Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:
https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/index
ISSN: 2462-1943

Convolutional Long Short-Term Memory for Fileless Malware Detection

Kunaprasan Kareegalan[1], Aziah Asmawi[1,*], Mohd Taufik Abdullah[1], Mohd Izuan Hafez Ninggal[1], Muhammad Daniel Hafiz Abdullah[1] , Yousif Raad Muhsen[2,3]

[1] Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia
[2] Department of Civil, college of engineering, Wasit University, Wasit 52001, Iraq
[3] Technical Engineering College, Al-Ayen University, ThiQar 64001, Iraq

ARTICLE INFO

ABSTRACT

In cybersecurity, the rise of fileless malware poses a significant challenge to endpoint security. Traditional detection methods often fail against these sophisticated attacks, necessitating advanced techniques like deep learning models. This study highlights the limitations of Bi-Directional Long Short-Term Memory (BLSTM) models in dynamic malware analysis and proposes enhancements through Convolutional Long Short-Term Memory (ConvLSTM) architecture. BLSTM models process input sequences in forward and backward directions, combining the results into one output. While this dual-layer approach improves analysis, it is time-consuming, potentially increasing the risk of fileless malware attacks. A key limitation of BLSTM is the lack of parameter sharing between forward and backward directions. This reduces its ability to capture spatial and temporal features simultaneously, hindering effectiveness in detecting fileless malware. To address this, the ConvLSTM model consolidates feature extraction within a single LSTM cell layer. ConvLSTM breaks down samples into subsequence and uses timesteps for additional feature extraction, enabling spatial-temporal data analysis and improving malware prediction accuracy. The model was tested using a dynamic malware dataset. Unlike traditional LSTM, ConvLSTM integrates convolutional layers, allowing parameter sharing across both spatial and temporal dimensions. This reduces computational complexity and improves model performance in handling multidimensional data. The research re-simulated prior work with BLSTM using the same malware dataset. The Spyder app ran the event simulator and the ConvLSTM model's results replaced BLSTM's using identical parameters. Time, accuracy and loss were the main performance metrics. ConvLSTM outperformed BLSTM, achieving 98% detection accuracy compared to BLSTM's 90%. It also significantly reduced processing time, averaging 10 seconds, while BLSTM took 22 seconds. ConvLSTM experienced lower losses, averaging 10% per epoch versus BLSTM's 20%. In conclusion, ConvLSTM offers superior performance over BLSTM in fileless malware detection. Its enhanced computational efficiency and ability to quickly mitigate threats make it a robust solution for fortifying endpoint security against evolving cyber threats. ConvLSTM holds potential in strengthening defence mechanisms against sophisticated malware attacks, providing a proactive approach to safeguarding networks and data.

Keywords:

Cybersecurity; fileless malware; endpoint security; dynamic malware; ConvLSTM; RNN

---

[*] *Corresponding author.*
E-mail address: a_aziah@upm.edu.my

https://doi.org/10.37934/araset.64.4.136157

1. Introduction

In today's digital landscape, fileless malware [11] has become a significant threat to endpoint security. Unlike traditional malware, fileless malware operates within a system's volatile memory, exploiting legitimate processes and applications to execute malicious activities, making it difficult to detect with conventional antivirus solutions. The rise of such attacks is driven by the growing sophistication of cybercriminals who exploit vulnerabilities in software and hardware, targeting various endpoints like desktops, laptops, mobile devices and servers.

The stealthy nature of fileless malware, often exhibiting polymorphic and obfuscated characteristics, complicates detection and analysis. This research digs into the mechanisms and behaviours of fileless malware, using dynamic malware analysis to uncover vulnerabilities and develop effective mitigation strategies. Through real-world attack scenarios and studying the tactics, techniques and procedures (TTPs) of fileless malware, the study aims to provide organizations with actionable insights and best practices for defending against these threats.

Dynamic malware analysis, which involves the live execution of malware in a controlled environment, is explored in-depth. This approach allows security analysts to observe the behaviour of malware, detect evasion techniques and develop countermeasures to protect digital assets and networks. The study aims to advance fileless malware detection and prevention, equipping organizations with the knowledge and tools needed to stay ahead of cyber adversaries.

1.1 Problem Statement

Fileless malware [16] has evolved to the point where it can now target the present network infrastructure without being detected by any security measures, despite Traditional Malware having been extinct. Attackers inject malicious payload directly into the memory (RAM) by using JavaScript to run malware code in the browser without utilizing the common legitimate tools and applications. This can cause zero-day attacks since that codes and programs can be manipulated easily. To mitigate that, a Dynamic Malware Analysis was handled to gather data about malware behaviour which runs on an isolated virtual environment to execute the suspicious code captured from real users to test any impact on the host or not by using Sandboxing.

Cuckoo Sandbox is an open-source tool that is customizable to run Dynamic Malware Analysis by using Deep Learning models which is Long Short-Term Memory (LTSM) for classification of malware. LTSM is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. This solution analyses past and future malware behaviour sequences which classify the possible zero-day attacks. Despite the success achieved using the LTSM models based on Deep Learning Dynamic Malware Analysis such as the Bidirectional LSTM (BLSTM) model used [17], there are still several other challenges that remain unresolved.

These include:

i. The BLSTM model reads input sequences in both forward and backward directions, combining the analysed results into an output. This is achieved by combining two LSTM cells with opposite timings to the same output, creating a dual layer between the sequences. However, this process consumes more time to complete the analysis, potentially increasing the likelihood of successful files malware attacks [17,30].
ii. The BLSTM model operates on spatial-temporal data structured in a 3D format: [samples, time steps, features]. It processes data both forward and backward using separate LSTM networks, but parameters are not shared between directions. This lack of parameter sharing may limit

the model's effectiveness in capturing spatial and temporal features simultaneously, potentially impacting its accuracy in understanding data across both dimensions especially in detecting fileless malware attacks [17,30].

1.2 Research Objective

The objectives of this study are:

i. To employ ConvLSTM Model which shortens the process of malware analysis and reduce the time taken to increase the number of malware detectable.
ii. To employ four-dimensional approach which able to run the samples into single sequence input and produce output in accurate and better results on the prediction.

1.3 The Proposed Solution

We have proposed ConvLSTM which is a type of artificial Recurrent Neural Network (RNN) architecture used in the field of deep learning to detect suspicious activity or any new traffic bypassing the network gate. We will be using the Sandboxing program to run and simulate the exact environment to analyse the existing solution and build a new prototype in this experimental environment.

We are implementing a machine learning technique with RNN architecture. ConvLSTM is a type of LSTM related where the convolutional reading of input is built directly into each LSTM unit under a Convolutional Neural Network (CNN) LSTM. The ConvLSTM was developed for reading two-dimensional spatial-temporal data but can be adapted for use with univariate time series forecasting. This solution is expected to handle the raw data in minimum amount usage of resource and increase the detection accuracy hitting real-time maximum. The overall framework of the ConvLSTM is depicted in Figure 1.

