Chen, Z., Liu, G., & Tai, Y. (2023). Hourly stepwise forecasting for solar irradiance using integrated hybrid models CNN-LSTM-MLP combined with error correction and VMD. *Energy Conversion and Management*, *280*, 116804.

[31] Feng, T., Liu, Y., Yu, Y., Chen, L., & Chen, R. (2024). CrowdLOC-S: Crowdsourced seamless localization framework based on CNN-LSTM-MLP enhanced quality indicator. *Expert Systems with Applications*, *243*, 122852.

[32] Ma, B., Lu, Q., Fang, X. *et al.* ARGCN: An intelligent prediction model for SDN network performance. *Peer-to-Peer Netw. Appl.* 17, 1422–1441 (2024). https://doi.org/10.1007/s12083-024-01656-4.

[33] Zhu, J., Karim, M. M., Sharif, K., Xu, C., Li, F., Du, X., & Guizani, M. (2020). SDN controllers: A comprehensive analysis and performance evaluation study. ACM Computing Surveys (CSUR), 53(6), 1–40. https://doi.org/10.1145/3391195

[34] Avci, C., Tekinerdogan, B., & Catal, C. (2023). Analysing the performance of long short-term memory architectures for malware detection models. *Concurrency and Computation: Practice and Experience*, *35*(6), 1-1.