Journal of Theoretical and Applied Information Technology
15st June 2026. Vol.104. No.11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TOKENIZATION-BASED TRUNCATING AND PADDING METHOD TO SOLVE LONG OPCODE SEQUENCE PROBLEM IN LSTM FOR IOT MALWARE DETECTION

FIRAS SHIHAB AHMED[1], NORWATI MUSTAPHA[2], NOR FAZLIDA MOHD SANI[3], RAIHANI MOHAMED[4]

[1]Faculty of Computer Science and Information Technology, University Putra Malaysia, Selangor, Malaysia

[2]Associate Professor, Department of Computer Science, Faculty of Computer Science and Information Technology, University Putra Malaysia, Malaysia

[3]Associate Professor, Department of Computer Science, Faculty of Computer Science and Information Technology, University Putra Malaysia, Malaysia

[4]Senior Lecturer, Department of Computer Science, Faculty of Computer Science and Information Technology, University Putra Malaysia, Malaysia

E-mail: [1]firasshahab48@gmail.com , [2]norwati@upm.edu.my, [3] fazlida@upm.edu.my, [4] raihanimohamed@upm.edu.my

ABSTRACT

Detecting malicious software in Internet of Things (IoT) datasets and environments remain a significant challenge for researchers striving to secure IoT networks. As the massive interconnection of Internet devices causes, most previous research has explored deep learning approaches, particularly the LSTM model, in detecting malware within operation codes (Opcodes) for ARM-based IoT applications, owing to its strong classification capabilities. Despite the advantages of using opcode features for detecting malicious software, the length of the opcode sequence poses a major challenge for deep learning methods such as the LSTM algorithm. Long opcode sequences can lead to information loss and increased computational burden, which may result in the vanishing gradient problem in the LSTM algorithm. To address this issue. This **paper proposes a tokenization-based truncating and padding method to solve this problem. It shortens the length of the opcode sequence while keeping the classification performance the same.** This approach extracts a subset of opcode sequences based on uniqueness, significantly reducing the length and size of the sequence while preserving the quality of the dataset. The method was evaluated on three IoT datasets from the Linux system and one dataset from the Windows system. The findings show that, for all datasets, the suggested approach performs better than current approaches in terms of accuracy and time.

Keywords: *Long Short-Term Memory (LSTM), Truncating and Padding, Cross-Validation, Malware detection, Opcode.*

1. INTRODUCTION

The expansion of IoT networks and systems has been rather significant and consistent recently. This growth has spurred the creation of several technologies in many different fields, including healthcare, agriculture, industry, smart cities, and smart electrical grids, thus increasing the number of smart devices linked to networks [1]. To meet the growing demand for IoT technologies, companies are designing intelligent IoT devices, often with inadequate security measures [2]. Modern malware such as Mirai is capable of launching rapid and large-scale Distributed Denial of Service (DDoS) attacks [3]. This escalating threat has prompted researchers to focus more seriously on ensuring security within IoT environments by exploring techniques such as game theory, fuzzy logic, behavior analysis, and machine learning [4-6].

The basic systems in IoT platforms are limited and heterogeneous, which creates challenges in designing effective security measures. Consequently, the answer should be a lightweight protection system guaranteeing rapid data processing by means of algorithms tuned for the lowest time and energy usage. Furthermore, protection mechanisms must be

adaptable and reliable to support various IoT-related technologies [7]. Most IoT malware detection systems now rely on signature-based approaches, security mode analysis, honeypots, or behavior- or anomaly-based detection. Among these, signature-based approaches have the best malware detection rate [8]. However, they are ineffective at identifying new malware attacks or recognizing previously unseen features. Recent studies have explored deep learning approaches for malware detection, as they are considered more efficient than traditional approaches [7, 9]. This study focuses on the LSTM deep learning model due to its strong classification capabilities.

Despite the effectiveness of LSTM in sequence-based malware detection, its performance is highly sensitive to input sequence length. In IoT environments, opcode sequences are often long and variable, which complicates training and increases computational overhead. These characteristics can degrade learning efficiency and limit the model's ability to capture meaningful patterns. [10].

This work addresses the problem by proposing a method to reduce long opcode sequences using the Truncating and Padding (TAP) technique, which is based on tokenizing the opcode sequences. The method extracts a subset of opcode sequences by identifying non-repeated opcode features. Consequently, it reconstructs the sequence length to align with the optimal input size for LSTM processing, thereby improving accuracy. This approach significantly enhances and maintains the dataset's quality in terms of both length and size when fed into the algorithm.

The overall contributions of this work are summarized as follows:

Reducing long opcode sequences by using the truncating and padding (TAP) method, which is based on the tokenized length of the opcode sequence. codes produced by the autoencoder to carry out the prediction task, while the front end converts a lengthy opcode sequence into a comparatively short, compressed sequence.

The study in [12] introduced a novel approach to address issues related to training the embedding layer and the autoencoder used in previous studies [13-15]. The proposed method requires the addition of a network layer during training or otherwise risks losing the semantic sequence after feature selection. The approach uses a preprocessing stage whereby an LSTM learns the features of the raw data to solve the Enhanced and maintained the quality of the dataset in both length and size by reconstructing the length of the opcode sequence to match the appropriate length required for efficient processing.

This paper's remaining sections are arranged as follows: Related work on opcode sequence reduction is reviewed in Section 2. The suggested approach is described in detail in Section 3. The experimental results are discussed in Section 4, and the study is concluded in Section 5.

2. RELATED WORKS

Sequential data based on prediction techniques takes time and is computationally expensive. Features include control flow, permissions analysis, machine learning, APIs, and system calls from current malware detection methods. Extraction of these characteristics, however, sometimes calls for professional investigation, which can take time. This work aims to extract rich, instructive features using opcode sequences. Opcode sequences are generally long and high-dimensional, so they can cause extended training times and poor performance. Many connected works have suggested different solutions for this problem.

One-hot encoding is judged inappropriate in [10] since it greatly extends the length of input vectors, producing high-dimensional data. The LSTM algorithm based on Word2Vec is used to handle this by including opcode names and API capabilities to classify malware into several groups. employed, incorporating opcode names and API functions to classify malware into different categories. Word2Vec is advantageous in this context, as it represents data using fewer dimensions compared to one-hot encoding. In other words, the model proposed by [11] consists of an opcode-level convolutional autoencoder. A dynamic recurrent neural network classifier at the back end uses the

issue of extended opcode sequences. This method lowers the computational cost connected with the N-gram technique and removes the necessity of hand feature specification.

In another study [16]. a convolutional recurrent neural network with opcode sequences was introduced as a new approach for malware detection using deep learning techniques. To deal with the long opcode sequence problem, the researchers proposed a convolutional autoencoder at opcode-level which transforms the long opcode sequences into relatively

compact representations at the front end. The method achieved an accuracy of 96% and an area under the receiver operating characteristic (ROC) curve of 0.99 with a true positive rate (TPR) of 95%.

In [17], the authors used both static and dynamic analytic techniques, and they used deep learning algorithms to identify malware that targets the graphics processing unit (GPU) or central processing unit (CPU) for bitcoin mining. In the static analysis of opcode sequences, shorter sequences were padded with zeros, while longer sequences were truncated to a uniform length. These sequences were then processed using the LSTM method. The SoftMax function was used to classify sequential data (opcodes and system calls) through LSTM, convolutional neural networks (CNN), and attention-based LSTM (ATT-LSTM). In both static and dynamic analysis, the study's accuracy rates were 95% and 99%, respectively.

The researchers in [18] proposed a new approach to opcode-based malware detection called SOEMD (Sequence of Opcode-based Embedded Malware Detection). The goal of this method is to capture common malware patterns while avoiding the challenges associated with extended opcode segments. SOEMD combines edge and node selection procedures with a random walk approach. It builds a new vector space out of low-dimensional sequential opcode representations using an embedding technique.

In another study, LSTM networks and opcode sequence features were used to identify malware [19]. A decompilation tool extracts the opcode sequence, allowing the LSTM to learn features related to malicious code patterns. To address the vanishing gradient problem that is caused by the long opcode sequences as and associated lengthy processing time, this work proposed a truncated backpropagation algorithm based on sequences. Additionally, the study enables parallel LSTM computation, which improves training efficiency and overall performance.

A hybrid approach to malware detection aimed at protecting IoT devices from obfuscated malware was proposed in [20]. The method employs an LSTM algorithm based on opcode sequences. An embedding technique turned string characteristics into vectors so that a deep learning model might be trained. The fast degradation of the vectors resulting from just the one-hot encoding approach can cause memory capacity to be exceeded. Thus, the research produced vectors with a predetermined embedding size for every opcode by using word embedding employing the Fast Text technique.

According to previous studies [21], only studies have explored the challenges posed by long opcode sequences. Where deep learning methods that used sequential data often face the vanishing gradient problem when analyzing long opcode sequence features. In order to address this problem, this research proposes a multi-model architecture for malware detection that emphasizes capturing general features. They used a Bi-LSTM to extract semantic information while keeping partial temporal relationships between the compressed features. This method compresses the original opcode sequence, resulting in a timing-rich semantic block sequence that acts as the input feature for the final classifier.

According to [22], they proposed a method for detecting malware by analyzing opcode sequences by a graph-based method, to addressing the problem of long opcode sequences. This approach constructs an opcode sequences graph for each executable file in the dataset, facilitating future extraction. With this method, packaged malware can be detected without the need for dynamic analysis or unpacking.