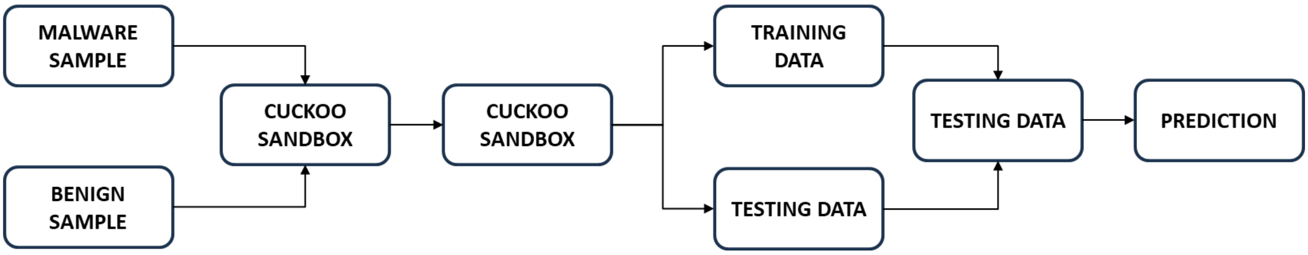

Fig. 1. Overall framework

The Figure 1 represents a typical flow for malware detection using a sandbox environment and machine learning.

i. Malware Sample / Benign Sample: These are the two categories of input samples being analysed. Malware samples represent known malicious software, while benign samples represent clean or non-malicious software.
ii. Cuckoo Sandbox: This is a dynamic malware analysis tool that runs malware or benign samples in a controlled environment. The sandbox observes the behaviour of the samples, capturing various system activities like file changes, network activity and system calls.

iii. Feature Extraction: Once the samples have been analysed by the Cuckoo Sandbox, important features are extracted from their behaviour. These features are metrics or data points that summarize the activity patterns of both benign and malware samples. For example, the number of file accesses or specific system calls can be features.
iv. Training Data: This is the dataset used to train the machine learning model. It consists of features extracted from malware and benign samples. The model learns the patterns and characteristics that distinguish malware from benign software.
v. Testing Data: This dataset is separate from the training data and is used to evaluate the model's performance. It ensures that the model can generalize and correctly predict unseen data.
vi. Prediction: After training, the machine learning model makes predictions on new or unseen testing data. It classifies whether a sample is malware or benign based on the extracted features.

In summary, this Figure 1 outlines the process of feeding malware and benign samples into a sandbox environment, extracting key features from their behaviour and using those features to train and test a machine learning model that predicts whether new samples are malicious or benign.

1.4 Research Contribution

By having the ConvLSTM as the solution in this research, it should be able to:

i. increase layers in an LSTM to capture progressively higher higher-level features in Deep Learning Dynamic Malware Analysis.
ii. increase handling spatiotemporal correlations for malware behaviour sequence.
iii. achieve more accuracy in malware behaviour detection on zero-day attacks.
iv. reduce overhead on the entire analysis proceed and increase the response time.

2. Related Works

2.1 Fileless Malware

Fileless malware is a risky threat because it is designed to be silent and hard to detect. Unlike traditional malware, which infects files or installs software on a computer's hard drive, fileless malware operates entirely in the computer's memory. This means it does not leave any traceable files or programs that antivirus software can easily identify. One technique fileless malware uses is exploiting legitimate tools and processes already present on the victim's computer, like PowerShell or Windows Management Instrumentation (WMI). These tools are part of the operating system and are intended for various tasks, but the malware repurposes them to execute malicious commands. By using these trusted tools, fileless malware can evade detection by antivirus programs that scan for known malware signatures [21-23].

Another method fileless malware uses is exploiting vulnerabilities in software or operating systems. These vulnerabilities are weaknesses in the code that attackers can exploit to gain unauthorized access to a system. Once the malware exploits a vulnerability and gains access to the computer's memory, it can execute its malicious code without leaving any files or programs behind. Fileless malware is often used in targeted attacks against organizations and individuals with valuable or sensitive information. Because it operates silently in the background without creating any obvious signs of infection, it can remain undetected for long periods, allowing attackers to steal data, spy on users or carry out other malicious actions without the victim's knowledge [21-23].

To protect against fileless malware, it's important to use comprehensive security measures beyond traditional antivirus software. This includes keeping software and operating systems up to date, using strong passwords and multi-factor authentication and regularly monitoring for unusual or suspicious activity on your network. Additionally, educating users about the risks of fileless malware and how to recognize potential threats can help mitigate the risk of infection [21-23].

2.2 Fileless Malware Types

Fileless malware includes several types of attacks as listed below:

i. <u>Memory-Based Attacks:</u> Malware stays only in the computer's RAM and doesn't write any files to disk, making it hard to detect [24].
ii. <u>Script-Based Attacks:</u> Attackers use scripting languages like PowerShell or JavaScript to run malicious commands directly in memory, bypassing security controls [25].
iii. <u>Registry-Based Attacks:</u> Malware alters the Windows registry to maintain control and execute code without leaving files on the disk, evading traditional antivirus [24].
iv. <u>Living-off-the-Land Attacks:</u> Attackers misuse legitimate system tools like PowerShell and WMI to execute malicious commands and avoid detection [25].
v. <u>Fileless Exploit Attacks:</u> These attacks exploit vulnerabilities in software or operating systems to run malicious code directly in memory, bypassing security controls [24].
vi. <u>Macro-Based Attacks:</u> Malicious macros in documents, like Word or Excel files, download and execute malware in memory, exploiting users' trust in these documents [25].

2.3 Deep Learning Techniques

Deep learning is a branch of machine learning within artificial intelligence (AI) that involves training artificial neural networks with large amounts of data to recognize patterns and make predictions. These models consist of layers of interconnected nodes (neurons) that process information hierarchically, allowing them to learn complex data representations automatically. Deep learning has been successful in fields like computer vision, natural language processing and speech recognition [26].

In cybersecurity, deep learning is valuable in combating fileless malware, a stealthy type of malware that operates entirely in a system's memory, evading traditional antivirus detection. Unlike conventional malware, which relies on files stored on disk, fileless malware uses legitimate system processes to execute its malicious activities, making it hard to detect with signature-based methods [26].

Deep learning offers a powerful solution by analysing large amounts of data to identify subtle patterns that signal malicious behaviour. These models can distinguish between legitimate and malicious processes, detecting fileless malware with high accuracy. Additionally, deep learning models can continuously improve and adapt to new threats, making them effective in the constantly evolving field of cybersecurity [26].

There are several types of deep learning techniques as listed below:

i. Convolutional Neural Networks (CNNs): CNNs are used for tasks like image classification and object detection. They process visual data through layers that detect various features, allowing them to understand images [26].
ii. Recurrent Neural Networks (RNNs): RNNs are designed for sequential data tasks, such as predicting time series or recognizing speech. They use connections that loop back to handle sequences and time-based data [26].
iii. Long Short-Term Memory (LSTM) Networks: LSTMs are a type of RNN that solves the problem of vanishing gradients in long sequences. They use memory cells and gates to manage and remember information over time [26].

2.4 Long Short-Term Memory (LSTM) Networks:

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture which been utilized in the deep learning field. Other than standard feed-forward neural networks, LSTM owns response connections. Not only it can process single data points like images but also the whole data sequences like speech or video. For example, LSTM is capable of handling tasks such as unsegmented, connected handwriting recognition, speech recognition and abnormal detection in network traffic or Intrusion Detection System (IDS) [11].

A usual LSTM unit is built of a cell, an input gate, an output gate and a forget gate. The cell keeps the values over arbitrary time intervals beside the 3 gates control the information flow into and out of the cell. LSTM networks are perfect to classify, process and make predictions based on time series data, since there are changes to be unknown duration lagging between key events in a time series [11].

The key components of an LSTM include:

i. Data Preparation
ii. Vanilla LSTM
iii. Stacked LSTM
iv. Bidirectional LSTM
v. CNN LSTM
vi. ConvLSTM

2.5 Bidirectional LSTM (BLSTM)

Bidirectional LSTM (BLSTM) is one of the deep learning models under malware detection. BSLTM is an advanced method of traditional LSTM that improved the performing model for sequence classification issues. Usually, almost all prediction and classification for time series data depend on Recurrent Neural Networks (RNN). The fundamental concept is the result of the previous and current will be using an input in the network and get the output and it will be repeated often. The network weight is controlled by Back Propagation Through Time (BPTT) algorithm [1-3].

The disadvantage of RNN is that it is a long-term reliance that will be forgotten, resulting in low predictive long-term memory, which is why Long Short-Term Memory is on the way [9]. BLSTM differs from traditional LSTM since it takes input from both bidirectional LSTM which is past and future feature extraction. There will be a forward and backward way of getting input and producing output but over here running simultaneous LSTM forward and backward to produce output consumes more time [1-3].

This requires more energy resources to produce outcomes. Perhaps, the aim of LSTM is to predict the accuracy of malware detection, but BLSTM seems to be a good option since it uses three-dimensional input to read the data and produce the output in 1 sequence. It can be observed that the BLSTM employs two LSTMs to provide a single result, one on forward for future or present extraction and the other on backward for previous extraction [10].

