

Improving Android Malware Detection with Convolutional Neural Networks and Long Short-Term Memory

Dr. Rafid Sagban^{1*}, Dr. Rana Hikmet Tobia Saloom², and Dr. Naseer Ali Hussien³

^{1*}Scientific Research Center, Al-Ayen Iraqi University, Thi-Qar, Iraq; College of Information Technology, University of Babylon, Hilla, Iraq. rsagban@alayen.edu.iq, <https://orcid.org/0000-0002-3518-1396>

²Ministry of Higher Education and Scientific Research, Iraq. ranasaloom3081978@gmail.com, <https://orcid.org/0000-0002-7126-5343>

³Scientific Research Center, Al-Ayen Iraqi University, Thi-Qar, Iraq. naseerali@alayen.edu.iq, <https://orcid.org/0000-0001-9499-6694>

Received: September 27, 2025; Revised: November 20, 2025; Accepted: December 23, 2025; Published: March 31, 2026

Abstract

Cybercriminals have become increasingly interested in the spread of critical information, particularly in interpersonal contact and mass distribution of programs and file downloads. This has heightened researchers' awareness of the rampant spread of malware and data breaches. It is anticipated that the number and intensity of malicious software will continue to rise, underscoring the imperative need for strong security architectures. This is especially critical for mobile networks and ubiquitous computing security, given the growing threat posed by hackers. The proposed project involves creating a new dataset using a strictly controlled, shared sample pool to address security threats in wireless mobile networks, leveraging dynamic analysis techniques for malware detection. The dataset aims to enhance the recognition of malicious software by leveraging methods such as encryption and obfuscation. The suggested classification algorithm is CNN-LSTM, a combination of Convolutional Neural Networks (CNNs) and the Long Short-Term Memory (LSTM) model, which excels at learning complex, sequential features. The CNN and LSTM models were tested on a dataset comprising more than 10,000 malware samples and achieved accuracies of 98% and 97%, respectively. These findings demonstrate how deep learning models can be used to enhance the security of mobile networks and provide effective protection against emerging threats in mobile and ubiquitous computing systems, in a highly beneficial way.

Keywords: Malicious Software, Hackers, Dynamic Analysis, Encryption, Convolutional Neural Network (CNN), Security Protocols, Mobile Networks, Ubiquitous Computing.

1 Introduction

Mobile applications have become integral to daily routines because they can perform many operations directly on portable devices. The advancement of technological paradigms, namely the integration of these applications into mobile computing platforms, has led to a significant revolution in communication. In contrast to traditional personal computers, contemporary smart gadgets are equipped with a diverse range of advanced sensors (Muzaffar et al., 2022). The sensors include cameras,

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), volume: 17, number: 1 (March-2026), pp. 229-250. DOI: 10.58346/JOWUA.2026.II.014

*Corresponding author: Scientific Research Center, Al-Ayen Iraqi University, Thi-Qar, Iraq; College of Information Technology, University of Babylon, Hilla, Iraq.

microphones, gyroscopes, and global positioning systems (GPS). According to (Aravind et al., 2023), using diverse sensors offers end-users opportunities to develop innovative applications and services. Conversely, this generates a substantial volume of data, often containing highly sensitive information. Because of such conditions, there is an urgent need for effective security strategies, which are very important for in protect in users against malicious programs the exploit of the high-level capabilities of intelligent devices and the invaluable information stored them.

The use of mobile devices has increased, and cyberattacks have become more advanced, making mobile network security a major issue. Android-based devices, which account for a significant share of the global smartphone market, are especially susceptible to malware that can compromise users' privacy and data. The security concerns of mobile networks and ubiquitous computing environments are increasingly a major challenge, as these devices continue to play a significant role in day-to-day communication, business, and data exchange. As the availability of the Internet of Things (IoT) grows and more devices are connected to it, these problems are becoming more distributed.

The following security concerns are discussed in this research paper: the detection of malware in mobile networks using deep learning frameworks, namely Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTMs). These models are meant to complement existing security designs and provide a powerful solution to the ever-changing landscape of cyber threats in mobile and ubiquitous computer systems. This work enhances security mechanisms in mobile and pervasive computing networks by focusing on detecting malicious software and applying capabilities such as encryption and dynamic analysis.

Malware, a collective term for malicious software, comprises any program that can cause damage or destruction to computer systems and that can enter a system without user permission (Alzaylaee et al., 2020). Researchers use three principal analyses to detect malware attacks: static, dynamic, and hybrid analysis (Ranganathan & Bhattacharya, 2021).

At times, there has been an extraordinary increase in people's dependence on mobile devices, particularly smartphones. Based on the research findings, the worldwide use of mobile devices has reached 3.92 billion in just 2 years (Alzaylaee et al., 2020). It is important to note that a substantial percentage of these devices, 70.68% (Ranganathan & Bhattacharya, 2021; Gohari et al., 2021), make use of the Android operating system. This statistical information holds considerable significance. Those who employ Android smartphones often underutilize the antivirus software pre-installed on the devices. Furthermore, even those who can successfully install the application may encounter challenges in effectively using it to detect illnesses (Bayazit et al., 2020; Giji Kiruba et al., 2023) accurately. Due to the widespread adoption of Android among a significant portion of the population and the potential to access substantial amounts of sensitive data through these devices, hackers are increasingly focusing on the Android platform (Bhat et al., 2023).

Signature-based detection is the predominant approach in developing anti-malware software for diverse operating systems, such as Android, Windows, and others. The distinctiveness of signatures may be attributed to the association with specific instances of malicious software. The signatures encompass cryptographic hashes of the files and byte patterns that may be utilized to identify malware inside a particular environment. According to the source (Sahin & Bahtiyar, 2020), Consider, in the context of scrutinizing an application, the hypothetical scenario in which the program's signature is obtained and subsequently cross-referenced with an established database to ascertain the presence or absence of harmful code within the software. Conversely, even minor modifications to the signature can impede anti-malware software's ability to detect hazardous applications accurately. To implement these updates, it may be necessary to modify specific lines of code, a series of instructions, or even specific keywords.

The limited capacity of anti-malware systems hinders their ability to detect newly emerging malware. New kinds of malware can arise from zero-day attacks or from modifications to previously identified malware.

Placing significant focus on the correlation between the increasing prevalence of Android devices and the resulting vulnerability to unauthorized access is paramount. These illicit organizations may employ software applications that have been successfully deployed on Google Play to gain entry and exploit the privileges they acquire to advance their nefarious objectives. As a result, individuals who lack vigilance are prone to inadvertently installing malicious software applications, thereby unknowingly granting the attacker access (Merlo et al., 2021; Santos et al., 2013). According to research released in the 3rd quarter of 2020 (12), the total number of mobile apps available for download on Android-based operating systems has surpassed 2.86 million. Moreover, a considerable number of viral software programs, totaling 482,579 cases and an approximate 16,000 applications per day detection rate, were found to be in operation every month (Yu et al., 2018). With many rogue software programs having been developed, it is now necessary to utilize more advanced techniques in order to identify and categorize various types of cyber threats in an efficient manner.

There are three primary methodologies for detecting malicious software on Android-based devices: static, dynamic, and hybrid analysis (Ciaramella et al., 2022). The detection of malicious programs on Android devices can be accomplished through three basic methods, with each of them comprising multiple methods. The term static analysis is associated with gaining an understanding of the working of an application without launching the software on a hard disk or an Android emulator. Because of the appearance of the given phenomenon, analysts have an opportunity to determine and investigate the actions that do not correspond to the norm and can portend suspicious behavior.

Although there is a potential risk of losing some functionality, code obfuscation, and dynamic code loading, this approach can encompass all aspects comprehensively.

The currently accessible functions and features. On the other hand, dynamic analysis involves the extraction of attributes via the execution of a process. It offers an advantage over static analysis as it reveals novel attributes and risks associated with dynamic analysis in terms of time and computational resources required (Yu et al., 2018).

In contrast, hybrid analysis combines elements of both static and dynamic analysis, leading to a detection method characterized by enhanced precision and efficiency (Alotaibi & Fawad, 2022; Elayan & Mustafa, 2021). In order to establish the hybrid analysis approach, static and dynamic analysis approaches must be blended into the process. In methodology, detecting dangerous software is conducted through static analysis-based methods.