In [23], a multi-view learning methodology was proposed that integrates several perspectives, including opcodes, bytecodes, header information, permissions, attacker intent, and API calls to detect malicious applications. This approach automatically assigns weights to the different perspectives to improve detection performance across diverse environments. The method was evaluated on various platforms, including Windows, Android, and the IoT.

In [24], this study proposed methodology used both global and short-range dependencies in the input data. where pre-training eliminates random initialization, thereby improving the accuracy and robustness of malware threat detection. The approach accelerates the process compared to stacked LSTM models by reducing the length of opcode or bytecode sequences, that simplifies the overall method. As a result, it achieves improved accuracy, a better Area Under the Curve (AUC), and higher Matthews Correlation Coefficients (MCC) than a traditional LSTM, all while preserving similar detection times. In safety-critical systems, such as electronic health systems or the Internet of Battlefield/Military Things (IoT), where inadequate model convergence could have disastrous effects, this approach is especially well-suited for real-time malware threat identification.

Journal of Theoretical and Applied Information Technology
15st June 2026. Vol.104. No.11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The researchers in [25] investigated the detection of IoT malware using deep learning with recurrent neural networks (RNNs). Specifically, the method analyzes the execution operation codes (opcodes) of ARM-based IoT applications using RNNs. They trained them algorithms on a dataset with 270 benign and 281 malicious IoT applications. The trained model was then evaluated using three distinct LSTM models on 100 brand-new IoT malware samples that the model had never seen before. configurations. Among these, the configuration with two-layer neurons achieved the highest accuracy, correctly identifying new malware samples with 98.18% accuracy, based on tests conducted with ten different methods. Additionally, comparisons showed that the LSTM technique produced the greatest results when compared to other machine learning classifiers.

Based on the related works discussed above, it is clear that using opcodes is important for improving the accuracy of detecting malicious patterns and distinguishing malware samples. However, processing the entire opcode sequence often requires significant time, while using only a partial sequence may lead to ineffective detection. Therefore, an effective method is needed to address this issue.

This research proposes a new method that reduces long opcode sequences using the Truncating and Padding (TAP) technique, which is based on the tokenized length of the opcode. The method extracts a subset of opcode sequences based on unique features and reconstructs the sequence length to an optimal size for LSTM processing, achieving better performance than previous methods in both time efficiency and accuracy.

3. PROPOSED METHODOLOGY

Based on Figure 1, we show in this part the proposed tokenizing-based truncating and padding technique for malware identification. Based on tokenization, classification, and model validation, the next subsections address opcode extraction, truncating, and padding.

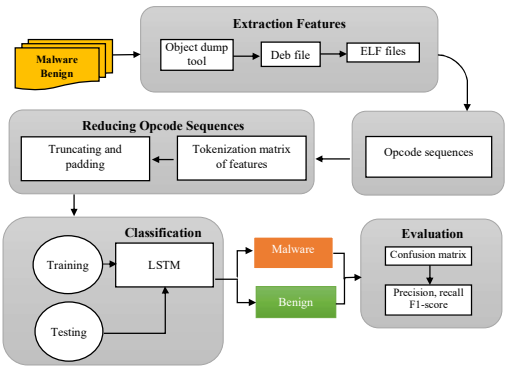


Figure.1 Proposed Methodology

3.1 Opcode features extraction

Preparing the machine code, translating it into assembly files, and then extracting features from the opcode sequences form the data extraction process. This procedure comprises numerous phases. Using the Debian installation package, all malware files were unpacked; each file was then broken down with the object dump tool. ELF files, which are chosen from the unpacked Debian package file (deb file), are displayed in Figure 2; each chosen file is decompiled to recover its opcode content. After that, the related opcode features are there and turned into sequences as depicted in Figure 1. Following the breakdown, a set of opcode sequences {Sq1, Sq2, Sq3 ... Sqn} with different lengths and sequences are depicted in Figure 1. Following breakdown, a set of opcode sequences {Sq1, Sq2, Sq3 ... Sqn} with different lengths and diameters

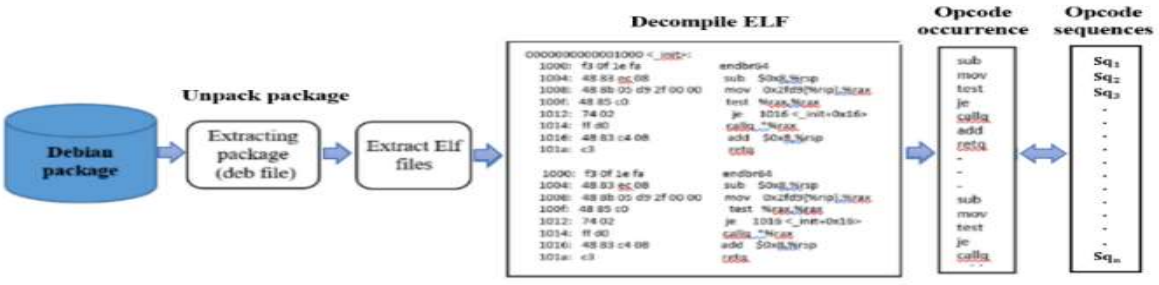


Figure.2. Steps of features extraction of the opcode sequences

Journal of Theoretical and Applied Information Technology
15st June 2026. Vol.104. No.11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.2 Tokenization based truncation and padding

This approach is able to reduce the complexity of the code sequence while simultaneously preserving the important behavioral characteristics of malicious samples in ELF format. Initially, the code extracted from binary files in ELF format is represented as two-dimensional arrays, then converted to one-dimensional sequences by sequentially combining the lines of code while maintaining the execution order. The generated sequences are stored and processed using TF-IDF analysis to identify the characteristics of distinctive operation tokens. During this stage, operation tokens with high Inverse Document Frequency (IDF) values, which represent potentially rare and unique malware behaviors, are retained, while the characteristics of frequently occurring operation tokens are removed to reduce redundancy and shorten the sequence length. The resulting reduced operation token sequences are then converted into numerical representations, and zeros are added to obtain uniformly sized vectors based on the maximum sequence size. This process reduces the computational load and produces uniform characteristic vectors suitable for machine learning and deep learning-based malware classification models.

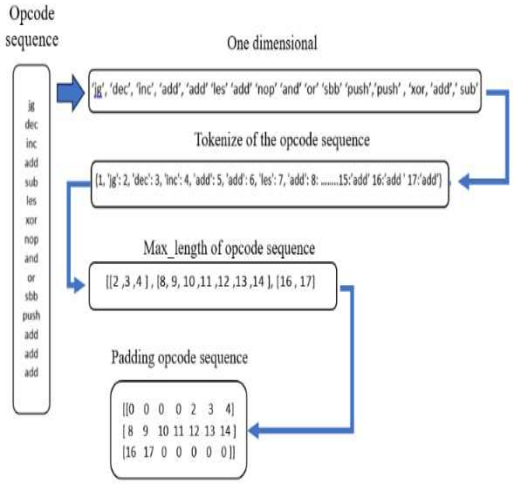


Figure 3. Steps of reducing opcode sequence

3.3 Classification

Using an LSTM model for classification, this study shows in Figure 4 the basic architecture consisting of three hidden layers. Three main gates, the input gate, forget gate, and output gate, form the basis for the LSTM. These gates let the model recall data for any length of time and record dependencies inside the input sequence, in addition to regulating the data that comes into and leaves the cell. Particularly, the forget gate decides which information from the cell state should be kept or thrown away [1, 2]. Its purpose is to spot data that, during the model's training phase,

Algorithm 1: TAP

Input: 2D opcode sequence files
Output: Equal-length reduced opcode sequences {$EqUSq_1$, $EqUSq_2$, …, $EqUSq_n$}
Initialization: Empty lists Sp_list, USq_list, EqUSq_list
Step 1 — Convert 2D opcode sequences into 1D vectors

```
For each opcode file in the dataset do
    Initialize empty vector Sp
    For each line in the 2D opcode file do
        Sp = Sp + line
    End for
    Add Sp to Sp_list
End for
```

Step 2 — Extract unique values using TF-IDF

```
For each Sp in Sp_list do
    Compute TF for each symbol in Sp
    Compute IDF for each symbol across all sequences in Sp_list
    Initialize empty USq
    For each symbol val in Sp do
        If IDF (val) is high (symbol is rare or unique)
            Add val to USq
    End for
    Add USq to USq_list
End for
```

Step 3 — Equalize sequence lengths using padding

```
Convert all USq sequences to decimal form
max_length = maximum length among all sequences in USq_list
For each USq_n in USq_list do
    If length (USq_n) < max_length
        Pad USq_n with zeros until length = max_length
    Add padded USq_n to EqUSq_list
End for
```

```
return EqUSq_list
```

has become useless. First, the current input and the previous hidden state (h_{t-1}). Following a bias term (f), the result passes through an activation function to produce a value within the 0–1 range. When the cell state value is almost zero, the model ignores the data, but when it is close to one, it keeps it for future use.The equation (1) shows this process applying the forget

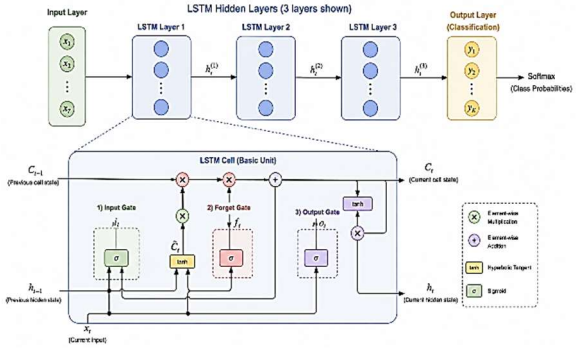


Figure 4 the basic architecture of LSTM

gate formula.

$$f_t = \sigma\,(Wf[h_{t-1}, x_t] + bf) \quad (1)$$

Where:

- f_t ; Forget gate activation vector at time step *t.*
- σ : Sigmoid activation function.
- W_f: Weight matrix for the forget gate.
- h_{t-1}: Hidden state from the previous time step.
- x_t : input vector at time step *t.*
- bf : Bias term for the forget gate.
- $[h_{t-1}, x_t]$ Concatenation of h_{t-1} and x_t.