BLSTM model is running dual LSTM and at the same time, pushing multiple inputs into a single sequence and extraction output in a short time. It may produce output but measuring the pressure handled by the single process is high. It may make mistakes or take time to produce output since it is pushing the input metric which will be samples, time steps and features. Although it appears straightforward, running the application requires more time and energy [17].

The Figure 2 diagram shows the input taken to two layers which are from Forward Layer LSTM and Backward Layer LSTM and running the process in both directions and get analyse the output in the Activation layer and produce output [17].

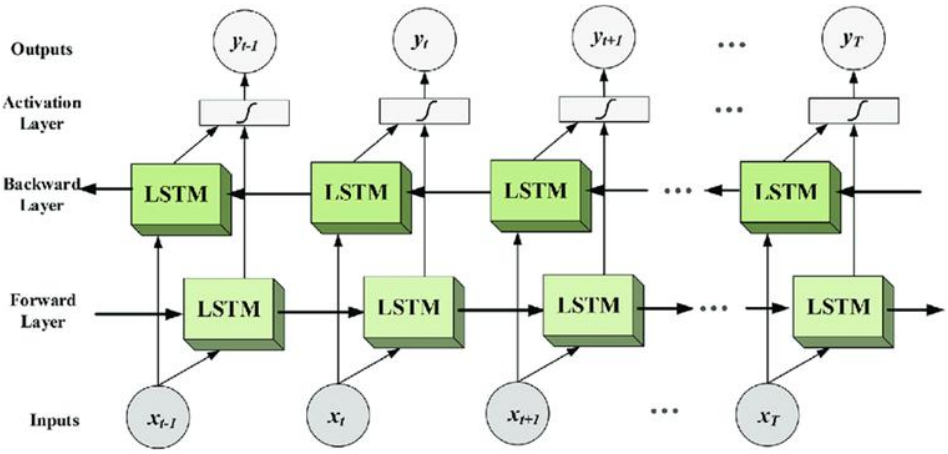

Fig. 2. Framework of BLSTM [17]

3. Methodology

We choose LSTMs are Recurrent Neural Network (RNN) types that are able to study long-term dependencies for example relationships distant between elements position of a sequence. They accomplish targets with complicated memory structures in LSTM cells which not included in traditional RNN cells. The matrices and vectors Wx, Ux and bx (with x ∈ {f, i, c, o}) in Figure 3 are the parameters θ of an LSTM [4].

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \\
f_t &= \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f) \\
o_t &= \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \\
\tilde{c_t} &= tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{c_t} \\
h_t &= o_t \odot tanh(C_t)
\end{aligned}
$$

Fig. 3. LSTM algorithm [4,5]

Same as traditional RNNs, LSTMs process every input in time *t* with the output from the previous timestep (*ht-1*). In extra, they add on a unit cell state (*C*) which holds on the entire sequence information. LSTMs bring to or extract minor information pieces from *C* through the operations in the forget (*ft*) and input (*it*) gates. The latest C shows the event of history utilized to compute the output *ht* by filtering C out with the output gate *ot*. This architecture gives LSTMs the space to relate the latest events with past events in sequence and arrange them to match to detect abnormal threat activities [4].

There are various types of models such as BLSM, CLSTM, CONV-LSTM, MULTI-LSTM and more. Other than LSTM, there are also other artificial models for study purposes or real industrial projects that are being daily testing. This will be the right choice for me to test for malware threat detection. Our aim to choose LSTM model is to emphasize the importance of Artificial Intelligence + Malware Detection can be used instead of using an Endpoint Detection tool for behaviour detection [4].

3.1 Research Framework

The research framework is shown in Figure 4, showing the sequential order the steps follow. These phases include problem formulation, a reimplementation of previous work proposed protocols designs, experiments and then performance evaluations with comparisons compared to the improper protocols.

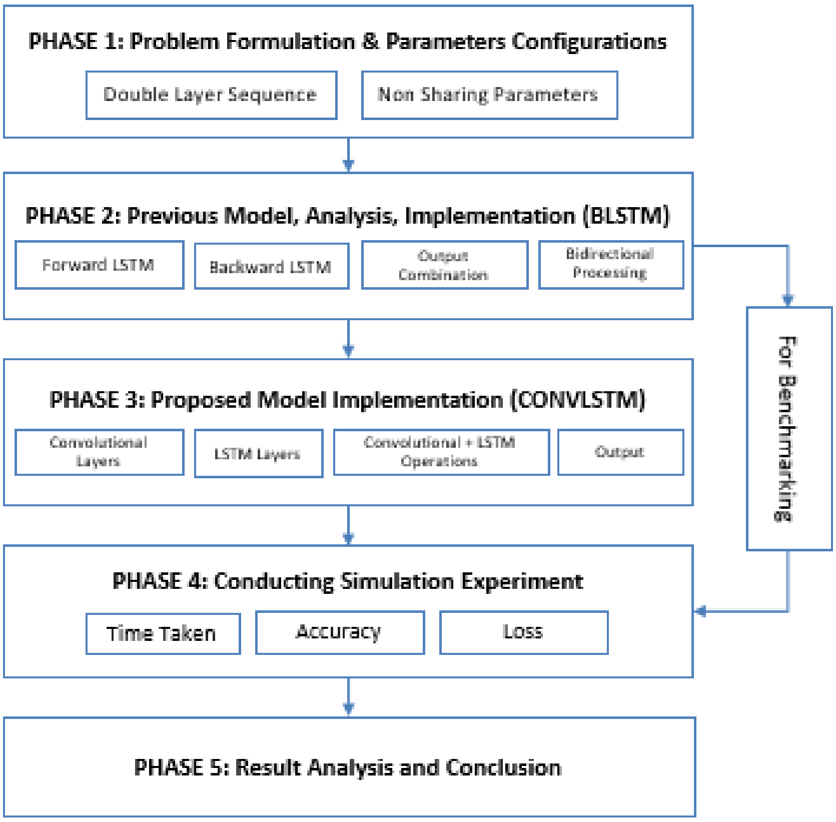
PHASE 1: Problem Formulation & Parameters Configurations
Double Layer Sequence
Non Sharing Parameters
PHASE 2: Previous Model, Analysis, Implementation (BLSTM)
Forward LSTM
Backward LSTM
Output Combination
Bidirectional Processing
PHASE 3: Proposed Model Implementation (CONVLSTM)
Convolutional Layers
LSTM Layers
Convolutional + LSTM Operations
Output
PHASE 4: Conducting Simulation Experiment
Time Taken
Accuracy
Loss
PHASE 5: Result Analysis and Conclusion
For Benchmarking

Fig. 4. Framework of the research

3.2 Dataset

To test the performance of ConvLSTM on fileless malware detection, we have conducted an experiment that will be described in this section. In our research, we have gained a big number of samples which have been used from previous work obtained from Virustotal [27]. The dataset consists of 13518 malicious samples and 7860 no malicious samples. These malware samples are gained from Virus Share and the non-malware sample are gained from Windows Operating

systems such as XP, Windows 7, and Windows 10.
The performance of ConvLSTM on fileless malware detection was tested through an experiment described in this section. In this research, the dataset is sourced from the GitHub repository "lstm_malware_detection" by Ocatak [28]. The repository provides scripts and data for implementing and testing LSTM models in the domain of malware detection. The dataset is specifically tailored for cybersecurity, focusing on the detection of malware, potentially including fileless malware, which is challenging to detect using traditional methods. he types of malicious malware included in the dataset are Adware, Backdoor, Downloader, Dropper, spyware, Trojan, Virus and Worm [28].