Based on the findings of a recent academic investigation conducted by a reputable computer security organization (Liu et al., 2020), it has been determined that internet repositories exhibit the rapid uploading of a novel form of malicious software within around eight seconds. There is a fast rate of roughly 11,000 newly generated instances of malware daily, resulting in a cumulative total of around 11,000 cases. Merely maintaining frequent anti-malware software updates is insufficient for effectively addressing the continuous emergence of newly created malicious software. Consequently, the anti-malware software exhibits limitations in detecting several malware strains concurrently, potentially reaching hundreds or even thousands. Deep learning (DL) can contribute to resolving this issue by enabling anti-malware software to identify novel malware strains by recognizing shared patterns observed in existing variants.

In applying machine learning algorithms to data, it is customary to initially use data preprocessing as a standard procedure. The file must undergo a specialized application either on a physical Android device or an Android emulator.

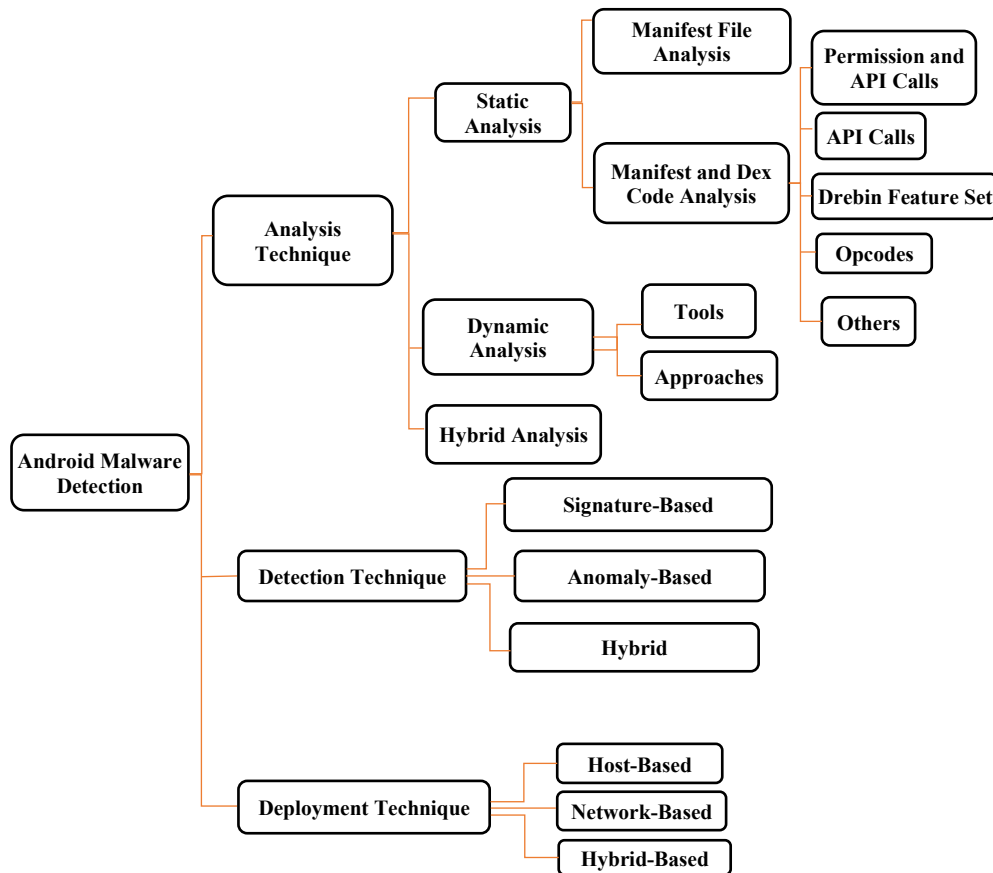


Figure 1: Strategies of android malware diagnosis

The operation can be performed using either manual labor or automated methods. This methodology offers a systematic approach that may be employed to ascertain its distinctive characteristics.

The data often needs an initial editing procedure. In order to extract the differentiating qualities of the data file (Chan et al., 2019), it is necessary to subject it to a particular method. These properties have been classified into two distinct groups, namely static and dynamic. "Static analysis" refers to thoroughly examining a file without executing the corresponding program (Kim et al., 2022). Conversely, dynamic analysis occurs during program execution and is frequently conducted in a controlled environment like a sandbox. Pre-release analysis of this nature is performed before the application's public release.

In contrast to static analysis, dynamic analysis often necessitates a more substantial allocation of processing resources for its completion. Academic research and commercially available anti-malware software usually prioritize static or dynamic characteristics or employ a combination of both approaches. The use of machine learning to identify dangerous software for Android has been the topic of earlier investigations (Choudhary & Kishore, 2018). Nevertheless, the predominant emphasis of these studies has been directed towards specific methodologies, so offering a restricted perspective on the existing corpus of literature rather than a comprehensive inquiry.

Key Contribution

- This paper presents a new dataset that is meant to boost the detection of malware in mobile networks through dynamic analysis methods in dealing with security threats in the wireless mobile environment.
- It suggests using a hybrid CNN-LSTM network to refine malware detection with the help of space feature extraction of Convolutional Neural Networks and sequential features dependence of Long Short-Term Memory.
- The CNN-LSTM model has been shown to be very efficient, as it records an accuracy of 98% using the CNN model and 97% using the LSTM model, thus illustrating that deep learning models are indeed effective in mobile network protection and countering cyber-attacks.

The paper is structured in the following way: The Introduction section gives the background information about the mobile network security and the necessity to detect malware in the most advanced ways. Related Works section is the review of the existing malware detection techniques, such as the static, dynamic and hybrid analysis, and identification of gaps that the proposed solution would have addressed. The Methodology section outlines the hybrid deep learning architecture that is a combination of Convolutional Neural Networks (CNN) network with Long Short-Term Memory (LSTM) network in detecting malware and the dataset and preprocessing stages adopted. The paper provides experimental settings and measure of performance of the CNN and LSTM models in the Experiments and Results section. The Discussion explains the results, which should result in the effectiveness of the model, and further work points should be proposed. Lastly, the Conclusion is a summary of the study contribution and suggestions to the future research in mobile network security with deep learning.

2 Related Works

This section comprehensively examines the primary research methodologies employed in previous studies on deep learning. To enhance comprehension of the subject matter, the forthcoming discussion will concentrate on three distinct forms of analysis: static, dynamic, and hybrid. The assessment of each category relies on the used models, the implemented accuracy metrics, the utilized benefits, and the selected datasets. The redundancy issue was identified as one of the problem areas requiring attention. The present text effectively tackles this concern by using measures to minimize redundancy. The final result involves a scientifically valid explanation of the topic (Figure 1).

Static Analysis

When it comes to examining software, also known as malware, there is a technique called analysis that doesn't require running the software itself. In this way, it is possible to consider the virus in relation to its execution. It entails decoding the files related to the software in a readable form so as to enable the specialists to understand the intended purposes of the virus better. Under analysis, the assembly's codes and file headers are analyzed, which are elements. A standard method of malware examination is malware scanning without malware execution, as it is time-saving and straightforward to apply.

Researchers have extensively studied the use of learning (DL) and machine learning (ML) techniques for identifying Android malware and enhancing Android device security. These studies are referenced in papers (Ciaramella et al., 2022; Mercaldo et al., 2022; Catal et al., 2022). Additionally, reference (Odusami et al., 2018) presents an examination of implementing static analysis techniques in evaluating

Android applications from 2011 to 2015. The duration of this research spanned five years. Moreover, the paper presented an extensive analysis of the multiplicity of the tools that can be used to conduct a static analysis of Android source code. The review yielded a compilation of fundamental analysis approaches, including abstract representation, taint analysis, symbolic execution, program slicing, code instrumentation, and type/model checking. The evaluation fails to consider the applicability of static analysis techniques in detecting malware, although acknowledging the widespread use in discovering privacy and security vulnerabilities. Furthermore, the study was unable to include current academic research introducing innovative methodologies for studying and identifying malware.