Two major difficulties in this gate are time consumption and the vanishing gradient problem. We thus suggested a fresh approach for handling large sequences arriving at the gate. This method guarantees that, when the incoming information is useful, the cell state is set to 1, enabling the gate to efficiently update its memory and prevent the vanishing gradient problem that would otherwise cause the cell state to remain at 0, resulting in continuous forgetting and loss of important information. This loss compromises the learning and recognition capacity of the model for malware characteristics. Our second contribution also aims to narrow the data, thus lowering the processing time at this gate. Scanning and removing outliers, as well as reducing the memory footprint of the acquired features, help to accomplish this goal.

Using the input data, the values are organized and filtered similarly to the process in the forget gate. The function produces outputs ranging from -1 to +1, covering all possible values, which are then used to form a vector. Finally, this vector is multiplied by the regularized values to extract relevant information. The equations for the input gate are provided in Equations (2) and (3).

$$i_t = \sigma\,(W_i\,[h_{t-1}\,, x_t\,] + b_i) \quad (2)$$

Where:

- i_t : Input gate activation vector.
- W_i : Weight matrix and bias for the input gate.

$$\hat{C}_t = tanh\,(W_C\,[h_{t-1}\,, x_t\,] + b_C) \quad (3)$$

Where:

- $\hat{C}_t$: Candidate cell state
- $tanh$: Hyperbolic tangent activation function.
- W_C, b_C: Weight matrix and bias for candidate state.

Equation (4) shows how the candidate values are changed depending on the amount calculated for each state value after multiplying the preceding cell state by $f_t C_{t-1}$, therefore discarding the previously chosen information and direction.

$$\hat{C}_t = f_t\,C_{t-1} + i_t\,\hat{C}_t \quad (4)$$

Where:

- $\hat{C}_t$: Updated cell state
- C_{t-1}: Previous cell state
- i_t: decides how much of the candidate information $\hat{C}_t$ gets added to the cell state.

The task of the output gate is to retrieve important information from the cell state for training than pass it as output. Firstly, the tanh (hyperbolic tangent) function is applied to the cell state to generate a vector. Then, the sigmoid function processes the inputs h_{t-1} and x_t to filter and select the relevant values. In the final step, the resulting vector is multiplied by the filtered values, and the output is passed to the next cell. The corresponding equation for this gate is presented in Equation (5).

$$O_t = \sigma(W_O\,.\,[h_{t-1}, x_t] + b_O) \quad (5)$$

Where:

O_t: Output gate activation vector
W_O b_O: Weight matrix and bias for the output gate

In this study, we constructed four LSTM models that using four distinct datasets. where the first layer of each model consists of 100 nodes with the ReLU activation function, that serves as the output function for that layer's inputs. The second layer consist 100 nodes and uses the ReLU activation function to process its inputs. The third layer consist 100 nodes and employs the SoftMax activation function to generate output from its inputs. The number of neurons in the final layers depends on the specific classification task. For binary classification, the output layer contains two neurons. Table 1 provides detailed architectural specifications for all four models. A detailed explanation of each model will be given in the Experimental Results section.

Table 1. Proposal Architectures of LSTM models

Model	Network	Network	Activation Function	Neuron
Model 1	LSTM	Input	-	100
	Dense	Hidden	Relu	100
	Dense	Hidden	Relu	100
	Dense	Output	Softmax	**2**
Model 2	LSTM	Input	-	100
	Dense	Hidden	Relu	100
	Dense	Hidden	Relu	100
	Dense	Hidden	Relu	100
	Dense	Output	Softmax	**9**
Model 3	LSTM	Input	-	300
	Dense	Hidden	Relu	300
	Dense	Hidden	Relu	300
	Flatten	Hidden	Relu	100
	Dense	Output	Softmax	**10**
Model 4	LSTM	Input	-	300
	Dense	Hidden	Relu	300
	Dense	Hidden	Relu	300
	Flatten	Hidden	Relu	100
	Dense	Output	Softmax	**24**

3.4- Model Validation

This study employed the k-fold cross-validation approach to evaluate the performance of deep learning models. As shown in Figure 5, the dataset is divided into k folds for each iteration, with one fold used as the test set and the remaining folds as the training set[3]. This process is repeated until every fold has served as the test set, and the average scores across all iterations are calculated to present the final results.

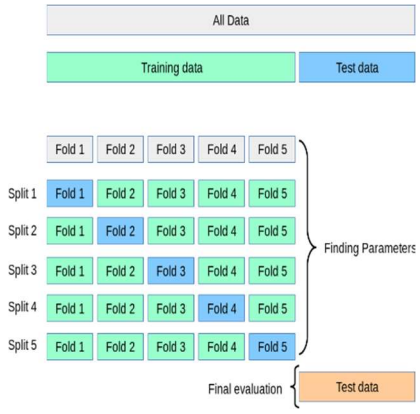


Figure 5. The k-fold cross validation

3.5- Performance metrics

Using a confusion matrix, which shows the actual class instances in the rows and the anticipated classes in the columns, the suggested model was assessed. Evaluation and computation of performance measures depend on this matrix. It is built from information taken during malware classification. Relatively to the overall expected results, the values assigned to each category following classification show prediction accuracy.

- True Positive (TP) denotes the actual instance as benign, as well as the prediction accuracy.
- True Negative (TN) refers to the actual case and the prediction, both of which properly diagnosed the sample as malware.
- False Positive (FP) means that a malware incident was mistakenly classified as benign.
- False Negative (FN) is the benign event that was mistakenly identified as malware.

From the confusion matrix, one can generate several evaluation measures to offer an understanding of the performance of the detection model. By use of these criteria, the validity and accuracy of the model in identifying malware are evaluated:

- **Accuracy**: the percentage of correct predictions out of all predictions made. It can be calculated using Equation (6).

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+F}} \quad (6)$$

- **Precision**: the percentage of predicted malware instances that are correctly identified as malware. It is calculated using the equation (7).

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \quad (7)$$

- **Recall**: or the detection rate, is the percentage of actual malware instances that were correctly identified. It can be calculated using the equation (8).

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \quad (8)$$

- **F-measure**: the harmonic mean of recall and precision. It is considered a crucial metric for evaluating the effectiveness of a detection model, especially in cases where the classes in the data are imbalanced. Recall measures the number of truly relevant results identified, while precision reflects the proportion of relevant results among all retrieved results. The F1-score is calculated using the formula shown in Equation (9).

$$\text{F} - \text{score} = \frac{2 \times \text{TP}}{2 \times \text{TP+FP+FN}} \quad (9)$$

Journal of Theoretical and Applied Information Technology
15st June 2026. Vol.104. No.11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. EXPERIMENTAL RESULTS

4.1 Experimental environment and datasets

This stydy employs an LSTM model to detect malware in Windows and IoT devices. The proposed model and experimental setup were implemented on a system running Windows 10 Pro 64-bit, equipped with 8GB of RAM and an Intel i5-2520M CPU @ 2.50GHz (4 cores). Four datasets of varying sizes and class distributions were employed in the tests. There are 552 malware samples from two different classifications in the first dataset, which is based on IoT. The Debian Linux package repository offers it for download at https://pkgs.org/. The second dataset, which Microsoft supplied, contains 10,868 samples divided into nine malware categories. It was collected from a Windows environment and can be accessed at the Kaggle Malware Classification Competition. The third and fourth datasets, containing malware from 10 and 24 families, respectively, were extracted from the Drebin dataset, which comprises 5,560 Android applications from 179 different malware families.