i. Data Collection: A diverse and representative dataset of malicious samples is obtained, covering various types of malwares, different attack vectors and potentially different stages of malware execution (e.g., static analysis, dynamic analysis). The dataset comprises 13,518 malicious samples and 7,860 non-malicious samples. These malware samples were collected from Virus Share, while the non-malware samples were obtained from Windows Operating Systems such as XP, Windows 7 and Windows 10 [28].
ii. Data Preprocessing: The dataset is pre-processed to ensure uniformity and cleanliness. Tasks such as removing duplicates, standardizing file formats, extracting relevant features and balancing the dataset if there are class imbalances are carried out. The hash values of each malware that been analysed were determined and these hash values were queried with the VirusTotal service. The analysis results from the VirusTotal service were stored into a database. Consequently, each malware underwent analysis by multiple antivirus engines and the outcomes of these analyses were recorded [28].
iii. Data Classification: Malicious and benign samples are categorized based on their characteristics and behaviour. This step ensures that the dataset is properly labelled for subsequent analysis. Our malware classification method, which is LSTM, is implemented at this stage [28].
iv. Partitioning: The dataset is divided into three subsets: training, validation and test.
    - Training Data: This subset is utilized to train the machine learning model. It has comprised the majority of the dataset, typically around 60-80%. The model learned from this data to recognize patterns and characteristics of malicious behaviour [29].
    - Validation Data: This subset is used to tune hyperparameters and evaluate model performance during training. It assisted in preventing overfitting by providing an independent dataset for validation. It usually consists of around 10-20% of the dataset [29].
    - Test Data: This subset is used to evaluate the final performance of the trained model. It serves as an unseen dataset to assess how well the model generalizes to new, unseen malicious samples The test set has been completely separated from the training and validation data and typically comprises the remaining 10-20% of the dataset [29].
v. Random Sampling: Instances are randomly sampled from the dataset to ensure that each subset (training, validation, test) is representative of the overall dataset. This helps prevent biases in the model training and evaluation [28].

3.3 Spatial Temporal Data Presentation

To present a dataset as spatial-temporal data, each sample in the dataset needs to be structured to capture both spatial and temporal aspects of the underlying phenomenon. In the context of malware detection, spatial features refer to characteristics extracted from a single point in time, such as static features derived from file properties or code structure. Temporal features, on the other hand, capture how these characteristics evolve over time, such as behavioural patterns observed during the execution of malware. Here is how the dataset can be structured to represent spatial-temporal data for malware detection:

i. Spatial Features
    - Static features extracted from each sample at a single point in time.
    - These features can include file metadata (e.g., file size, file type), header information, opcode sequences, API call frequencies and static analysis results (e.g., presence of specific strings, signatures).
    - Each sample is represented as a vector of static features.

ii. Temporal Features
    - Dynamic features capturing the behaviour of the malware over time.
    - These features are extracted during the execution of the malware and represent how its behaviour evolves.
    - Examples of temporal features include system call sequences, network traffic patterns, memory usage, registry modifications and any other dynamic behaviour observed during runtime.
    - Each sample is represented as a sequence of temporal feature vectors, where each vector corresponds to a specific time step or observation window during the execution of the malware.

iii. Spatial-Temporal Representation
    - Combine spatial and temporal features to create a unified representation for each sample.
    - This can be achieved by concatenating the static features with the corresponding temporal feature sequences.
    - Alternatively, spatial and temporal features can be fed into separate branches of a neural network model, allowing the model to learn spatial and temporal patterns independently before merging the representations at later stages.

iv. Dataset Structure
    - Each sample in the dataset is represented as a structured data point containing both spatial and temporal information.
    - The dataset consists of a collection of these structured data points, where each data point represents a single instance of malware or benign software.
    - The dataset is partitioned into training, validation and test sets following the previously described methodology while preserving the spatial-temporal structure of the data.

3.4 Computer Requirement

It is recommended that a mandatory specification be included for the computer to observe better performance before the experiment can be conducted.

Processor	Intel(R) Core (TM) i5-6300U CPU @ 2.40GHz 2.50 GHz
RAM	16.0 GB
System type	64-bit operating system, x64-based processor
GPU	NVIDIA GeForce GTX 1080 GPU
Edition	Windows 10 Pro
Version	21H1
OS build	19043.1889
Experience	Windows Feature Experience Pack 120.2212.4180.0

3.5 Software Requirement

i. Spyder Python: Spyder is an open-source cross-platform integrated development environment for scientific programming in the Python language.
ii. Anaconda Plug-In: Spyder is included by default in the Anaconda Python distribution, which comes with everything you need to get started in an all-in-one package.
iii. Keras: Keras is an open-source library program that gives Python Programming for artificial neural networks. Keras runs as a key for the TensorFlow library to support ConvLSTM to produce better results [6].
iv. TensorFlow: TensorFlow is an open-source library program for artificial intelligence and machine learning. It can be utilized across multiple tasks simultaneously but owns a specific target on training and inference of deep neural networks.

3.6 Performance Evaluation

Under this section, we have done an evaluation of the BLSTM performance and the proposed ConvLSTM. To evaluate deeply, both protocols' performances, the simulation evaluated the capacity and ability of similar datasets, batch size, number of epochs and dropout. The results are discussed in terms of accuracy, loss and time taken.

3.7 Simulation Setup

We have executed the ConvLSTM neural network in Python using Keras as the framework. Moreover, we have split the dataset into a train set, validation set and test set conferring to the ratio (8:1:1). For the training BLSTM neural network, the dropout is set to 0.5, the batch size is fixed to 64 and the number of epochs is set to 30. Besides, the loss function will be adopted according to log loss as depicted in Table 1.

Table 1
Parameters

Parameters	Value
Ratio	(8:1:1)
Dropout	0.5
Epoch	30
Batch Size	64
Kernel Size	1
Filters	32
Dense	128,256
Return Sequence	True
Verbose	1

4. CONVLSTM: Recurrent Neural Network Approach

The ConvLSTM predicts the future state of specific cells in the input grids and the past state from local. If we observe the states as the hidden representation of moving objects, a ConvLSTM has a larger transitional kernel that has the ability to capture faster motions while one of the small kernels can capture slower motions [4].

This feature supports a lot of malware detection for fast capturing which has a higher contribution for time. ConvLSTM is using 2D Dimensional input and converts it into 3D Dimensional for reading input. A standard LSTM model has 2D Dimensional for it has a soft process, but BLSTM has dual LSTM cells which means has two layers of 2D Dimensional Input which gives a heavy shot of input causing input time taken to increase and produces results known as 3D Dimensional [4].

The Figure 5 and Figure 6 you provided illustrate concepts related to ConvLSTM (Convolutional Long Short-Term Memory) networks, which are designed to handle spatiotemporal data by combining convolutional operations with LSTM's recurrent processing capabilities. Let's explain these details more thoroughly.

Figure 5 shows how a standard 2D image is converted into a 3D tensor. In the context of ConvLSTM, this conversion process helps the model to manage temporal data along with spatial information:

i. <u>2D Image:</u> The image is represented by its pixel values organized in a 2D grid (height and width).
ii. <u>3D Tensor:</u> To incorporate temporal information, the image is transformed into a 3D tensor, where an additional dimension (usually representing time or sequence depth) is added.

This transformation allows the ConvLSTM to handle sequences of images, making it suitable for tasks like video prediction or time-series forecasting, where the spatial structure of data over multiple time steps needs to be analysed.

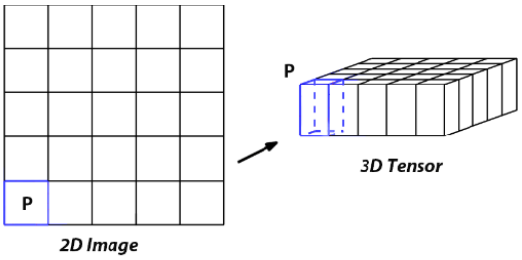

Fig. 5. Transforming 2D image into 3D tensor [4,5]

Figure 6 illustrates the structure of a ConvLSTM cell, showing how it processes data over multiple timesteps. Each ConvLSTM cell has multiple components:

i. <u>Hidden State (Ht) and Cell State (Ct):</u> The hidden state carries information from previous timesteps to the current timestep, while the cell state manages long-term dependencies.
ii. <u>Input (Xt):</u> Represents the data coming into the ConvLSTM cell at each timestep.
iii. <u>Convolutional Operations:</u> The ConvLSTM cell applies convolutional operations to the input data, which allows it to capture spatial patterns within each timestep.