An additional intriguing method, which Zhu and Zhang (Ashawa & Morris, 2021) published in 2018, is among the most effective for classifying Android malware. Utilizing the permissions specified in the manifest, a call graph is generated to facilitate comprehension of the flow from Smali code. The architecture is built based on malware detection with the help of a Graph Convolutional Network (GCN). The approach is suitable for the DREBIN dataset, one of the most popular datasets applied in malware detection. The research on source code vulnerabilities is necessary to solve the security problems within the Android platform. In addition to identifying malware specifically engineered to target Android devices, this task must also be undertaken. In the study conducted (Li et al., 2017), a comprehensive analysis was undertaken, which involved scrutinizing and evaluating several scholarly articles. The main topic of this research was the study of the use of deep learning (DL), machine learning (ML)-based, and data mining methods to detect software vulnerabilities. It is necessary to agree that the research concentrated on the search of the literature in general about software security and did not particularly focus on vulnerability analysis within Android code. This distinction is essential to make, as it was the main subject of the research. The research, therefore, failed to explore the possibility of conducting a risk assessment of the code of the Android operating system. However, the information obtained under deep learning models in vulnerability studies can be helpful to carry out extensive research in a given programming language.

3 Dynamic Analysis

More and more Android malware avoids static detection through techniques such as repackaging and code obfuscation; dynamic analysis methods based on behavioral characteristics can solve this problem well (Zhu et al., 2018). Dynamic analysis involves observing and evaluating the operational characteristics of Android application software during its execution. The primary focus of dynamic analysis techniques often includes monitoring sensitive data access and API calls, among other factors.

In particular research (Ghaffarian & Shahriari, 2017), the focus of inquiry revolved around linters, specifically examining instances such as Klint and Android Lint. Android Studio incorporates a software tool called Android Lint, which can identify 339 potential issues of validity, security, performance, usefulness, accessibility, and internationalization. After identifying security vulnerabilities using Android Lint, the Fix Droid framework, proposed in reference (Cai et al., 2018), offers developers security-oriented advice and corresponding fixes. Utilizing (DL) techniques can boost the effectiveness of the Fix Droid methodology by generating security suggestions with a heightened level of accuracy.

In the published study, the authors endeavored to include statistical functions into integrated development environments (IDEs) and text editors such as Jupiter, Sublime Text, and PyCharm. To do this, it introduced a mechanism called Magpie Bridge (Habchi et al., 2018). However, it is imperative to engage in further discourse to assess the feasibility of extending this approach to the Android platform.

The authors of Nguyen et al., (2017) conducted a research study on the vulnerabilities exposed by Certificate validation in Android applications, utilizing Transport Layer Security (TLS) and Secure Sockets Layer (SSL). The strategy encompasses the utilization of several dynamic analytic methodologies. The study entails the extensive analysis of 2213 different Android applications. The study found that out of the 360 apps analyzed, there are code vulnerabilities in some of the applications. The DC Droid framework was instrumental in identifying these weaknesses. This study highlights how SSL/TLS certificates can effectively detect system vulnerabilities.

In a publication (Boden, 2019), the authors introduced Service Monitor, a methodology designed to detect hosts and effectively identify malware across devices. Dynamic analytic methods were used in this framework. The Service Monitor tool collected data to create a model based on Markov chain theory principles. This model was aimed at monitoring the system service requests process. The feature vector was Markov, and it was employed to address classification problems with the help of the machine learning (ML) algorithms, including Random Forest (RF), k Neighbors (KNN), and Support Vector Machines (SVM). RF is an acronym that is used to denote Random Forest, k-nearest neighbors (KNN), as well as Support Vector Machines (SVM). Having used numerous datasets such as Andro Zoo, Drebin, and Malware Genome to train the model, it was the Random Forest (RF) method that showed a high level of effectiveness, and its accuracy rate was 96.7. The task was completed. It is imperative to acknowledge that specific benign applications have the potential to exhibit behavior like that of malware through the utilization of system services. The potential for misclassification by the model should be acknowledged since it is a plausible outcome.

A. Hybrid Analysis

The approach (Wang et al., 2020) develops a Hidden Markov Model (HMM) detector that identifies malware by utilizing small opcode sequences disassembling from the .apk file. After training using many code sequences extracted from malware, the HMM model under consideration computes a generative probability for every upcoming opcode sequence. Classification of the novel sequences will be based on the generative probabilities. Other works employ permission requests as a means of detection.

Salehi et al., (2019) employs an innovative NLP algorithm to produce fingerprints automatically from the dynamic actions of Android malware. Improved Android malware detection tasks with family classification and showed strong detection performance with excellent scalability when tested against a real-world data set. Feature engineering is required because most current NLP-based solutions use classical ML approaches. In contrast, strategy utilizes a hybrid sequence-based feature that does not need feature engineering and a sophisticated DL algorithm. As a consequence, the technique was able to capture more sensitive information, resulting in improved performance.

The hybrid malware detection approach described in the referenced paper (Li et al., 2015) incorporates the utilization of the Tree Augmented Naive Bayes (Tree TAN) model. In the mentioned paper (Li et al., 2015), a hybrid approach for detecting malware is described. This approach combines dynamic elements, such as permissions and system calls, to improve malware detection effectiveness. An example of this combination can be seen in the context of Application Programming Interface (API) calls. The researchers employed the Logistic Regression (LR) method in training the attributes. The tree augmented naive bayes (TAN) structure was built using datasets to come up with output associations. The datasets included GitHub and GP platforms. The main goal of the model was to categorize programs as either harmful or benign, achieving a classification accuracy rate of 0.97.

In 2021, (Zou et al., 2019) introduced a framework called DeepAMD that employs neural networks (ANNs) for detecting malware on Android devices. The study came to the conclusion that DeepAMD performs better than the methods in detecting malware at both the dynamic levels based on the CICAndMal2017 dataset. The framework had a 90% accuracy in classifying families on a layer and a 92.5% accuracy in category classification. Additionally, the model achieved a 93.4% accuracy in classification. On the layer, the architecture demonstrated an accuracy of 80.3% for category classification and 59% for identifying malware families.

After conducting an investigation into the utilization of machine learning and learning techniques for identifying malware, it was found that approximately 65% of the research papers surveyed relied on static analysis methods. In contrast, 15% of the studies employed dynamic analysis strategies. A mixed analytic methodology was implemented in the remaining 20% of trials.

B. Dataset

Table 1: Classifications of malware dataset

Category	Type	Count
Benign	Benign	965
Ako	Ransomware	260
Autorun. NE	Virus	335
Beinker. Y	Trojan Spy	315
Dalt. U	Backdoor	305
Drolinur. K	Worm	340
Egnof. D	Worm	355
Gand Dreef. E	Ransomware	275
Ganeld. S	Worm	345
Linkrey. MT	Adware	335
Necod. T	Trojan	325
Nerucol	Trojan Downloader	330
Nujit. A	Trojan Downloader	360
Installer	PUA	330
Play. Tech	PUA	345
QPas.GP	PWS	320
Quart	Trojan Spy	345
Resur. A	Virus	330
Sodi. A	Virus	305
Shimda. E	PWS	240
Sivi. l.K	Virus	350
Lark. M	Trojan Spy	345
Soltirne	Worm	350
Trickot. F	Trojan	365
Unrily. F	Trojan Downloader	245
Upaq. A	Trojan Downloader	395
Ureless. A	Trojan	315
Waboot. M	Backdoor	330
Yoofed. A	Worm	390
Zombie. S	Trojan	345

A total of 10,490 malevolent Portable Executable (PE) files were collected during the research endeavor. These files were subsequently classified into 30 discrete families. The files above were obtained from

Virus Share's website (Surendran et al., 2020), a repository of malware samples specifically intended for researchers specialized in information security. The files above were acquired by downloading from the specified website. Moreover, the website serves as a platform for facilitating the sharing of diverse file formats. Microsoft Malware Protection facilitated the execution of searches that identified many malware families, including the AKO family. This particular family has been shown to have associations with ransomware.