Four models were suggested in this study, each corresponding to one of the four datasets with different sizes and class distributions, as discussed in the previous section. Every rectangle in the LSTM algorithm applied in the suggested model stands for a layer in the algorithmic framework. Based on the initial dataset, Figure 6 shows Model 1. The LSTM model consists of four layers an input layer with 552 rows and 45 columns, two hidden LSTM layers each containing 100 units, and an output layer that employs the ReLU activation function, which is effective for classification. The batch size is 32, and the default learning rate is 0.01 to make the LSTM model work better. The model was built from scratch using the architecture described above, with only the batch size and learning rate hyperparameters tuned to ensure optimal performance,

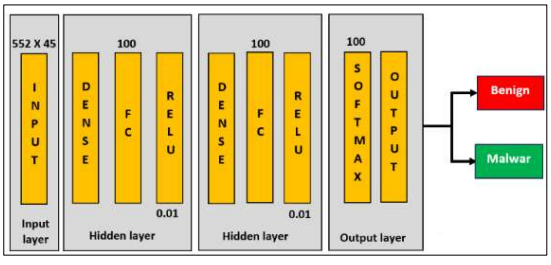


Figure 6. Model -1 For First Dataset

Figure 7, inspired by the second dataset, Figure 8 shows Model 2. The model's input layer counts 39 columns and 10,868 rows. Following a Dense (fully connected) layer with 100 nodes in the first hidden layer comes a ReLU activation function. on a corrected linear scale of 0.01. The structure comprises a dense and completely linked layer with 100 nodes, the second hidden layer also features a Softmax activation algorithm. Nine neurons in the output layer match the nine malware types,

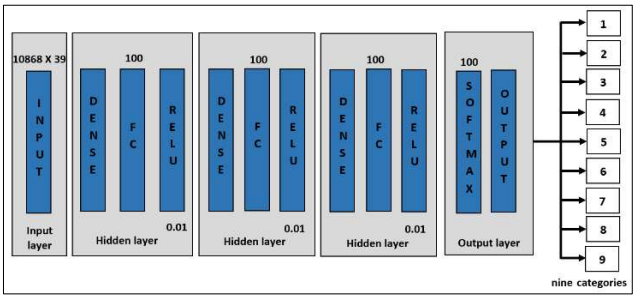


Figure 7. Model-2 for second dataset

Based on a third dataset, Figure 8 shows Model 3. The input layer runs 2,230 rows over 15 columns. Comprising a dense (completely connected) layer with 300 nodes and a ReLU activation function with a corrected linear scale of 0.01, the first hidden layer uses the same ReLU activation function. The second hidden layer also comprises dense and totally linked layers with 600 nodes. Comprising a dense and fully linked layer with 300 nodes and a flatten layer with 100 nodes, which transforms the output into a one-dimensional vector, the third hidden layer is the last output layer that utilizes the Softmax activation function for classification, after which this vector is transferred.

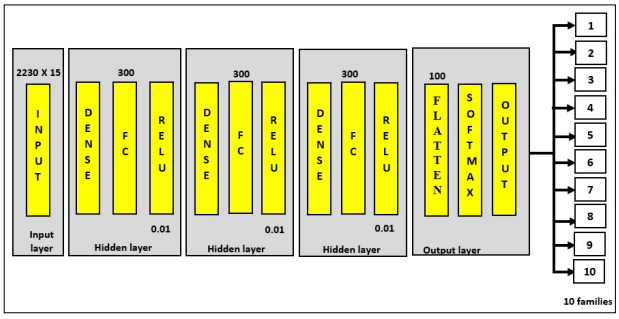


Figure 8. Model-3 For Third Dataset

Journal of Theoretical and Applied Information Technology
15st June 2026. Vol.104. No.11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Based on a third dataset, Figure 9 shows a fourth dataset. The input layer consists of 2,920 rows and 15 columns. Following a dense (fully connected) layer with 600 nodes in the first hidden layer comes a ReLU activation function with a rectified linear scale of 0.01. The second hidden layer, with the same ReLU activation function, contains 300 nodes and is dense and completely linked. The third hidden layer consists of still another thick, completely linked layer with three hundred nodes. Next is a one-dimensional vector derived from the output of the previous layer via a 100-node Flatten layer. Receiving this vector, the last output layer classifies it using the Softmax activation function.

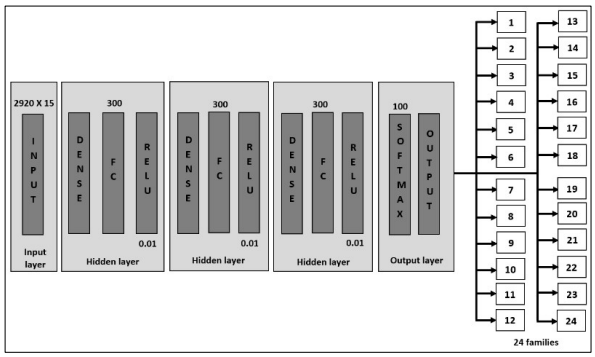


Figure 9. Model-4 For Fourth Dataset

4.2 Effect on length and size of opcode sequences

Using the tokenizing-based truncating and padding approach, we assess in this experiment the effect of opcode sequence length and size. Twelve opcode sequences taken from the first dataset are shown in Table 2, and Figure 10 each has a different length and width. The last two columns of the table clearly show that using the suggested approach considerably shortened the opcode sequence length and size. Table 3 lists, across all datasets, the lowered opcode sequence sizes.

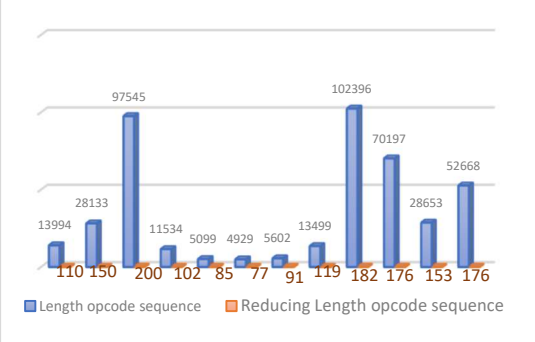


Figure 10. Reduced Length Sequences of First Dataset

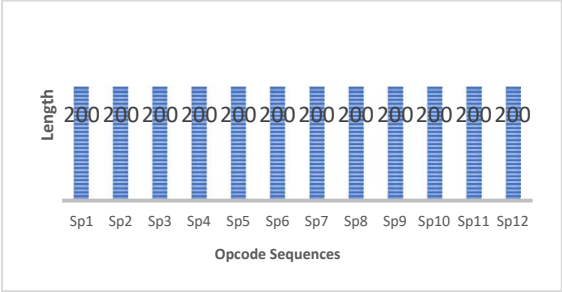


Figure 11. Padding The Sequence of First Dataset

Figure 11 shows the lengths of twelve opcode sequences after applying the padding process, with the longest sequence in the set identified as 200.

Table 2: Reduced lengths and sizes of sequences for first dataset

	Lengt	Size	Reduced	Reduced
Sq1	13994	14 KB	110	1 KB
Sq2	28133	28 KB	150	1 KB
Sq3	97545	96 KB	200	2 KB
Sq4	11534	12 KB	102	1 KB
Sq5	5099	5 KB	85	1 KB
Sq6	4929	5 KB	77	1 KB
Sq7	5602	6 KB	91	1 KB
Sq8	13499	14 BK	119	1 KB
Sq9	102396	100 KB	182	2 KB
Sq10	70197	69 KB	176	2 KB
Sq11	28653	28 KB	153	1 KB
Sq12	52668	52 KB	176	2 KB

Table 3: Reduced sequence sizes across all dataset

Dataset	Size of dataset	Size of dataset after reducing
First dataset	176.9 KB	23.3 KB
Second dataset	4.3 MB	1.1 MB
Third dataset	226.9 KB	52.5 KB
Fourth dataset	296.6 KB	68.5 KB

4.3 Features selection of opcode sequence

Using the Random Forest (RF) method, we identify and rank the most important opcode features contributing to malware detection. RF inherently provides feature importance scores by evaluating the impact of each feature on classification accuracy across multiple decision trees. In this study, we analyze opcode sequence features extracted from three of the four datasets considered, as shown in Figure 12. We notice that some features have higher importance scores than others in the first dataset because they have a stronger influence and are more relevant, such as features 10, 20, 38, etc.

Journal of Theoretical and Applied Information Technology
15st June 2026. Vol.104. No.11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

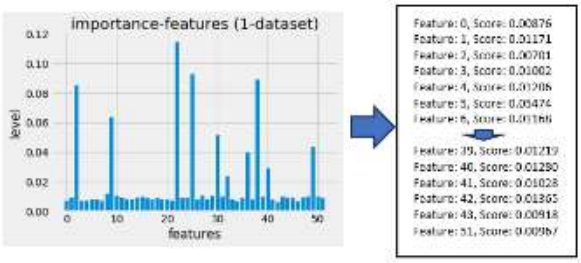


Figure 12. Important Features of First Dataset

In the second initial dataset, we observe that certain characteristics have higher relevance ratings than others. This is because certain features have a greater influence and are more significant. Some examples of these features include features 1, 9, 10, 20, and 39, among others, as shown in Figure 13.

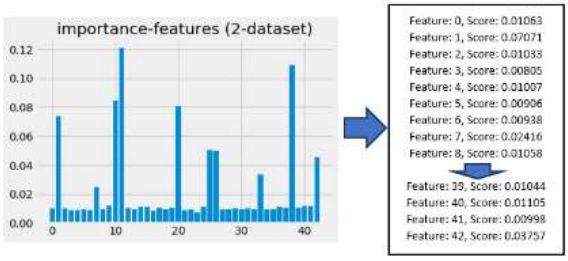
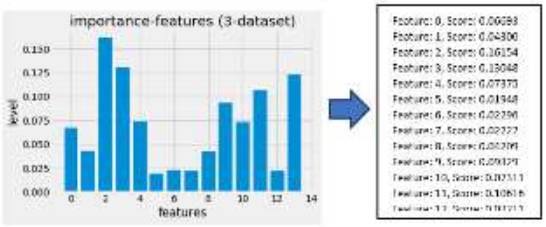


Figure 13. Important Features of Second Dataset

In the third dataset, we find that some characteristics are more important than others, which suggests that some features have a greater impact on the overall result. This variation implies that not all features are equally influential and a subset plays a more dominant role in shaping the results. Notable features are features 2, 4, 10 and 14 with relatively high importance scores and others with less prominence. This distribution shows the existence of important influential features in the data set as shown in Figure 14



.Figure 14. Important Features of Third Dataset

The fourth dataset's results clearly show that some characteristics are considered more relevant than others. It is because certain traits are more significant and have a greater influence to consider. Properties 2, 3, 9, and 13 are depicted, respectively, in Figure 15

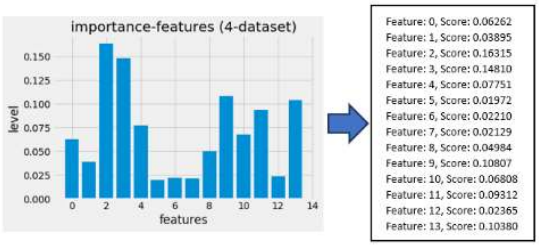


Figure 15. Important Features of Fourth Dataset

.