This approach enables the ConvLSTM to preserve spatial information while learning temporal dependencies, making it more efficient for applications that involve both spatial and temporal data, such as weather forecasting or video analysis.

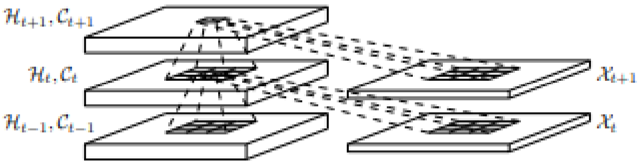
$\mathcal{H}_{t+1}, \mathcal{C}_{t+1}$
$\mathcal{H}_{t}, \mathcal{C}_{t}$
$\mathcal{H}_{t-1}, \mathcal{C}_{t-1}$
$\mathcal{X}_{t+1}$
$\mathcal{X}_{t}$

Fig. 6. Inner structure of ConvLSTM [4,5]

4.1 CONVLSTM Architecture

For such fileless malware detection, a motivating method is to utilize a model-based LSTM which is Long Short-Term Memory as an architecture of a Recurrent Neural Network [11]. In this type of architecture, the model gives the last hidden state to the forward step of the sequence. They are keeping the information from the past data from the network that has been observed previously before and used it to make a decision. To describe it better, the order of data is very important. Figure 7 shows an LSTM cell [14,15].

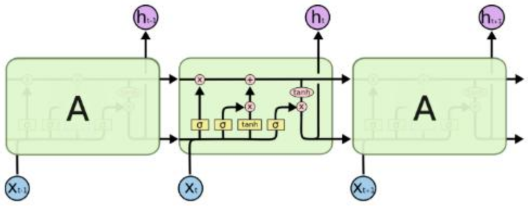

Fig. 7. A LSTM cell [14,15]

During working with malware detection, the best method would be a CNN (Convolutional Neural Network) architecture. The malware passed through Convolutional Layers, in which certain filters extract the significant features. Later, passing some convolutional layers in sequence, the output will be connected to an interconnected dense network. In our approach, fileless malware, one method is using ConvLSTM layers. It is a Recurrent Layer, similar to LSTM but internal matrix multiplications are swapped with convolution operations. In the output, the flows of data thru the ConvLSTM cells hold the input dimension (3D) in its place of becoming a 1D vector together features. Figure 8 shows the ConvLSTM cell [11].

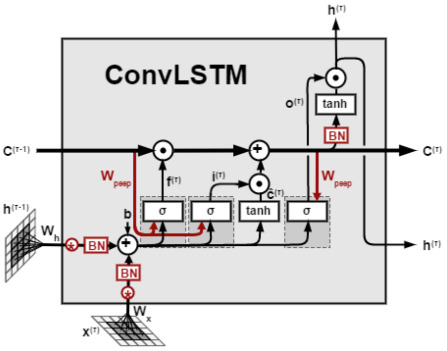

Fig. 8. A ConvLSTM cell [14,15]

Another method of ConvLSTM is the Convolutional-LSTM model, in which the Fileless malware passes through the convolution's layers and the output is set fixed to a 1D array with the gained features. During the repentance of the same process to all Malware in the time set, the output is a feature set across time and this is called LSTM layer input [11].

4.2 CONVLSTM Layer Input

The LSTM cell input is a data set across time which is a 3D tensor together shape containing samples, timesteps and features that a standard LSTM cell contains. The Convolution layer will input a set of malwares as a 4D tensor with the shape of samples, channels, rows and columns. This is another also 3D tensor but as we mentioned before, ConvLSTM breaks the sequence of 3D tensor (3D Dimensional) into sub-sequences where comes another 1D tensor in a total of 4D tensor. For input of a ConvLSTM is a set of malwares across as a 5D tensor which adds a 2D tensor with the shape of samples, timesteps, channels, rows and columns) [14,15].

4.3 CONVLSTM Layer Output

The LSTM cell results depend on the return sequence attribute. During set True, the sequence of output across a time where one output to each input. In such a situation, the output is a 3D tensor together with the shape of samples, timesteps and features. During the return sequence is False which would be a default, the result of output is the sequence's last value which will be a 2D tensor with the shape of samples and features [14,15].

The convolution layer output is Malware set as a 4D Tensor together with samples, filters, rows and columns). The ConvLSTM layer output is a set of Convolutions and an LSTM output. Similar to LSTM, if the return sequence is True, then the return sequence will be a 5D tensor with the shape of samples, timesteps, filters, rows and columns. Other ends, if the return sequence is false, then a 4D tensor with the shape of samples, filters, rows and columns [14,15].

4.4 Other Parameters

The other ConvLSTM characteristics speak from the Convolutions and the LSTM layer. In the Convolution layer, the significant ones are:

i. Filters: The output filter numbers in convolution.
ii. Kernel Size: Defining the width and height of the convolution window.
iii. Padding: Define whether the input data is valid or same
iv. Data Format: The format of malware if the channel comes first or last.
v. Activation: A default is a linear function.

From the LSTM layer, the most significant one is:

i. Recurrent activation: Activation function to utilize the step of recurrent. It is a default hard sigmoid.
ii. Return Sequences: Whether to return the end output in the sequence of output which is false or else full sequence. The default is False.

4.5 CONVLSTM Design

The ConvLSTM model owns one input, the beginning features in sequence, 6 individual outputs and one for its own category. The model input which we used is the malware dataset. It will be broken into samples, features, timesteps, rows and columns. Limit the features by 300 per sample and the max length is 1 and balance rows and columns are set to 1. Samples are the number of available malware data for training [12,13].

In this example, we utilize the return sequence equal to False which is the default so the output should be in samples and features but due to the model having 5 individual outputs, the expected output will be in samples, features, timesteps, rows and columns. The impact of using return sequences is the model will categorize each sample in one set [12,13].

The design starts with 2 ConvLSTM layers where one will be Batch Normalization and MaxPooling. In sequence, it breaks into branches and one for every category. Every branch is similar, they begin with one of the ConvLSTM shadows by MaxPooling. Then, this output is attached to an interconnected Dense Network. Finally, the end layer is Dense with one cell. The following sample speaks the basic model [12,13].

4.6 Test, Validation, Training Set

When we start the ConvLSTM program, we begin with a standard test set for all the processes. This test set is not used to run the samples with the test set only. Instead, we use it to train the dataset and continuously improve the program, which is known as the train set. The classification output of every test set is validated using the training set output, which is known as the validation set, as shown in Figure 9. This process is iterative, with each iteration representing a step forward in refining the program.

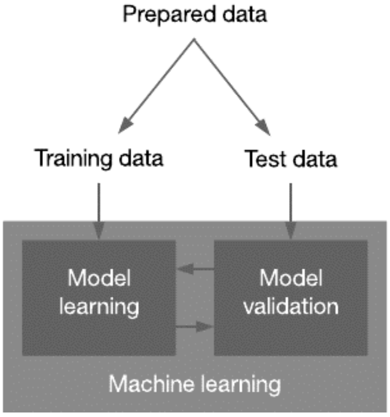

Fig. 9. Test, validation, training set

4.7 CONVLSTM Algorithm

The main weakness of Fully Connected LSTM in controlling spatiotemporal data is its full connection’s usage input to state and state-to-state transitions in which no information of spatial is encoded. To solve this issue, a better design feature is all inputs put on X1, …., Xt, cell outputs C1, …. Ct the hidden state H1, …. Ht and the gates it, ft, 0t of ConvLSTM are 3D tensors where last two dimensions are spatial dimensions which are rows and columns [18-20].