To facilitate a comprehensive investigation, a substantial quantity of EXE files sourced from an affiliated website that remained unaffected by any breach was systematically arranged (Imtiaz et al., 2021). The statistical data provided in table 1 were utilized to classify the benign profiles into thirty distinct groups. The main aim of this endeavor was to integrate the information from two different databases. Before forming the first data set, the samples will be converted into a visual representation. Conversely, the second dataset will be generated by eliminating sequences from the API calls executed by each sample. The goal was to utilize deep learning methodologies to effectively classify malware groups and distinguish between malicious software and harmless apps, incorporating instances of both malicious and benign characteristics. Subsequently, the samples above were evenly allocated among the 30 groups.

C. Preprocessing

The methodology outlined in references (Bhandary et al., 2022; Feijó et al., 2022; Kalash et al., 2018) was used to convert the collected samples, including both harmful and benign instances, into grayscale images suitable for constructing the dataset. At the start of the processing steps, each sample's binary representation was converted into an 8-bit vector to fulfill the necessary prerequisites for the subsequent processing stages. The first step undertaken during the implementation of the processing methodology included the execution of this particular phase. After completing the conversion method, the 8-bit vector was transformed into a visual depiction as a picture.

The final output is an image composed of a contiguous spectrum of integer values from 0 to 255. The number range from 0 to 255 is often used in representing colors, with zero symbolizing black and 255 symbolizing white, owing to the established association between these numerical values and these particular shades. Grayscale gradients are used to represent intermediate shades of gray that lie between the two extremes mentioned above. The height of a photograph may vary depending on its dimensions, but the width remains constant. When a malicious executable is displayed as a visual representation, it is much simpler to identify and differentiate the various components that make up the binary structure. This approach has a significant benefit.

The preprocessing of the samples, including both harmful and benign files, was conducted using the technique developed by the authors (Mallik et al., 2022). At the start of the processing step, every sample undergoes a conversion procedure to transform it into a grayscale picture. This is done as part of the staging. Figure 2 depicts the sequential stages included within the data translation procedure. Following the conversion of the binary sample, it was then represented as an 8-bit vector, which served as an intermediary stage in the process. The use of an 8-bit vector was necessary at this period. Afterwards, an exact copy of the specimen is made, followed by the next step in the process. The original image was created using an 8-bit vector, ensuring that each pixel is accurately reproduced based on its intended representation.

4 Implementation

The pictures were initially normalized as part of the method to prepare the picture data. This involved resizing the images and standardizing the resolution to 150 by 150 pixels. The data gathering process was effectively carried out by dividing the information into three components, ensuring data for each specific purpose. For training purposes, 80% of the dataset was allocated to the training subset, while 10% each was assigned to the testing and validation subsets.

To enhance its applicability in scenarios and improve effectiveness, reorganization was performed on the model's architecture. The considered architecture consists of three layers in a Convolutional Neural Network (CNN). The first layer comprises sixty-four filters with dimensions of three by three. The second layer includes 128 filters, with dimensions of three-by-three pixels.

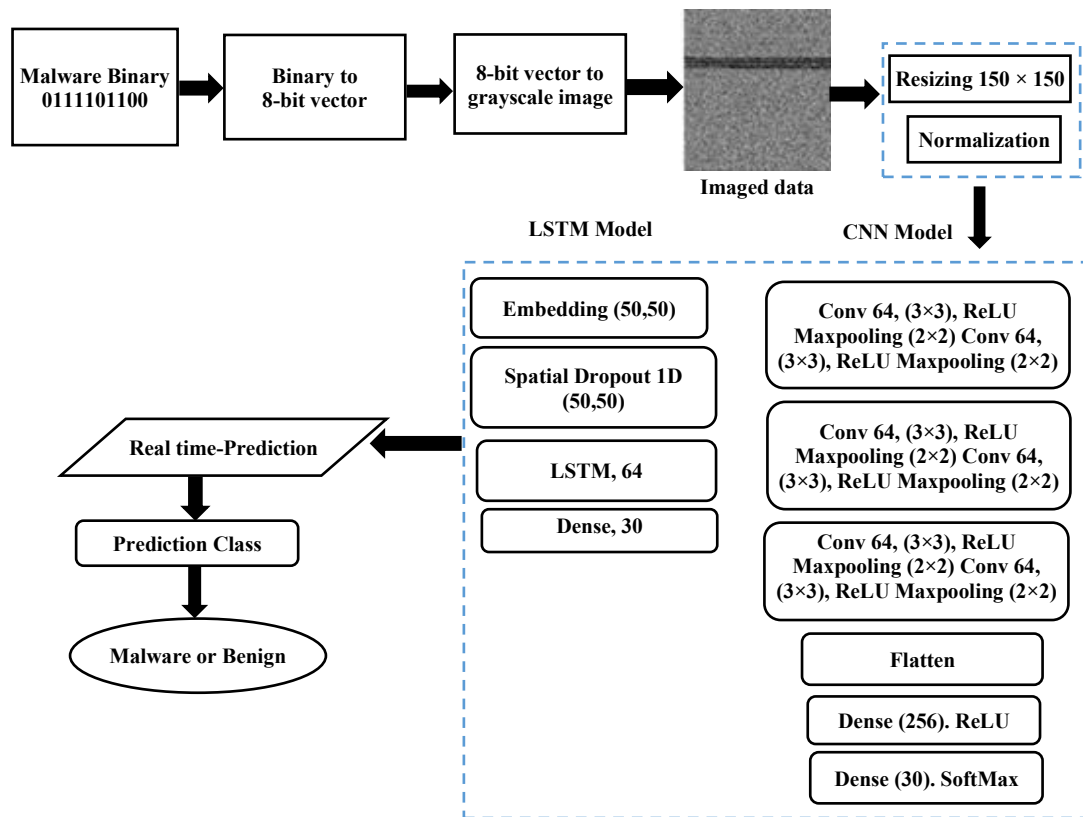


Figure 2: Proposed system

The proposal utilizes a structured approach to identify and categorize data efficiently. The primary focus, during the selection of models, is to ensure accuracy. The model's capabilities are assessed by evaluating both the number of parameters and the resources utilized in simulations. To prepare the image data, the photos are initially normalized by Adjusting the dimensions and converting to a resolution of 150 by 150 pixels.

The data collection process effectively divides the information into three components, providing data for each specific objective. As for allocation, 10% is dedicated to testing and validation subsets, while 80% is allocated for training purposes.

The reorganization was done to improve the model's ability to be used in situations and make it more effective. The architecture being considered includes three layers within the Convolutional Neural

Network (CNN). The first layer has sixty-four filters, each with a size of three by three. The second layer contains 128 filters, with a 3 by 3-pixel size. The third layer includes 256 3-by-3-dimensional filters. Such layers are made possible through the assistance of the Rectified Linear Unit (ReLU) activation function and linear transformations. Max pooling layers are then narrowed down to six to reduce the size of the attributes.

After applying the pooling layers, there is a fully connected layer with 256 neurons. The implementation of this step aims at enhancing. The quality of representation of features improves learning. Lastly, there is the output layer, with 30 neurons, in order to fit the 30 tasks.

To address overfitting, dropout regularization is applied after each Max pooling layer and in the connected layer. In this study, a sparse category cross-entropy loss function is used to measure the difference between expected and actual class labels (Falana et al., 2022). Adam optimizer is also significant as far as optimizing the model and ensuring that the parameters are optimized are concerned. A deeper exploration of these techniques will be provided in the sections of this book.

To prevent and address the problem of overfitting, the SpatialDropout1D algorithm is used during the training process. This algorithm focuses explicitly on the data to ensure that excessive attention is not given to certain features. Consequently, the flexibility of the model to change under various circumstances, as well as its capacity to process data occurrences efficiently, is enhanced.

A. Proposed CNN Architecture

The CNN, or a network, is made up of three major layers, including the convolution layer, the fully connected layer, and the pooling layer. The layers are interrelated in order to create a two-activation map with the help of kernels. The convolution process helps to complete this task, whereby the kernel is applied systematically to the entire spatial domain of the input. This takes place in every level. One potential approach to reducing the number of activation parameters involves down-sampling the input dimension and using pooling layers. Subsequently, the interconnected layers will collaborate to construct a classification model using the provided data (shown in figure 3).