The most important features after reducing the opcode sequence are very important in detecting the malware efficiently by considering the most important opcodes which reduce the data dimensions and therefore improves the computational efficiency and scalability. Additionally, this process helps to remove redundant or irrelevant opcode features, which improves the ability of the model to better detect malware and distinguish between the behaviors of malicious and benign programs. This process makes the model interpretable, which enables the researchers to understand the attack mechanism, and achieves more efficient classification and better resource utilization.

4.4 Effect on accuracy and training time compared to methods used the same dataset

Table 4 shows the hyperparameters of the first model of LSTM, and the Table 5 and Table 6 present the outcomes of applying the LSTM method to the first dataset, comparing them with results from other research that utilized the same dataset in [4],[5], and [6], which employed deep learning and machine learning techniques. Regarding accuracy and execution time, our proposed method performed better than the others. This development shows the significant influence of our method on improving LSTM performance for malware detection. Moreover, this study presented performance experiments on other methods such as CNN, RNN, GRU, LR, SVC, and KNN, but the method proposed in this study remains better, and the reason is due to the networks.

Table 4: The Hyperparameters for Model -1

Layer Name	Layer Type	Output Shape	Units	Return Sequences	Activation	Parameters
lstm_11	LSTM	(None, 1, 100)	100	Yes	tanh	88000
lstm_12	LSTM	(None, 1, 100)	100	Yes	tanh	80,400
lstm_13	LSTM	(None, 1, 100)	100	Yes	tanh	80,400
dense_4	Dense	(None, 1, 1)	1	No	ReLU → Softmax	101
Optimizer	Learning Rate	Batch Size	Epochs			
Adam	0.001	32	1000			

(CNN) Being effective with spatial features, such as images, but with time series or sequence-based

malware, they can only capture local patterns via convolution windows. The GRU method offers faster and simpler processing; however, it suffers from limited computational capacity and lacks a separate memory cell. The RNN method has a common problem, which is the vanishing gradient. The KNN method works well with classes that are low-dimensional and well-separated. It has a weakness with high-dimensional sequences and is sensitive to irrelevant features. The LR method is effective with fixed feature spaces, but there is a problem in exploiting the temporal order of features, which affects the performance of the model. The SVC method is unable to capture nonlinear temporal interactions.

Table 5: Comparison of Results with Other Researches Using the First Dataset

Papers	Accuracy
[4] Haddad Pajouh, H	98.18%
[5] Darabian, H	99.6%
[6] Carlin, D	99.10%
Our method (LSTM)	**100.%**
CNN	93.37%
RNN	97.06%
GRU	92.16%
KNN	92.75%
SVC	92.74%
LR	92%

Table 6: Classification results of model -1

	Precision	recall	f1-score	Support
1	1.00	1.00	1.00	204
0	1.00	0.99	1.00	236
Accuracy				1.00
Macro avg		1.00	1.00	1.00
Weighted Avg		1.00	1.00	1.00

Figure 16 shows Model-1 correctness, therefore assessing the performance of the model learning over deep epochs. The graph illustrates that the suggested model is learning and generalizing well since both training and validation accuracy increases gradual

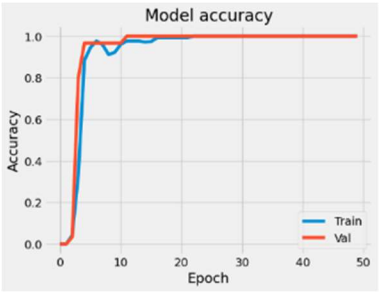


Figure 16. Accuracy Curve of Model-1

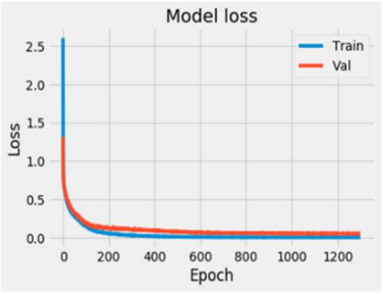


Figure 17. Loss Curve of Model-1

Figure 17 shows the loss of Model-1 for the first dataset. Model loss is a fundamental feature of deep learning methods since it connects actual results with model predictions. To ascertain the degree of a model's performance, they calculate the expected and actual values. This disparity, sometimes known as loss, guides optimization toward higher model accuracy. The chart shows that the suggested model keeps declining for both training and validation over epoch, which clearly indicates efficient model learning.

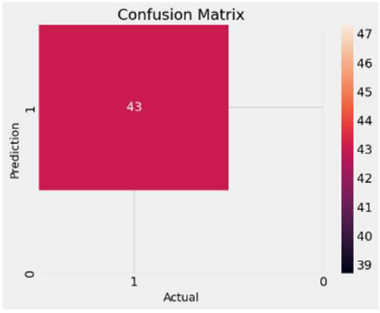


Figure 18. Confusion Matrix of Model-1

Figure 18 displays the confusion matrix comparing the performance of Classifier 1 to the actual results. The findings show that the suggested model performed rather well in differentiating benign from malicious samples.

4.5 ROC AUC curve

This study used a crucial measure, the ROC AUC curve, to evaluate the performance of the four models of the proposed LSTM method in this research. The acronym for receiver operating characteristic is ROC. As the decision threshold changes, the plot illustrates how well a binary classifier can distinguish between two classes (for example, malware vs. benign IoT device behaviour, as shown in the 10 and 11 equations).

AUC = 1.0 → perfect separation (malware always ranked above benign devices)
AUC = 0.5 → no discrimination (like flipping a coin)
AUC < 0.5 → worse than random (model is inverted)

The Equation:
X-axis → False Positive Rate (FPR) = percentage of typical devices that are mistakenly identified as malicious

$$FPR = \frac{False\ Positives}{False\ Positives + True\ Negatives} \quad (10)$$

$$TPR = \frac{True\ Positives}{True\ Positives + Fal\quad Negatives} \quad (11)$$

ROC curves and AUC scores are calculated for LSTM models using the code specific to each class. ROC curves (G) are then plotted for the classes, showing the discrimination performance of each with a line representing a random guess. The resulting graph provides a graphical assessment of the models' classification performance. Here, we observe that in our model, a higher AUC indicates better discrimination between malware and benign behavior, indicating excellent model performance with LSTM: ROC AUC=1.000, as shown in Figure 19.

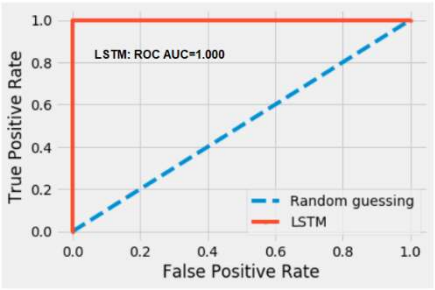


Figure 19. Confusion Matrix of Model-1

4.6 SECOND DATASET

Figure 7 shows the hyperparameters for Model -2 of LSTM, and the findings shown in Table 8, which show our suggested method implemented on the second dataset, amply show that this method beats those utilized in past studies on the same dataset. Reducing long opcode sequences with the truncating and padding technique, which emphasizes opcode sequence length, achieved a notable increase in malware detection performance employing the LSTM algorithm. The approach essentially reduces the sequences by creating a unique set of characteristics from the opcode sequence, therefore preserving dataset quality that finally results in an outstanding accuracy of up to 99.19%. In addition, this study presented the performance experiments of different methods for the second dataset, which contains 9 different classes of these methods: CNN, RNN, GRU, KNN, SVC, and LR, where the performance of the LR model was very satisfactory, achieving an accuracy of 98%, but the method proposed in this study remains the best, with an accuracy of 99.19%.