To overview a better observation of the input and states, we may take them as vectors standing on the partial grid. The ConvLSTM predicts the future state of some cells in the grid thru inputs and past states from the local neighbours. This is possible to be accomplished by utilizing the convolution operator and '◦' as before denotes Hadamard Product in Figure 10 [18-20].

$$
\begin{gathered}
i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \odot \mathcal{C}_{t-1} + b_i) \\
f_t = \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \odot \mathcal{C}_{t-1} + b_f) \\
\mathcal{C}_t = f_t \odot \mathcal{C}_{t-1} + i_t \odot \tanh(W_{xc} * X_t + W_{hc} * \mathcal{H}_{t-1} + b_c) \\
o_t = \sigma(W_{xo} * X_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \odot \mathcal{C}_t + b_o) \\
\mathcal{H}_t = o_t \odot \tanh(C_t)
\end{gathered}
$$

Fig. 10. Conv-LSTM Algorithm [18-20]

If we observe the states as hidden representations of objects which move, a ConvLSTM with a larger kernel is able to capture faster motions besides one with a smaller kernel captures slower motions. Also, if we take the same view, the inputs, the cell outputs and hidden states of the basic FC-LSTM represented above algorithm may also be seen as 3D tensors together with the last one dimensions be one. In that think, FC-LSTM is a rare case of ConvLSTM with all standing features on one cell [18-20].

To make sure the states own a similar number of rows and columns as the inputs, padding is required before introducing the convolution operation. There you go, padding the hidden states on the border points can be observed as utilizing the states on the world outside for the math. Previous the first input reaches, we begin all the states of LSTM to 0 which correlates to total ignorance for the future [20].

Same, if we run zero-padding on hidden states, we can set the state to the world outside to 0 and think that no knowledge of the outside. Through padding on states, we can teach border points in another way, which would be a handful in other cases. For example, think that the system we are looking at is a moving ball covered by walls. Even though we are unable to see those walls, we can conclude their presence by searching for the ball bouncing over them again and again, which can be difficult to run if the boundary point has similar state transition dynamics as the inner points [20,21].

From Figure 11, the proposed model utilizes the convolution operation layer to pull higher-level collaborating features from more than one colliding object in multiple interactive layers in variant time segments, seeing that the pooling will split like a time sequential organization since its suspended chosen features. Later, the LSTM model is taken to extract the feature sequence which has a long-term dependency on the historical period features. Besides, looking at the influence of prediction syntax on malware detection, we have proposed this algorithm [7,8].

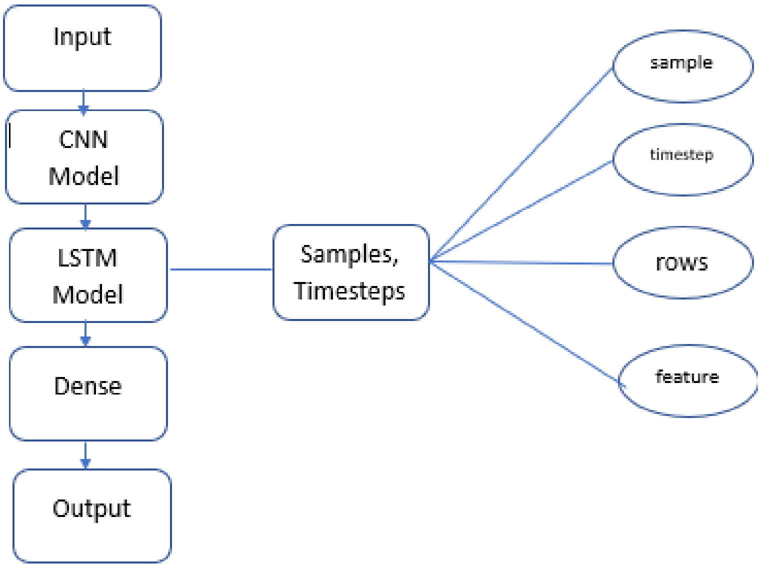

Fig. 11. Flowchart of ConvLSTM [8]

From the traditional deep learning models, CNN and BSLTM are the most broadly applied ones. CNN can obtain bigger features from data and improve the anti-degeneration of the extracted features together with pooling and convolution operations, but they are unable mode the time variation series. LSTM in another way, can sort out this problem that other neural networks are incapable to manage in terms of time series [7,8].

That neuron's outputs can be straightly impacted at the next time step in the LSTM. Moreover, LSTM can manage input sizes smaller than CNN. In fileless malware, the maximum input size is 300 data only. The CNN input size can be up to more than 1000 data which shows that the feature extraction capacity of LSTM is lower than CNN. In this study, we have merged the different pros of CNN and LSTM to propose the ConvLSTM model for fileless malware detection [7,8].

5. Results and Discussion

In this experiment, the research conducted program simulation test using Python 3.9.7 Spyder Program by running API call algorithm to extract the malware from malware dataset in form of zip file. There is an addition of ConvLSTM1D for this testing which sufficient to run for this program to break the sequence into four sub sequence as feature extraction which max length, input shape, max word as samples, timesteps, rows, features. Those samples have run with the parameter set.

The programs set the max length of 300 malware per process of epoch where been set 30 and every end of epoch test, it produces the accuracy, loss and time taken. When every epoch completes, it starts with test set and continue with validation set end. Training set will be learning the mistakes from test set and train the next epoch to produce better results in terms of increasing the accuracy and reduce the number of malware loss.

End of the 30 epoch, the test and train set result pushed into history module to plot the graph for accuracy and loss to show the results of graph line. The results show from the low value and increase to high value. These results are variant due to train set and validation test conduct on this program which improves the result within set number of epochs to be processed. Results will be in decimal point as 0.9 = 90%. The results of performance evaluation will be discussing in the next subsection.

5.1 Impact to Accuracy

The comparative analysis graph in Figure 12 illustrates the accuracy performance of four models which are BLSTM, LSTM, CNN-BLSTM and ConvLSTM—in malware detection over 30 epochs. The LSTM model starts at around 75% accuracy and improves steadily to about 90%. However, it struggles to surpass the 0.95 benchmark, indicating limitations in its simpler architecture. This suggests that while LSTM can effectively learn from the data, it lacks the complexity to achieve the highest accuracy. The CNN-BLSTM model, which combines convolutional layers with BLSTM, starts at 75% accuracy and improves to nearly 90% by the 15th epoch.

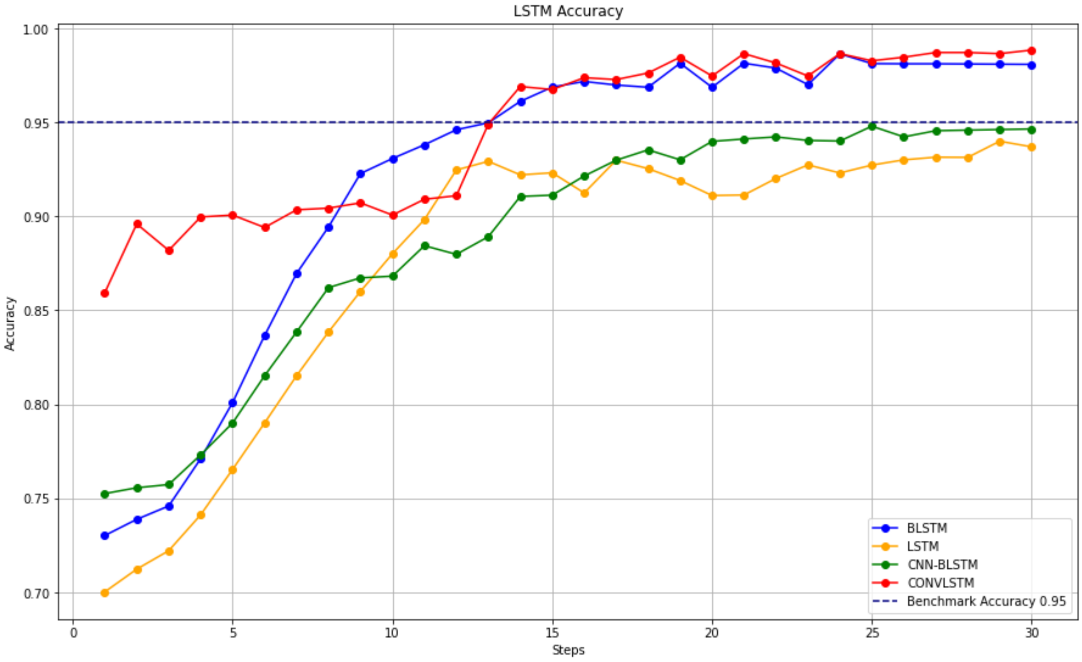

Fig. 12. Accuracy result

However, it shows some instability and does not match the performance of ConvLSTM and BLSTM. This instability might be due to the convolutional layers not fully compensating for the sequential dependencies that BLSTM handles. The BLSTM model begins at 70% accuracy and experiences more fluctuations in the early epochs. Nevertheless, it catches up to around 0.95 accuracy by the 13th epoch, benefiting from its bidirectional sequence processing.