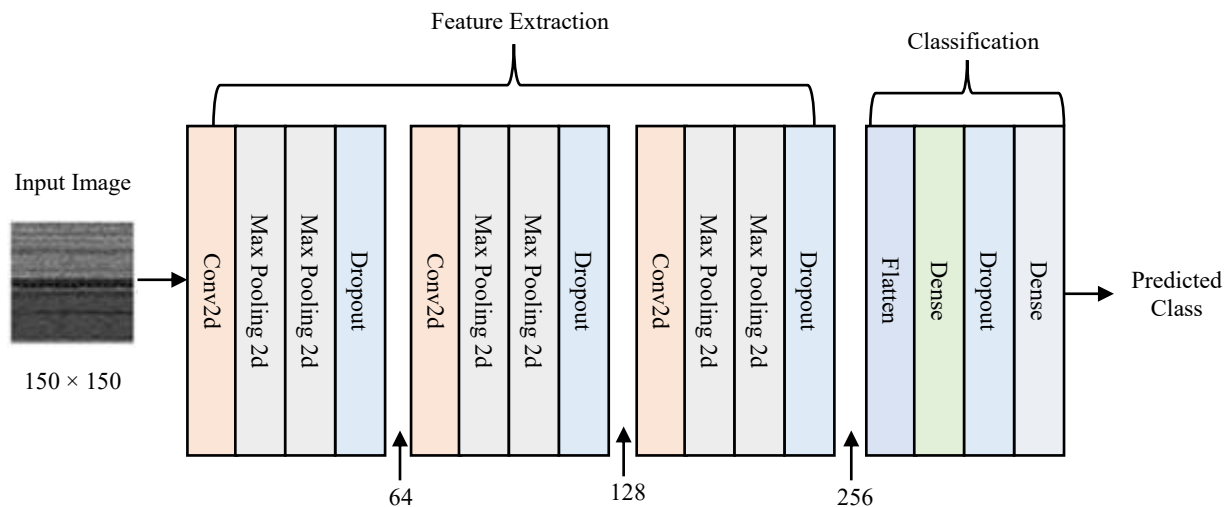


Figure 3: CNN implementation

B. LSTM

Long Short-Term Memory (LSTM) model is a framework that is created to extend the cognitive processes and the sustainability of knowledge in the case of a wide range of inputs. It is the stratum that is widely known to be able to identify associations that are spatially distant. The LSTM (Long Short-Term Memory) layer is used to make sure that processing and storage of the information available as inputs take place.

The LSTM architecture includes three layers: an embedding layer, an LSTM layer with 64 memory cells, and a dense layer that incorporates a SoftMax classifier. The given arrangement permits the administration of classes. The unique trait of this neural network is that it has the ability to process information through the feedback connections, unlike traditional feed-forward neural networks. An LSTM cell has the three basic elements, which include an input gate, an output gate, and a forget gate.

The main idea of the embedding layer is to map the input data sequences into a vector space, which is effective in terms of capturing the meaning of content. The LSTM layer has 64 memory cells that have activation functions to facilitate easy processing and storage of information.

C. Evaluation

To create subsets for analysis, the entire dataset was divided into two parts: a training dataset and a testing dataset. The training dataset accounts for 70% of the data, while 15% is reserved for performance evaluation. Further division was done within the training data to enhance its usability in validation analyses. In this subdivision 70% is allocated to validation data and 15% to training data. Establishing these subgroups (training, testing, and validation) plays a role in classification tasks. Figure 4 provides a representation of these subgroups.

The LSTM architecture works well for tasks involving time series data, such as classification and prediction, because it can effectively capture relationships. This proves beneficial when significant events exhibit time delays. Long Short-Term Memory (LSTM) network is not only good at storing information, but can also last longer, reflecting its capability in perceiving and interpreting the patterns of a sequence in a temporal manner.

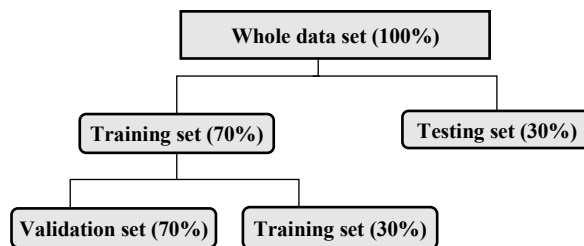


Figure 4: Dataset splitting

Algorithm 1: Hybrid CNN-LSTM Model for Malware Detection

Input:

- $D = \{X, Y\}$: Training dataset with inputs X (malware sample data) and labels Y (malware classification labels).
- $\Phi = \{\phi_1, \phi_2, \dots, \phi_n\}$: Set of logical constraints related to malware classification and security.
- $\theta = \{W, B\}$: Neural network weights and biases for CNN and LSTM models.

- η : Learning rate for model optimization.
- T : Number of training epochs.
- λ : Weighting factor for logical consistency in classification and malware detection.
- $A = \{C, L\}$: Hyperparameters for CNN layers (C : number of filters, L : filter size).

Output:

- Trained CNN-LSTM model θ that accurately classifies malware and satisfies security and privacy constraints.

Pseudocode:

1. Initialize model parameters θ for CNN and LSTM models:
 - o Initialize CNN layers with C filters and L filter sizes.
 - o Initialize LSTM layers with appropriate memory cells.
 - o Encode logical constraints Φ into classification and security protocols.
2. For each epoch $t = 1$ to T :
3. a. For each training sample $(x, y) \in D$:
4. i. Pass input x through the CNN model to extract spatial features.
5. ii. Apply the LSTM model to the CNN output for sequential feature learning.
6. iii. Compute the malware prediction using the softmax layer: $f(x) = \text{softmax}(\text{CNN_LSTM}(x, \theta))$.
7. iv. If the prediction is malicious:
8. - Trigger security alert and block access (as per model design).
9. v. Else:
10. - Continue with data processing (normal access granted).
 - b. Apply real-time anomaly detection:
 - i. Analyze network traffic using an anomaly detection model $A(x)$.
 - ii. If anomaly detected:
 - Trigger security alert and block access.
 - iii. If no anomaly detected:
 - Continue monitoring network traffic and data.
 - c. Privacy Protection:
 - i. Apply anonymization techniques to sensitive data (e.g., student or personal data) before storage.
 - ii. Ensure privacy compliance by using $P(x, \theta)$ data anonymization rules.
 - d. Compute task loss:
 - i. Data classification loss: $L_{\text{data}} = \text{CrossEntropy}(f(x), y)$.
 - ii. Privacy loss: $L_{\text{privacy}} = \text{PrivacyLoss}(x, \lambda)$.
 - iii. Logical consistency loss: $L_{\text{security}} = 0$.
 - e. For each security formula $\phi \in \Phi$:
 - i. Compute logical satisfaction using fuzzy logic:
 $G(P, Q) = \max(1 - G(P), G(Q))$.
 - ii. Accumulate security loss:

$$L_{security} \leftarrow L_{security} + (1 - G(\phi)).$$

f. Aggregate total loss:

$$L_{total} = L_{data} + \lambda * L_{security}.$$

g. Update parameters via backpropagation:

$$\theta \leftarrow \theta - \eta * \nabla_{\theta} L_{total}.$$

11. End For (Repeat for all epochs).

12. Return trained model θ that satisfies both statistical accuracy (malware classification) and logical consistency (security and privacy constraints).

The following algorithm 1 is used to train a CNN-LSTM hybrid model to detect malware without violating security and privacy limitations. The training data D (X (malware sample data) and Y (labels to classify malware)) is initialized, and logical constraints Φ according to which malware classification and security will be performed. The CNN-LSTM model is trained with weights θ , denoted as θ , and a learning rate η , denoted as η . The model has T training epochs, whereby at each epoch, the malware is classified using CNN layers, and then sequential pattern recognition is done using the LSTM layers. This algorithm entails the consideration of privacy through the use of the weighting factor, λ , which is used to guarantee the logical consistency of the model so that the model is accurate in classifying the malware and at the same time complies with the security and privacy regulations. The ultimate product is a trained CNN-LSTM model θ that can identify malware with high precision and meet classification as well as security factors.

5 Results and Discussion

The final outcome will depend on how the model is trained, which in turn affects the precision of the generated results.

A. Confusion Matrix

This dataset contains metrics such as recall, F1 score, accuracy, and precision. These measurements provide insights into the effectiveness of the algorithms being used (In equations (1-4)).