Table 7: The Hyperparameters For Model -2

Layer Name	Layer Type	Output Shape	Units	Return Sequences	Activation	Parameters
lstm_12	LSTM	(None, 1, 100)	100	Yes	tanh	54800
lstm_13	LSTM	(None, 1, 100)	100	Yes	tanh	80,400
lstm_13	LSTM	(None, 1, 100)	100	Yes	tanh	80,400
lstm_13	LSTM	(None, 1, 100)	100	Yes	tanh	80,400
dense_4	Dense	(None, 9)	9	No	ReLU → Softmax	909
Optimizer	Learning Rate	Batch Size	Epochs			
Adam	0.001	32	1000			

Table 8: Comparison of Results with Other Researches Using The Second Dataset

Methods	Accuracy (%)
[7] Jahromi, A.N.	99.1%
[8] Kang, J	97.59%
Our method (LSTM)	**99.19%**
CNN	90.58%
RNN	46.61%
GRU	96.03%
KNN	91.3%
SVC	83.33%
LR	98 %

Table 9: Classification Result of Model-2

	Precision	recall	f1-score	Support
0	0.95	0.98	0.96	445
1	1.00	1.00	1.00	724
2	1.00	1.00	1.00	908
3	1.00	0.93	0.97	136
4	0.86	1.00	0.99	6
5	1.00	0.99	0.99	246
6	0.99	0.96	0.98	109
7	0.99	0.96	0.97	374
8	0.99	0.98	0.99	313
Accuracy	0.99	0.98	0.99	3261
Macro avg	0.99	0.98	0.99	3261
Weighted Avg	0.98	0.98	0.98	3261

Table 9 shows the classification results for the second dataset, Model 2 classification results, together with weighted average, macro average, and accuracy. These outcomes capture the general efficiency and quality of the classification model. Figures 20 and 21 show Model 2 accuracy and loss curves for the second data set. While the loss curve shows how the decrease in loss helped to increase the accuracy of the model during development, the accuracy curve shows how

Journal of Theoretical and Applied Information Technology
15st June 2026. Vol.104. No.11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

well the model matched actual values with expected values. The confusion matrix in Figure 22 shows really good model quality.

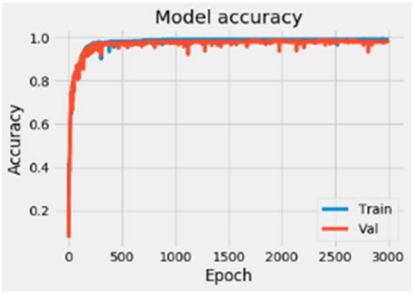


Figure 20. Accuracy curve of Model-2

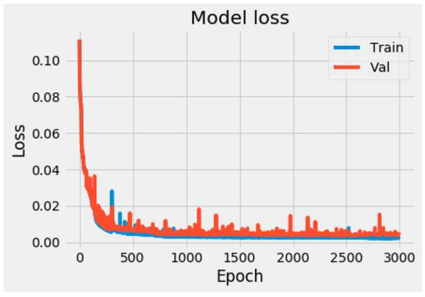


Figure 21. Loss curve of Model-2

Figure 22. Confusion Matrix of Model- 2

This study has calculated ROC curves and AUC scores for LSTM Model-2 to discriminate between nine classes. The evaluation scores for the model's performance using ROC curves and AUC scores were as follows: class 0 = 0.99, class 1 = 1.00, class 2 = 1.00, class 3 = 1.00, class 4 = 0.97, class 5 = 1.00, class 6 = 1.00, class 7 = 1.00, and class 8 = 1.00. This evaluation confirmed the good performance of Model 2 with the second dataset, as shown in Figure 23.

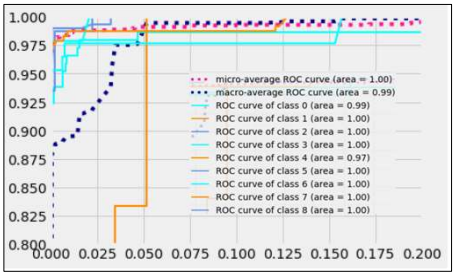


Figure 23. ROC AUC Curve of Model- 2

4.7 Third dataset

The table shows 10 hyperparameters for Model -3. The third collection comes from the Drebin project and comprises 5,560 programs spanning 179 different malware families. This study only used a subset of 10 families instead of the whole list of 179 families like the related study in [9], though (Fakelnstaller, DroidKungFu, Plankton, Opfake, Ginmaster, Basebridge, Kmin, Geinimi, ADRD, and DroidDream). It proposed a new preprocessing method based on raw data as input to solve the challenges presented by long sequences and varying input lengths in serialization models. They split the sequence into N chunks and tallied the occurrences in each one in order to reduce sequence length. Table 12 shows that our proposed method improves model correctness over [9]. scoring 98.80% against 95.58%. With just 0.017 seconds, our method also clearly benefits training time, 0.78 minutes over 2.27 minutes, and greatly faster preprocessing for malware identification. In addition, this study presented the performance experiments of various methods applied to the third dataset, which includes 10 distinct classes: CNN, RNN, GRU, KNN, SVC, and LR. The performance of the GRU model was very satisfactory, achieving an accuracy of 96.18%, but the method proposed in this study remains the best, with an accuracy of 98.80%.

Table 10: The Hyperparameters For Model -3

Layer Name	Layer Type	Output Shape	Units	Return Sequences	Activation	Parameters
lstm_11	LSTM	(None, 1, 300)	300	Yes	tanh	423600
lstm_12	LSTM	(None, 1, 300)	300	Yes	tanh	721200
lstm_13	LSTM	(None, 1, 300)	300	Yes	tanh	721200
dense_4	Dense	(None, 10)	100	No	ReLU → Softmax	3010
Optimizer	Learning Rate	Batch Size	Epochs			
Adam	0.001	32	1000			

Table 11: Comparison Results with Other Researches Using The Third Dataset

Methods	Accuracy	Training Time	Detection Time	Dataset (family)
[9] Chen	95.58%	2.27 m	27.34 s	10
Our method (LSTM)	98.80%	0.78 m	0.017 s	10
CNN	95.80%	1.68 m	0.0185 s	10
RNN	70.25%	1.36 m	0.0132 s	10
GRU	96.18%	1.61 m	0.023 s	10
KNN	84.10%	-	-	10
SVC	85.04%	-	-	10
LR	86.23%	-	-	10

Table 12 shows Model-3 classification report on the third dataset together with weighted average, macro average, and accuracy. These findings capture the general efficacy and quality of the classification performance of the model.

Table 12: Classifications Results of Model-3

	Precision	recall	f1-score	Support
1	1.00	1.00	1.00	57
2	1.00	1.00	1.00	137
3	0.99	1.00	1.00	135
4	1.00	0.99	0.99	98
5	0.97	1.00	0.99	73
6	0.88	0.78	0.82	9
7	0.99	0.99	0.99	77
8	0.98	0.98	0.99	43
9	0.93	0.96	0.95	27
10	1.00	0.98	0.95	11
Accuracy	0.99	0.98	0.99	3261
Macro avg	0.99	0.98	0.99	3261
Weighted Avg	0.98	0.98	0.98	3261

Figures 24 and 25 show, for the third dataset, the accuracy and loss curves for Model-3. Whereas the loss curve shows how decreasing loss helped to improve the model's optimization and accuracy, the accuracy curve shows the capacity of the model to match expected values with actual values. Furthermore, indicating good model quality is the classification performance shown by the confusion matrix in Figure 26.

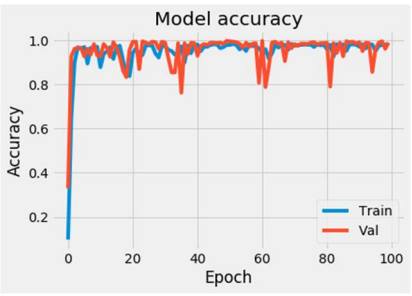


Figure 24. Accuracy Curve of Model- 3

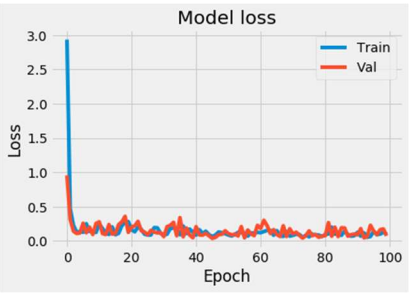


Figure 25. Loss curve of Model-3

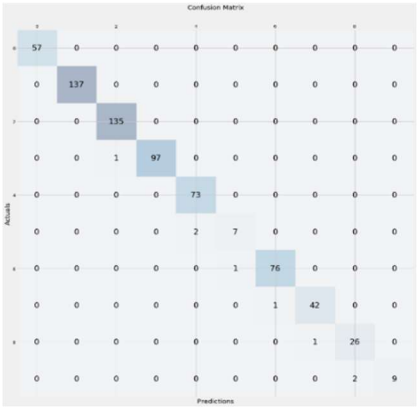


Figure 26. Confusion Matrix of Model- 3

This study has calculated ROC curves and AUC scores for LSTM Model-3 to discriminate between nine classes. The evaluation scores for the model's performance using ROC curves and AUC scores were as follows: class 0 = 0.99, class 1 = 1.00, class 2 = 1.00, class 3 = 1.00, class 4 = 1.00, class 5 = 0.94, class 6 = 1.00, class 7 = 0.98, class 8 = 0.99, and class 9 = 0.94. This evaluation confirmed the good performance of Model 3 with the second dataset, as shown in Figure 27.