This dual processing allows BLSTM to learn more comprehensive features from the data, leading to improved accuracy over time. ConvLSTM demonstrates the strongest initial performance with a 90% starting accuracy. It maintains a stable learning curve and consistently achieves high accuracy, attributed to its convolutional layers' ability to capture spatial features effectively. ConvLSTM use of SpatialDropout2D promotes feature independence and stability, further enhancing its performance.

5.2 Impact to Loss

Figure 13 shows the initial performance at epoch 1, BLSTM starts with a high loss of approximately 70%, indicating poor initial performance in recognizing malware. In contrast, ConvLSTM shows a significantly lower initial loss of about 10%, suggesting better initial detection capabilities. LSTM and CNN-BLSTM display intermediate initial losses between those of BLSTM and ConvLSTM.

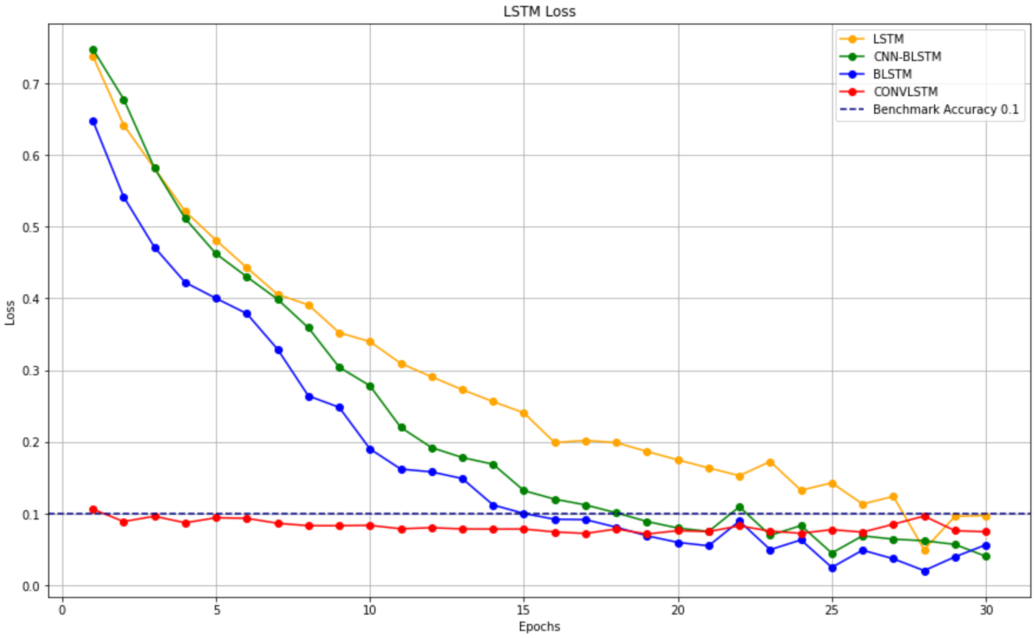

Fig. 13. Loss result

Over the epochs, ConvLSTM maintains a stable and consistently low loss. This stability can be attributed to the model's ability to leverage filters and larger kernel sizes effectively, enabling it to capture both fast and slow-motion patterns in the data. ConvLSTM's use of parameter sharing further enhances its performance. By sharing parameters across different parts of the model, ConvLSTM can efficiently capture spatial and temporal dependencies in the data, leading to a more robust detection mechanism.

While BLSTM starts with a high loss, it gradually reduces its loss, becoming competitive with ConvLSTM by the end of 30 epochs. However, BLSTM's performance fluctuates significantly until around the 13th epoch, indicating instability during the training phase. LSTM and CNN-BLSTM exhibit a gradual decline in loss but do not reach the low loss levels of ConvLSTM or BLSTM by the end of the epochs. Key factors contributing to ConvLSTM's superior performance include its ability to use multiple filters and larger kernel sizes, which help minimize loss and improve pattern recognition in malware detection.

ConvLSTM's loss curve remains stable across epochs, highlighting its reliability in malware detection. In contrast, BLSTM experiences fluctuations, which could hinder its ability to consistently capture true malware. BLSTM requires the entire sequence to be available before processing, leading to high initial losses. Over time, as more data is processed, BLSTM improves, but this requires around 30 epochs to achieve a comparable performance to ConvLSTM.

In conclusion, ConvLSTM demonstrates superior and stable performance in reducing loss during malware detection across all epochs compared to other LSTM models. Although BLSTM shows improvement over time and achieves low loss towards the end, its initial instability and fluctuations indicate that it may not be as reliable for early-stage detection as ConvLSTM. The ability of ConvLSTM to maintain low and stable loss levels, along with its efficient parameter sharing, makes it a more robust choice for fileless malware analysis.

5.3 Impact to Time

Figure 14, BLSTM starts at a higher time but quickly drops, stabilizing between the 3rd and 16th epochs. From the 16th epoch onwards, the time increases significantly, peaking at 174 seconds by the 30th epoch. This indicates that BLSTM becomes less time-efficient over longer training periods. ConvLSTM starts at a lower time compared to BLSTM and maintains a more consistent reduction in time. ConvLSTM shows a stable performance with time averaging around 110 seconds, indicating better efficiency and less variability.

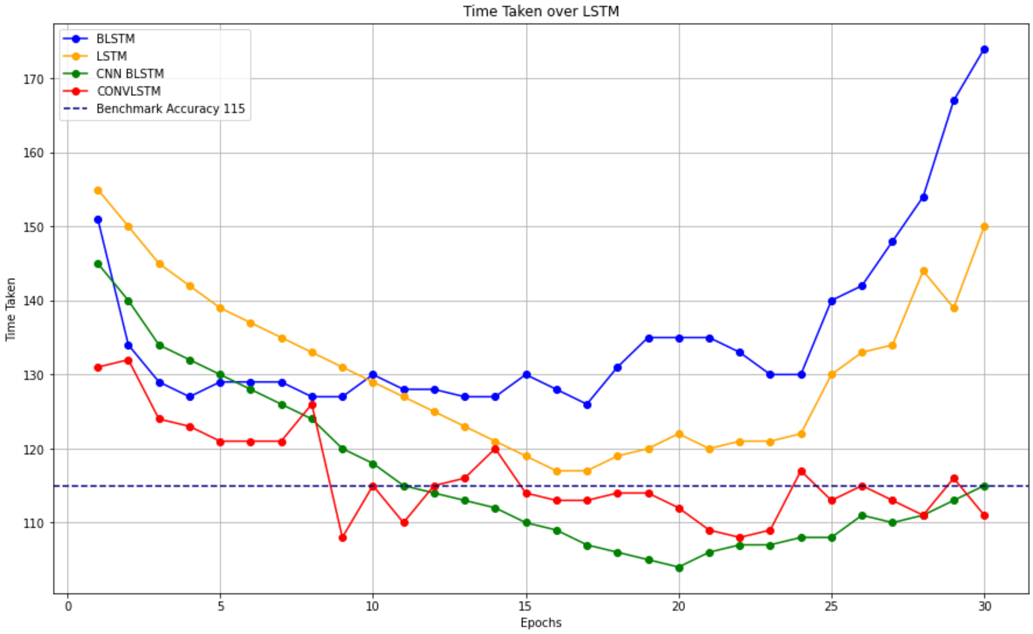

Fig. 14. Time taken result

The differences in time efficiency between the models can be attributed to their underlying mechanisms. BLSTM utilizes two LSTM cells requiring complete sequences from both directions to produce an output. This dual processing leads to longer times, especially as the number of epochs increases, making it less suitable for real-time analysis. ConvLSTM uses a single LSTM cell, running processes within itself, which enhances time efficiency. The parameter-sharing mechanism of ConvLSTM contributes to its stable and faster performance.