By utilizing a confusion matrix, assess performance measures like Positive (TP), False Negative (FN), False Positive (FP), and True Negative (TN). Evaluating these metrics is crucial for determining how well the classifier can distinguish between instances in the dataset.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$Precision = \frac{TP}{TP + FP} \quad (3)$$

$$F1 \text{ score} = 2 * \frac{Recall * Precision}{Recall + Precision} \quad (4)$$

B. Experimental Result

After conducting testing over more than 50 iterations, the CNN model achieved an impressive accuracy rate of 98%. Comparatively, the LSTM model had an accuracy rate of 97%. The accuracy and validation accuracy of the two models are predicted and graphically illustrated in figure 5 and figure 6. The

graphical representation of loss and validation loss provides context regarding the performance of these models. Table 2 presents the rating metrics obtained by applying a Convolutional Neural Network (CNN) model to this dataset. These metrics include accuracy, precision, recall, and F1 score.

It was noted that in the process of testing, the CNN model always had an accuracy rate of 98% and the LSTM model had a slightly lower rate of 97%. Figure 5 and figure 6 gives the images that represent the performance of both models regarding their accuracy and validation performance. These visual illustrations offer insights into how the models perform, showing measurements of loss and validation loss. Furthermore, table 2 shows the rating metrics achieved by applying a Convolutional Neural Network (CNN) model to the dataset. These metrics include accuracy, precision, recall, and F1 score.

To comprehensively evaluate metrics like F1 score, it is crucial to employ techniques such as the confusion matrix and categorization report.

Table 2: Classification metrics for pre-trained and proposed models

Class	LSTM				CNN			
	Accuracy	Precision	Recall	F1-score	Acc	Pre	Rec	F1
Benign	0.99	0.96	0.94	0.95	0.99	0.99	0.94	0.96
Ako	1.00	0.91	0.95	0.93	1.00	1.00	1.00	1.00
Autorun.NE	1.00	0.93	0.93	0.93	1.00	0.97	0.89	0.93
Beinker. Y	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.99
Dalt. U	1.00	0.96	0.96	0.96	1.00	0.98	1.00	0.99
Drolinur. K	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Egnof.D	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.99
GandDreef.E	1.00	0.93	1.00	0.96	1.00	0.97	1.00	0.99
Ganeld.S	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Linkrey. MT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Necod.T	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Nerucol	1.00	0.96	1.00	0.98	1.00	0.97	1.00	0.99
Nujit. A	1.00	0.97	1.00	0.98	1.00	1.00	1.00	1.00
Installer	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Play. Tech	1.00	1.00	0.95	0.98	1.00	1.00	1.00	1.00
QPas.GP	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Quart	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Resur.A	1.00	0.91	1.00	0.95	1.00	0.94	1.00	0.97
Sodi. A	1.00	0.90	0.86	0.88	1.00	0.94	0.88	0.91
Shimda. E	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Sivi. l.K	0.99	1.00	0.71	0.83	1.00	1.00	0.95	0.97
Lark.M	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Soltirne	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Trickot. F	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Unrily. F	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Upaq.A	1.00	0.96	1.00	0.98	1.00	0.94	1.00	0.97
Urelass. A	1.00	0.96	0.93	0.95	1.00	0.95	0.97	0.96
Waboot.M	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Yoofed. A	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Zombie. S	0.99	0.84	0.96	0.90	1.00	0.88	0.97	0.92
Accuracy	0.97				0.98			
Macro Avg		0.97	0.97	0.97		0.98	0.99	0.98
Weighted Avg.		0.97	0.97	0.97		0.98	0.98	0.98

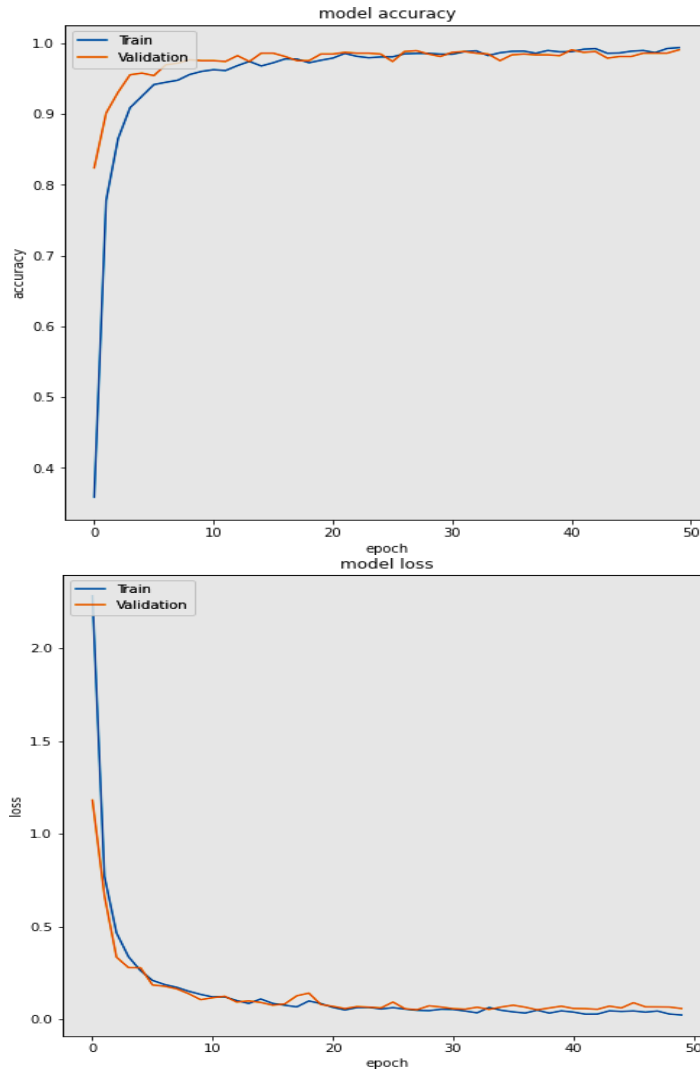
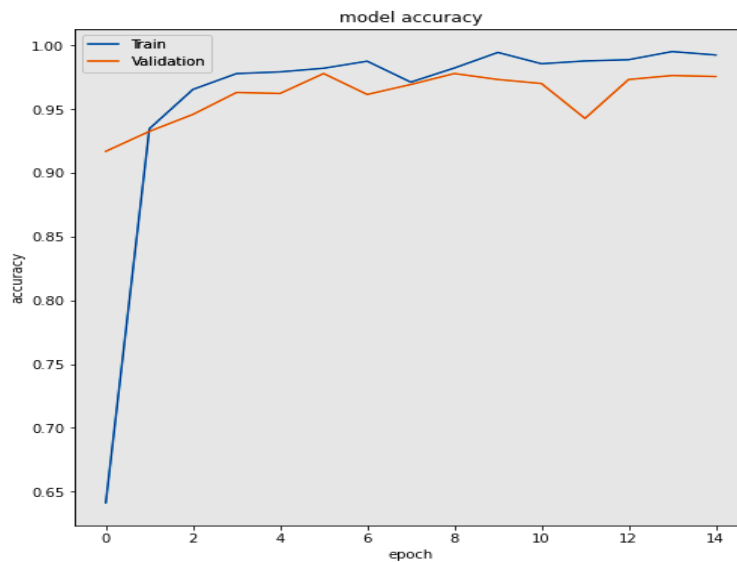


Figure 5: Plots of accuracy and loss for the CNN model



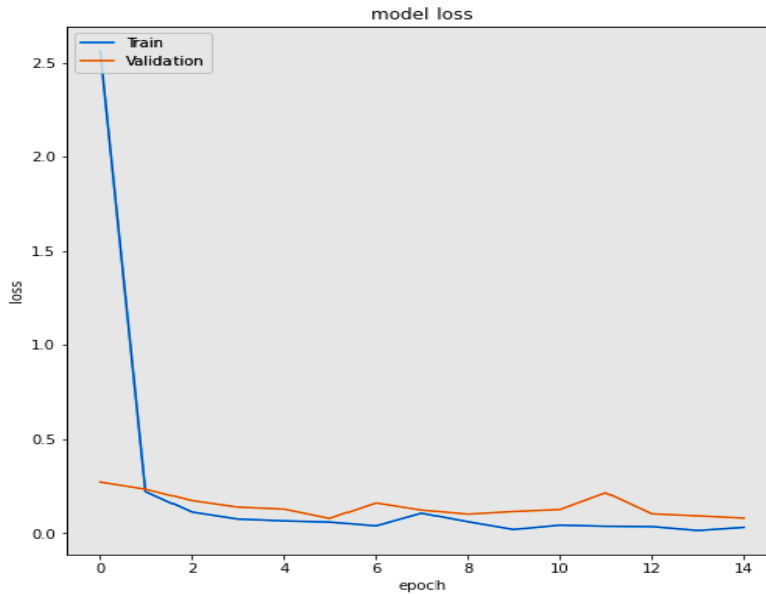


Figure 6: Plots of accuracy and loss for the LSTM model

The results of the study indicated that there was a level of accuracy for all categories within the groups. This was determined by comparing the expected values with the values that revealed differences between the two.