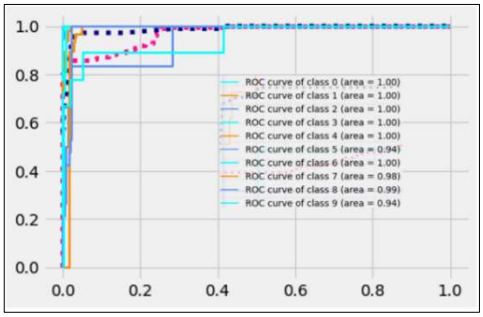


Figure 27: ROC AUC curve of Model-3

4.8 Fourth dataset

Derived from the Drebin dataset, which has 5,560 programs spanning 179 malware families, the fourth dataset also In this instance, the related research [9] used a larger subset comprising 24 families (Fakelnstaller, GinMaster, FakeDoc, ExploitLinuxLotoor, SendPay, Yzhc, DroidKungFu, BaseBridge, Geinimi, MobileTx, Gappusin, Jifake, Plankton, Iconosys, Adrd, Glodream, Imlog, Hamob, Opfake, Kmim, DroidDream, FakeRun, SMSreg, Boxer). We followed the same selection of malware families for consistency and comparison as in the preceding study. Table 13 The hyperparameters for Model 4, and Table 14 shows that, with better model accuracy (96.56%) than [9] (79.54%), our suggested approach additionally demonstrates a notable improvement in training duration, 1.5 minutes against 31.9 minutes in [9], as well as faster preprocessing for malware detection with a time of just 0.017 seconds. In addition, this study presented the performance experiments of various methods applied to the third dataset, which includes 10 distinct classes: CNN, RNN, GRU, KNN, SVC, and LR. The performance of the GRU model was very satisfactory, achieving an accuracy of 93.65%, but the method proposed in this study remains the best, with an accuracy of 98.80%.

Table 13: The Hyperparameters for model- 4

Layer Name	Layer Type	Output Shape	Units	Return Sequences	Activation	Parameters
lstm_11	LSTM	(None, 1, 300)	300	Yes	tanh	410400
lstm_12	LSTM	(None, 1, 300)	300	Yes	tanh	721200
lstm_13	LSTM	(None, 1, 300)	300	Yes	tanh	721200
dense_4	Dense	(None, 24)	100	No	ReLU → Softmax	7224
Optimizer	Learning Rate	Batch Size	Epochs			
Adam	0.001	32	1000			

Table 14: Comparison Results with Other Researches Using Fourth

Methods	Accuracy	Training Time	Detection Time	Dataset (Family)
[9] Chen Y.M., C.H.	79.54%	31.9m	20.4 s	24
Our model	96.56%	1.5m	0.017s	24
CNN	85.71%	1.29 m	0.0114s	24
RNN	60.43%	3.18m	0.028s	24
GRU	93.65%	1.40m	0.0126s	24
KNN	80.5 %	-	-	24
SVC	70.45%	-	-	24
LR	75%	-	-	24

Applying our suggested strategy to the fourth dataset, which represents the highest number of classes (24 malware families) in this study, shows clear superiority over earlier approaches utilizing the same dataset, according to the results displayed in Table 14. Our method greatly improves the performance of the LSTM algorithm in malware detection. This is confirmed by the classification report Table 15.

Table 15: Classification Results of Model-4

	Precision	recall	f1-score	Support
1	1.00	1.00	0.99	69
2	0.99	0.99	0.99	98
3	0.95	0.97	0.90	36
4	0.93	0.88	1.00	16
5	0.97	0.97	0.97	30
6	0.99	0.99	0.99	127
7	1.00	0.99	0.99	94
8	0.00	0.00	0.00	0
9	0.93	0.93	0.93	15
10	0.99	0.99	0.99	148
11	0.92	1.00	0.896	14
12	0.92	0.94	0.94	22
13	0.94	0.88	0.91	17
14	1.00	0.88	0.93	8
15	0.50	1.00	0.67	1
16	1.00	0.99	0.99	76
17	0.97	1.00	0.99	36
18	0.90	0.90	0.90	10
19	0.97	0.97	0.97	37
20	1.00	0.85	0.89	5
21	0.00	0.00	0.00	0
22	0.00	0.00	0.00	1
23	0.00	0.00	0.00	0
24	1.00	0.07	0.12	15
Accuracy				875
Macro avg	0.79	0.75	0.75	875
Weighted Avg	0.98	0.96	0.96	875

Classification report on the fourth dataset, together with weighted average, macro average, and accuracy. Particularly in relation to other studies using the same dataset but obtaining reduced accuracy, the results show the great classification performance of the model. Figures 28 and 29 show the Model 4 accuracy and loss curve for the fourth dataset's values. Whereas the loss curve shows how the decrease in loss helped to maximize the accuracy of the model, the accuracy curve shows how well the model aligned expected values with actual values. Furthermore, displaying robust and consistent results is the classification performance given in the confusion matrix in Figure 30.

Journal of Theoretical and Applied Information Technology
15st June 2026. Vol.104. No.11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

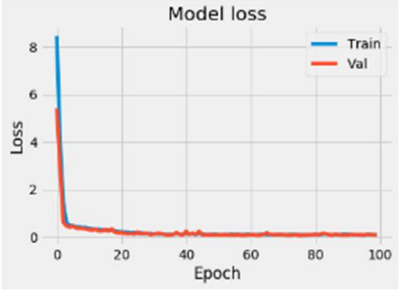


Figure 28. Accuracy Curve of Model-4

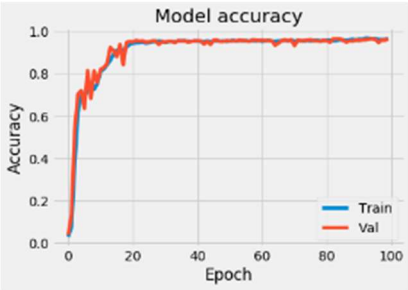


Figure 29. Loss Curve of Model-4

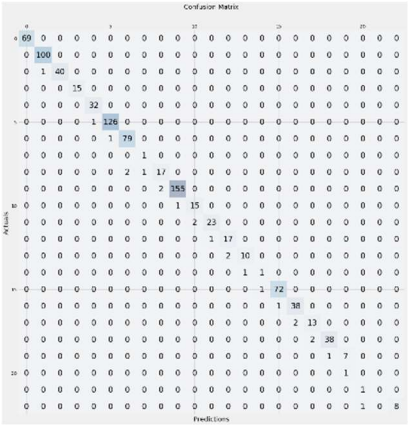


Figure 30. Confusion Matrix of Model-4

This study presented ROC curves and AUC scores as metrics to evaluate the fourth model of LSTM in distinguishing 24 classes. The results showed the model's amazing accuracy in distinguishing between the different classes. Classes 0 to 6, 8 to 19, 23, and 24 all achieved an AUC of 1.00, while classes 7, 20, 21, and 22 showed only a slightly lower but still outstanding AUC of 0.99. Through these evaluations, the model's exceptional ability to distinguish between the 24 classes was demonstrated. These results reinforce the superior accuracy of Model 4 on the fourth dataset, as shown in figure 31.

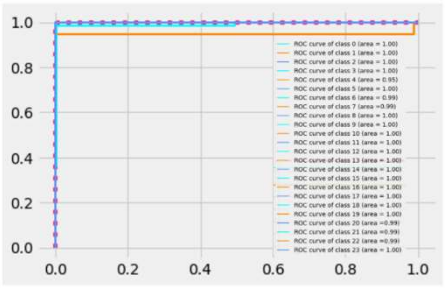


Figure 31. ROC AUC Curve of Model-4

This study presented a solution to the limitations of the LSTM method in dealing with long opcode sequences by reducing the sequence based on finding a subsequence based on encoding the unique values for the original sequence. The experimental results proved that the proposed method was highly performant. To generalize the proposed technology, other best-in-class methods for reducing opcode sequences were added to increase the scope and resilience of the model. These methods include autoencoder, which compresses the original sequence while preserving key features; embedding, which represents the code in a lower-dimensional vector space; and padding, which ensures sequence homogeneity for efficient processing within the model, in addition to other methods, such as compressing sequences. Slide windows in chunks, hot encode, autoencoder, and compress the original opcode sequence. Furthermore, the model developed in this study is flexible enough to be applied in a variety of settings by generalizing it with different sequence reduction or representation strategies. This includes the potential to apply the practical approach proposed in this study for detecting malware in a wider context.

Table 16 presents the generalization of the proposed method in this study, alongside the most widely used techniques for sequence shortening and dimensionality reduction applied in previous research on opcode-based malware detection. The proposed approach was evaluated against these established methods, and the results were compared. The findings demonstrate that our method is more efficient and offers greater generalizability.

Table 16: Application of Our Proposal on Methods of Previous Works in Reducing Opcode Sequencies

RF	The method of detection	Type dataset	A technique -solve long opcode sequence	Accuracy of RF	Accuracy of RF methods
[10]	CNN	Opcode	Embedding	98.18%	99.35%
[11]	CNN	Opcode	Embedding	95.5%	98.33%
[12]	DG-CNN	Opcode	Embedding	96%	99.35%
[13]	GEMAL	Opcode	Embedding	99.81%.	100%
[14]	ML	Opcode	Embedding	99.99%.	100%
[15]	CNN	Opcode	Embedding	95%	99.35%
[24]	CNN	Opcode	Embedding	95%	99.35%
[16]	LSTM	Opcode	Embedding	98.35%	99.35%
[17]	CNN-ML learning	Opcode	autoencoder	99%	99.35%
[18]	Xgboost	Opcode	autoencoder	98.2%	99.6%
[19]	LSTM	Opcode	autoencoder	96%	100%
[8]	LSTM	opcode	hot encoding	97.59%	100%
[20]	LSTM	Opcode	autoencoder	98.14%	100%
[21]	LSTM	Opcode	compresses sequence	98.39%	99.34%
[9]	LSTM	Opcode	Windows N chunk	95.58%	96.56%
[6]	LSTM	Opcode	opcode window lengths	99.10%	100%
[4]	LSTM	Opcode	Padding technique	98.18%	100%
[22]	LSTM	Opcode	Padding technique	97.52%	100%
[23]	LSTM	Opcode	Padding technique	97.2%	100%
[24]	LSTM	Opcode	Padding technique	96.6%	100%
	CNN	Opcode	Padding technique	98.5%	99.35%
[25]	CNN	Opcode	Padding technique	98.89%	99.35%
[26]	CNN	Opcode	Padding technique	99%	99.35%

The proposed method exhibits consistently high performance across all datasets evaluated as shown in Figure 17 with MCC values ranging from 0.9750 to 0.9896 and low standard deviations, suggesting stable behavior of the model. The paired t-test results also provide further evidence for the statistical significance of the observed improvements across all conditions. Large t-statistics (11.66–16.23) and very small p-values (≤ 0.0003) support this conclusion. The results strongly suggest that the observed improvements in performance are unlikely to be due to chance.