The time efficiency of these models is significantly influenced by the computational resources available. The CPU and memory capacity of the machine can impact the time taken for each epoch. This suggests that with better hardware, both models could potentially perform faster, but the relative efficiency between them would remain. In terms of implications for malware detection, ConvLSTM's stable and lower time consumption makes it more suitable for real-time scenarios where prompt responses are critical. BLSTM's higher time requirements indicate that it might be better suited for scenarios where accuracy is more critical than speed and sufficient computational resources are available to manage the longer processing times.

6. Conclusion

In summary, this research highlights the limitations of traditional malware detection methods, such as Bi-Directional Long Short-Term Memory (BLSTM) models, in effectively identifying fileless malware due to high computational costs and processing delays. Fileless malware operates silently in memory and evades traditional detection techniques, making rapid detection and response challenging.

To address these issues, the study proposes using Convolutional Long Short-Term Memory (ConvLSTM) models, which show significant improvements in detection accuracy, processing speed and overall performance. ConvLSTM achieved a detection accuracy of 98% compared to BLSTM’s 90% and reduced processing time from 22 to 10 seconds. Additionally, ConvLSTM showed lower loss rates, indicating better reliability and stability.

ConvLSTM's success is attributed to its ability to capture spatial-temporal features more effectively by integrating convolutional layers with LSTM architecture. This integration reduces computational complexity and enhances the model’s ability to manage sophisticated fileless malware.

The research underscores ConvLSTM’s potential as a superior tool for real-time malware detection and mitigation. Future work could focus on optimizing ConvLSTM's configurations and evaluating it across various cybersecurity scenarios to further enhance its performance and scalability. This study contributes to advancing malware detection technologies, offering a promising approach for securing digital infrastructures against evolving cyber threats.

7. Future Works

This ConvLSTM model has been introduced in the fileless malware detection method and this method in their field have significantly increased the security level of the Endpoint Security System. However, there remain certain problems that might need to be deep dive in the future. Below are the keys that need to be investigated:

i. Input size limitation: Using ConvLSTM1D due to the input size limitation is understandable, but it would be beneficial to find a way to adapt ConvLSTM2D for your dataset. Perhaps preprocessing the data to add an additional feature could resolve this issue.

ii. Dual ConvLSTM cells: Implementing a dual ConvLSTM cell could indeed enhance the model's ability to predict both past and future sequences. This would require modifications to the architecture and training process to accommodate the increased complexity.

iii. Fluctuation in accuracy and high loss: To address fluctuations in accuracy and increasing losses during training, you might need to fine-tune hyperparameters, adjust learning rates or explore different optimization techniques. Additionally, analysing the causes of fluctuations, such as data imbalance or model overfitting, could help mitigate these issues.

iv. Improved visualization with Metaplot: Enhancing Metaplot or using other visualization libraries to extract and display ConvLSTM results more effectively could streamline the analysis process. This would involve integrating the necessary functionalities to plot relevant metrics and visualize model performance.

v. Enhanced reporting of malware analysis results: Beyond accuracy, loss and training time, incorporating additional metrics such as the number of positive and negative malware samples, along with their respective types, can provide more comprehensive insights into the model's performance and the characteristics of the analyzed malware datasets.

Acknowledgment

This work is supported by PUTRA Grant Scheme (IPM), Universiti Putra Malaysia under Project No. GP-IPM/2019 Vote: 9676100.

References

[1] Brownlee, Jason. "Cnn long short-term memory networks." *Machine Learning Mastery* (2017).

[2] Brownlee, Jason. *How to Develop a Bidirectional LSTM for Sequence Classification in Python with Keras. 2021*. 2019.

[3] Brownlee, Jason. "How to develop LSTM models for time series forecasting." *Machine learning mastery* 14 (2018).

[4] Chauhan, Nagesh Singh. "Introduction to RNN and LSTM." *The AI Dream*, (2022).

[5] Dey, Polash, Emam Hossain, Md Ishtiaque Hossain, Mohammed Armanuzzaman Chowdhury, Md Shariful Alam, Mohammad Shahadat Hossain and Karl Andersson. "Comparative analysis of recurrent neural networks in stock price prediction for different frequency domains." *Algorithms* 14, no. 8 (2021): 251. https://doi.org/10.3390/a14080251

[6] Fotheringham, Greig, XuekeXueke1 and M. Nuramzan IftariM. "Keras AttributeError: 'Sequential' Object Has No Attribute 'Predict_classes.'" *Stack Overflow*, (1968).

[7] Hochreiter, Sepp. "LSTM RNN in TensorFlow - Javatpoint." *Javatpoint*. (2023).

[8] Keydana, Sigrid. "Convolutional LSTM for Spatial Forecasting." *Posit AI Blog*, (2020).

[9] Miguel, Tiago. "How the LSTM Improves the RNN." *Medium*, (2021).

[10] Niu, Kun. "Protocol Exchange, a Platform for Researchers to Find, Share and Discuss Life Science Protocols." *BLSTM Architecture*, (2023).

[11] Oinkina. "Understanding LSTM Networks." *Colah's Blog*, (2023).

[12] Sinha, Nimesh. "Understanding lstm and its quick implementation in keras for sentiment analysis." *URl: https://towardsdatascience. com/understanding-lstm-andits-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47 (visited on 04/24/2018)* (2018).

[13] Soni, Manik. "Understanding architecture of LSTM cell from scratch with code." *Medium* (2018).

[14] Srivastava, P. "*Essentials of Deep Learning: Introduction to Long Short-Term Memory, Analytics Vidhya*. 2023.

[15] Srivastava, P. "What Is Endpoint Detection and Response." *Webroot*, (2023).

[16] Cybereason Team. "Fileless Malware 101: Understanding Non-Malware Attacks." *Cybersecurity Software*, (2023).

[17] Liu, Yingying and Yiwei Wang. "A robust malware detection system using deep learning on API calls." In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 1456-1460. IEEE, 2019. https://doi.org/10.1109/ITNEC.2019.8728992

[18] IGI Global. "What Is ConvLSTM." *IGI Global*, (2023).

[19] Quora. "What Is the Difference between ConvLSTM and CNN LSTM?" *Quora*, (2023). https://www.quora.com/What-is-the-difference-between-ConvLSTM-and-CNN-LSTM

[20] Xavier, Alexandre. "An introduction to ConvLSTM." *Published in Neuronio* (2019).

[21] McDonough, Greg. "How to Defend Against Fileless Malware in 2024." *RSA Conference*, (2024).

[22] Any Run. "Malware That Resides in RAM: Explaining Fileless Malware." *ANY.RUN*.

[23] Trend Micro. "Risks Under the Radar: Understanding Fileless Threats." *Trend Micro*, (2019).

[24] Any Run. "Malware That Resides in RAM: Explaining Fileless Malware." ANY.RUN.

[25] Fu, Josh. "Memory-Based Attacks Are on the Rise: How to Stop Them." *BlackBerry Cylance*, (2023).

[26] Cybersecurity Journal. "Deep Learning in Cybersecurity: Combating Fileless Malware." *Cybersecurity Journal*, (2023).

[27] Qiang, Weizhong, Lin Yang and Hai Jin. "Efficient and robust malware detection based on control flow traces using deep neural networks." *Computers & Security* 122 (2022): 102871. https://doi.org/10.1016/j.cose.2022.102871

[28] Ocatak, O. "LSTM Malware Detection Dataset."

[29] Goodfellow, Ian, Yoshua Bengio, Aaron Courville and Yoshua Bengio. *Deep learning*. Vol. 1, no. 2. Cambridge: MIT press, 2016.

[30] Hassan, M. F., R. Akbar, K. Savita, R. Ullah and S. Mandala. "Ransomware Classification with Deep Neural Network and Bi-LSTM." *Journal of Advanced Research in Applied Sciences and Engineering Technology* 47, no. 2 (2024): 266-280. https://doi.org/10.37934/araset.47.2.266280