The research involved analyzing a dataset mainly composed of photographs, which served as the source of data. The models were evaluated in terms of Convolutional Neural Network (CNN) as well as Long Short-term memory (LSTM). Both of these forms of architecture are neural networks. The photos were preprocessed before being incorporated into the models. In the preprocessing phase, the images were rescaled to 150 by 150 pixels. It is important to note that the models were developed with the assumption of training using color images, utilizing the full spectrum of all three color channels. This aspect was taken into account throughout the design process of the models. Conversely, the dataset used in this study only included monochromatic images depicting the highlighted samples.

Upon completing the dataset normalization procedure, it was partitioned into three separate subsets: 70% for training, 15% for testing, and 15% for validation.

The results of the research suggest that the CNN-based model developed demonstrates superiority over the LSTM-based equivalent, particularly in terms of parameter creation and memory usage. The benefit above becomes readily apparent due to the use of photographs with input dimensions of 150×150 pixels, as seen in figure 3. This facilitates the identification of the benefit.

Figure 7 represents the comparison of the performance of two deep learning-based models, Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), in Android malware detection. The graph shows that the CNN model has a marginally better accuracy rate of 98% as compared to the LSTM model, which has an accuracy of 97%. This shows the CNN model is an efficient way of enhancing the security measures of the mobile networks by identifying malicious software effectively.

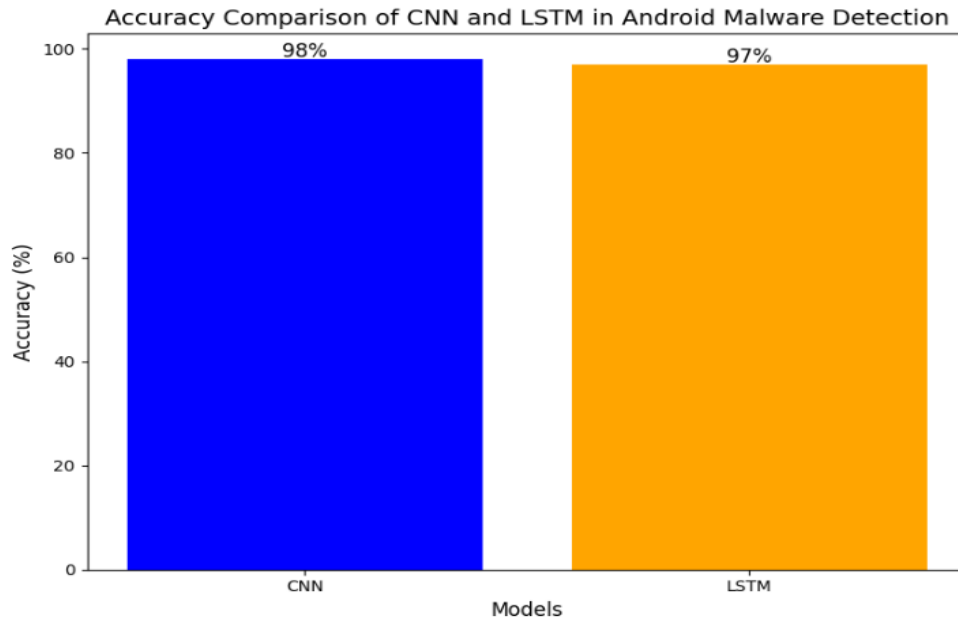


Figure 7: Accuracy comparison of CNN and LSTM models in android malware detection

6 Conclusion

This study aims to apply two machine-learning models, Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), to identify common features in malicious Android applications. The effectiveness of these models was evaluated using various performance metrics, considering measures for handling missing data. The CNN and LSTM models achieved prediction accuracies of 97% and 98%, respectively, with CNN showing the highest improvement in accuracy compared to other reviewed methodologies.

The research demonstrates how these machine learning models can address critical security issues in mobile networks and ubiquitous computing systems, particularly in protecting mobile devices from malware attacks. By leveraging the capabilities of deep learning, such as CNN and LSTM, the study contributes to strengthening security measures and safeguarding sensitive information within mobile networks. This approach fosters a more secure, efficient, and reliable mobile environment, as mobile technology continues to evolve.

Future work can enhance the models by incorporating reinforcement learning to adapt to changing malware threats. Additionally, integrating anomaly detection or behavior analysis could further improve accuracy. Expanding the models' scalability to larger datasets and enabling real-time integration into mobile security applications is another avenue for development. Furthermore, incorporating cross-platform malware detection could extend protection to other operating systems, offering broader device coverage.

Acknowledgment

I appreciate my colleagues on the teaching and administrative staff at Al-ayen University in Nasiriyah for the tremendous support, which gave the author vital knowledge and understanding of the research.

References

- [1] Alzaylaee, M. K., Yerima, S. Y., & Sezer, S. (2020). DL-Droid: Deep learning based android malware detection using real devices. *Computers & Security*, *89*, 101663. <https://doi.org/10.1016/j.cose.2019.101663>
- [2] Aravind, B., Harikrishnan, S., Santhosh, G., Vijay, J. E., & Saran Suaji, T. (2023). An Efficient Privacy-Aware Authentication Framework for Mobile Cloud Computing. *International Academic Journal of Innovative Research*, *10*(1), 1–7.
- [3] Ashawa, M., & Morris, S. (2021). Modeling correlation between android permissions based on threat and protection level using exploratory factor plane analysis. *Journal of Cybersecurity and Privacy*, *1*(4), 704-742. <https://doi.org/10.3390/jcp1040035>
- [4] Bayazit, E. C., Sahingoz, O. K., & Dogan, B. (2020, June). Malware detection in android systems with traditional machine learning models: a survey. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)* (pp. 1-8). IEEE. <https://doi.org/10.1109/HORA49412.2020.9152840>
- [5] Bhandary, P., Vieson, C., Kiendrebeogo, A., Adetunji, I., Joyce, R. J., Eren, M. E., & Nicholas, C. (2022, March). Malware Antivirus Scan Pattern Mining via Tensor Decomposition. In *Malware Technical Exchange Meeting* (Vol. 14, p. 35).
- [6] Bhat, P., Behal, S., & Dutta, K. (2023). Machine learning and deep learning techniques for detecting malicious android applications: An empirical analysis. *Proceedings of the Indian National Science Academy*, *89*(3), 429-444. <https://doi.org/10.1007/s43538-023-00182-w>
- [7] Boden, E. (2019). MagPiBRID: A general approach to integrate static analysis into IDEs and editors (Tool Insights Paper). In *33rd European Conference on Object-Oriented Programming (ECOOP 2019)* (pp. 1–16). Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
- [8] Cai, H., Meng, N., Ryder, B., & Yao, D. (2018). Droidcat: Effective android malware detection and categorization via app-level profiling. *IEEE Transactions on Information Forensics and Security*, *14*(6), 1455-1470. <https://doi.org/10.1109/TIFS.2018.2879302>
- [9] Catal, C., Giray, G., & Tekinerdogan, B. (2022). Applications of deep learning for mobile malware detection: A systematic literature review. *Neural Computing and Applications*, *34*(2), 1007-1032. <https://doi.org/10.1007/s00521-021-06597-0>
- [10] Chan, L., Morgan, I., Simon, H., Alshabanat, F., Ober, D., Gentry, J., ... & Cao, R. (2019, June). Survey of AI in cybersecurity for information technology management. In *2019 IEEE technology & engineering management conference (TEMSCON)* (pp. 1-8). IEEE. <https://doi.org/10.1109/TEMSCON.2019.8813605>
- [11] Choudhary, M., & Kishore, B. (2018, January). Haamd: Hybrid analysis for android malware detection. In *2018 International Conference on Computer Communication and Informatics (ICCCI)* (pp. 1-4). IEEE. <https://doi.org/10.1109/ICCCI.2018.8441295>
- [12] Ciaramella, G., Iadarola, G., Mercaldo, F., Storto, M., Santone, A., & Martinelli, F. (2022, August). Introducing quantum computing in mobile malware detection. In *Proceedings of the 17th international conference on availability, reliability and security* (pp. 1-8). <https://doi.org/10.1145/3538969.3543816>
- [13] Elayan, O. N., & Mustafa, A. M. (2021). Android malware detection using deep learning. *Procedia Computer Science*, *184*, 847-852. <https://doi.org/10.1016/j.procs.2021.03.106>
- [14] Falana, O. J., Sodiya, A. S., Onashoga, S. A., & Badmus, B. S. (2022). Mal-Detect: An intelligent visualization approach for malware detection. *Journal of King Saud University Computer and Information Sciences*, *34*(5), 1968–1983. <https://doi.org/10.1016/j.jksuci.2019.10.004>
- [15] Feijó, E. V. R. D. S., Barbosa, B. L., van den Berg, C., & Oliveira, L. M. D. (2022). Genetic diversity of *Lippia origanoides* Kunth. in natural populations using ISSR markers. *Ciência e Agrotecnologia*, *46*, e000822. <https://doi.org/10.1590/1413-7054202246000822>