Table 17: The Statistical Rigor of Performance Comparisons

Model	MCC (mean ± std)	t-statistic	p-value	Interpretation
Model-1 First dataset	0.9896 ± 0.0127	16.2301	0.0001	Statistically significant improvement
Model-2 Second dataset	0.9880 ± 0.0009	12.7969	0.0002	Statistically significant improvement
Model-3 Third dataset	0.9885 ± 0.0021	11.9168	0.0003	Statistically significant improvement
Model-4 Fourth dataset	0.9750 ± 0.0025	11.6593	0.0003	Statistically significant improvement

5. DISCUSSION

The results indicate that the proposed approach improves both classification accuracy and computational efficiency by preserving essential opcode patterns while reducing redundancy. Although LSTM models have shown outstanding effectiveness in malware classification using opcode sequences, their use in IoT environments requires consideration of the merging computation, latency, and footprint. While LSTMs might be more expensive than traditional classifiers optimizations like model pruning, quantization, and reduction of sequence length can reduce the memory footprint significantly.

Which was suggested by this study and reached excellent results compared to other research. We expect that after the frequent model size reductions, the LSTM model will be fit for mid-tier IoT gateways or edge nodes. While ultra-constrained sensor nodes will have to offload processing to nearby gateways, runtime performance on ultra-lightweight CPUs or NPUs should still be acceptable on some devices, operating in the range for average opcode sequence length. Practically, the model can be deployed in conjunction with networked intrusion detection systems, lightweight anomaly detection, or signature-based scanners in a multi-layered defense architecture. Due to this hybrid approach, opcode-based LSTM detection can be safely adopted for IoT environments, as the sensor nodes have resource and performance constraints.

6. CONCLUSION

The count of IoT systems is rising constantly. Simultaneously, malware creators are becoming more adept and producing harmful programs that are difficult for conventional defense measures to find. This emphasizes how urgently strong detection systems able to spot both modern and emerging malware threats are needed. Despite the effectiveness of LSTM in malware detection, handling long opcode sequences remains a key challenge that affects model efficiency.

The present work presented a fresh method to solve this by rebuilding opcode sequences into subsets depending on certain characteristics, hence shortening their length. Furthermore, a padding strategy was used to manage changing sequence lengths following reduction. The results unequivocally show the efficiency of the proposed technique in terms of both accuracy and processing time. This dual-step process effectively addresses the challenge of sequence length while preserving data quality and enabling the model to process sequences of different sizes in detecting malicious patterns and distinguishing malware accurately, thus differentiating our work from previous research. This verified that our suggested approach, together with the LSTM algorithm is quite successful for virus detection in IoT systems.

This study proposes future work on the scalability of the model for generalization to large-scale IoT environments. The IoT often generates massive and heterogeneous data flows, which call for performance evaluation in such conditions. This involves using a large dataset of different types of devices under different network conditions to identify potential bottlenecks. From here, improvement strategies such as edge computing integration, distributed processing, and incremental learning can be found. By enhancing accuracy and efficiency at scale, this model becomes more reliable and suitable for deployment in complex real-time IoT infrastructures.

ACKNOWLEDGMENT

This research is supported by Universiti Putra Malaysia.

REFERENCES

.[1] F. Meng, "An effective network attack detection method based on kernel PCA and LSTM-RNN," *International Conference on Computer Systems, Electronics and Control (ICCSEC). 2017. IEEE,* 2017.

[2] S. Prakash, A.S. Jalal, and P. Pathak, " Forecasting covid-19 pandemic using prophet, lstm, hybrid gru-lstm, cnn-lstm, bi-lstm and stacked-lstm for india," *International Conference on information systems and computer networks (ISCON). 2023. IEEE.,* 2023.

[3] J. Yan, Y. Qi, and Q. Rao, , "Detecting malware with an ensemble method based on deep neural network," *Security and Communication Networks,* vol. 2018(1): p. 7247095, 2018.

[4] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "A deep Recurrent Neural Network based approach for Internet of Things malware threat hunting," *Future Generation Computer Systems,* vol. 85, pp. 88-96, 2018/08/01/ 2018, doi: https://doi.org/10.1016/j.future.2018.03.007.

[5] Darabian, "A multiview learning method for malware threat hunting: windows, IoT and android as case studies," *World Wide Web,* vol. 23, pp. 1241-1260, , et al.,2020.

[6] D. Carlin, "Towards the Detection of Malware in the Internet of Things at Run-Time," in *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*, 6-8 Jan. 2025 2025, pp. 00552-00560, doi: 10.1109/CCWC62904.2025.10903744.

[7] A. Jahromi, S. Hashemi, A. Dehghantanha, R. Parizi, and K.-K. R. Choo, "An Enhanced Stacked LSTM Method With No Random Initialization for Malware Threat Hunting in Safety and Time-Critical Systems," *IEEE Transactions on Emerging Topics in Computational Intelligence,* vol. PP, pp. 1-11, 06/22 2020, doi: 10.1109/TETCI.2019.2910243.

[8] J. Kang, S. Jang, S. Li, Y.-S. Jeong, and Y. Sung, "Long short-term memory-based Malware

classification method for information security," *Computers & Electrical Engineering,* vol. 77, pp. 366-375, 07/01 2019, doi: 10.1016/j.compeleceng.2019.06.014.

[9] Y. M. Chen, C. H. Hsu, and K. C. K. Chung, "A novel preprocessing method for solving long sequence problem in android malware detection," in *2019 Twelfth International Conference on Ubi-Media Computing (Ubi-Media)*, 2019: IEEE, pp. 12-17.

[10] Q. Wang and Q. Qian, "Malicious code classification based on opcode sequences and textCNN network," *Journal of Information Security and Applications,* vol. 67, p. 103151, 2022.

[11] J. D. R. N McLaughlin, "Data augmentation for opcode sequence based malware detection
" *Cyber Research Conference-Ireland (Cyber-RCI), 2022•ieeexplore.ieee.org,* 2022.

[12] G. Y. J Yan, D Jin "Classifying malware represented as control flow graphs using deep graph convolutional neural network," *annual IEEE/IFIP international ..., 2019 - ieeexplore.ieee.org,* 2019.

[13] Y. W. XW Wu, Y Fang, P Jia, "Embedding vector generation based on function call graph for effective malware detection and classification," *Neural Computing and Applications,•Springer,* 2022.

[14] F. J. X Han, R Wang, S Wang, Y Yuan, "Classification of malware for self-driving systems," *Neurocomputing, 2021•Elsevier,* 2021.

[15] J. L. Y Huang, X Xiang, P Wen, S Wen, Y Chen, L Chen, Y Zhang, "Malware identification method in industrial control systems based on Opcode2vec and CVAE-GAN," *mdpi,* 2024.

[16] M. A. E Snow, A Glandon, K Iftekharuddin, "End-to-end multimodel deep learning for malware classification," *International Joint Conference on Neural Networks (IJCNN), 2020,* 2020.

[17] C. M. D Gibert, J Planes, "An end-to-end deep learning architecture for classification of malware's binary content," *International Conference on Artificial Neural Networks,* 2018.

[18] V. V. M Yousefi-Azar, L Hamey, U Tupakula, "Autoencoder-based feature learning for cyber security applications," *International Joint Conference on Neural Networks (IJCNN), 2020,* 2017.

[19] Y. Cho, "Dynamic RNN-CNN based malware classifier for deep learning algorithm," *International Telecommunication Networks,* 2019.

[20] S. J. Y Sung, YS Jeong, J Hyuk, "Malware classification algorithm using advanced Word2vec-based Bi-LSTM for ground control stations," *Computer Communications,* 2020.

[21] Y.-M. Chen, A.-C. He, G.-C. Chen, and Y.-C. Liu, *Android malware detection system integrating block feature extraction and multi-head attention mechanism*. 2020, pp. 408-413.

[22] X. L. J Sun, H Gao, W Wang, Y Gao, X Yang, "Categorizing malware via a Word2Vec-based temporal convolutional network scheme," *Journal of Cloud Computing,* 2020.

[23] J. D. P Wyrwinski, C Jedrzejek, "Ensemble malware classification using neural networks," *International conference on multimedia communications, services and security,* 2020.

[24] F. Z. H Rana, F Raoof, A Zahoor, "A Prediction of Network Intrusion Using CNN-LSTM: Hybrid Approach By Deep Learning For Prediction of Network Intrusion," *KIET Journal of Computing and Information Sciences,* 2024.

[25] P. V. V Anandhi, VG Menon, "Malware visualization and detection using DenseNets," *Personal and Ubiquitous Computing,* 2024.

[26] M. A. A Rahali, "Malbertv2: Code aware bert-based model for malware identification," *Big Data and Cognitive Computing,,* 2023.