- [16] Ghaffarian, S. M., & Shahriari, H. R. (2017). Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM computing surveys (CSUR)*, 50(4), 1-36. <https://doi.org/10.1145/3092566>
- [17] Giji Kiruba, D., Benita, J., & Rajesh, D. (2023). A Proficient Obtrusion Recognition Clustered Mechanism for Malicious Sensor Nodes in a Mobile Wireless Sensor Network. *Indian Journal of Information Sources and Services*, 13(2), 53–63. <https://doi.org/10.51983/ijiss-2023.13.2.3793>
- [18] Gohari, M., Hashemi, S., & Abdi, L. (2021, May). Android malware detection and classification based on network traffic using deep learning. In *2021 7th International Conference on Web Research (ICWR)* (pp. 71-77). IEEE. <https://ieeexplore.ieee.org/xpl/conhome/9443089/proceeding>
- [19] Habchi, S., Blanc, X., & Rouvoy, R. (2018, September). On adopting linters to deal with performance concerns in android apps. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering* (pp. 6-16). <https://doi.org/10.5281/zenodo.1320453>
- [20] Imtiaz, S. I., ur Rehman, S., Javed, A. R., Jalil, Z., Liu, X., & Alnumay, W. S. (2021). DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network. *Future Generation computer systems*, 115, 844-856. <https://doi.org/10.1016/j.future.2020.10.008>
- [21] Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D., Wang, Y., & Iqbal, F. (2018, February). Malware classification with deep convolutional neural networks. In *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)* (pp. 1-5). IEEE. <https://doi.org/10.1109/NTMS.2018.8328749>
- [22] Kim, Y. K., Lee, J. J., Go, M. H., Kang, H. Y., & Lee, K. (2022). A systematic overview of the machine learning methods for mobile malware detection. *Security and Communication Networks*, 2022(1), 8621083. <https://doi.org/10.1155/2022/8621083>
- [23] Li, L., Bissyandé, T. F., Papadakis, M., Rasthofer, S., Bartel, A., Octeau, D., ... & Traon, L. (2017). Static analysis of android apps: A systematic literature review. *Information and Software Technology*, 88, 67-95. <https://doi.org/10.1016/j.infsof.2017.04.001>
- [24] Li, Y., Shen, T., Sun, X., Pan, X., & Mao, B. (2015, October). Detection, classification and characterization of android malware using api data dependency. In *International Conference on Security and Privacy in Communication Systems* (pp. 23-40). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-28865-9_2
- [25] Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., & Liu, H. (2020). A review of android malware detection approaches based on machine learning. *IEEE access*, 8, 124579-124607. <https://doi.org/10.1109/ACCESS.2020.3006143>
- [26] Mallik, A., Khetarpal, A., & Kumar, S. (2022). ConRec: malware classification using convolutional recurrence. *Journal of Computer Virology and Hacking Techniques*, 18(4), 297-313. <https://doi.org/10.1007/s11416-022-00416-3>
- [27] Mazaed Alotaibi, F., & Fawad. (2022). A multifaceted deep generative adversarial networks model for mobile malware detection. *Applied Sciences*, 12(19), 9403. <https://doi.org/10.3390/app12199403>
- [28] Mercaldo, F., Ciaramella, G., Iadarola, G., Storto, M., Martinelli, F., & Santone, A. (2022). Towards explainable quantum machine learning for mobile malware detection and classification. *Applied Sciences*, 12(23), 12025. <https://doi.org/10.3390/app122312025>
- [29] Merlo, A., Ruggia, A., Sciolla, L., & Verderame, L. (2021). You shall not repackage! demystifying anti-repackaging on android. *Computers & Security*, 103, 102181. <https://doi.org/10.1016/j.cose.2021.102181>
- [30] Muzaffar, A., Hassen, H. R., Lones, M. A., & Zantout, H. (2022). An in-depth review of machine learning based Android malware detection. *Computers & Security*, 121, 102833. <https://doi.org/10.1016/j.cose.2022.102833>

- [31] Nguyen, D. C., Wermke, D., Acar, Y., Backes, M., Weir, C., & Fahl, S. (2017, October). A stitch in time: Supporting android developers in writing secure code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1065-1077). <https://doi.org/10.1145/3133956.3133977>
- [32] Odusami, M., Abayomi-Alli, O., Misra, S., Shobayo, O., Damasevicius, R., & Maskeliunas, R. (2018, October). Android malware detection: A survey. In *International conference on applied informatics* (pp. 255-266). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-01535-0_19
- [33] Ranganathan, P., & Bhattacharya, M. (2021). Design and Analysis of Intelligent Home Automation Using the IoT Applications. *International Academic Journal of Science and Engineering*, 8(2), 21–25. <https://doi.org/10.71086/IAJSE/V8I2/IAJSE0812>
- [34] Sahin, M., & Bahtiyar, S. (2020, November). A survey on malware detection with deep learning. In *13th international conference on security of information and networks* (pp. 1-6). <https://doi.org/10.1145/3433174.3433609>
- [35] Salehi, M., Amini, M., & Crispo, B. (2019, November). Detecting malicious applications using system services request behavior. In *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services* (pp. 200-209). <https://doi.org/10.1145/3360774.3360805>
- [36] Santos, I., Devesa, J., Brezo, F., Nieves, J., & Bringas, P. G. (2013). Opem: A static-dynamic approach for machine-learning-based malware detection. In *International joint conference CISIS'12-ICEUTE' 12-SOCO' 12 special sessions* (pp. 271-280). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-33018-6_28
- [37] Surendran, R., Thomas, T., & Emmanuel, S. (2020). A TAN based hybrid model for android malware detection. *Journal of Information Security and Applications*, 54, 102483. <https://doi.org/10.1016/j.jisa.2020.102483>
- [38] Wang, Y., Xu, G., Liu, X., Mao, W., Si, C., Pedrycz, W., & Wang, W. (2020). Identifying vulnerabilities of SSL/TLS certificate verification in Android apps with static and dynamic analysis. *Journal of Systems and Software*, 167, 110609. <https://doi.org/10.1016/j.jss.2020.110609>
- [39] Yu, B., Fang, Y., Yang, Q., Tang, Y., & Liu, L. (2018). A survey of malware behavior description and analysis. *Frontiers of Information Technology & Electronic Engineering*, 19(5), 583-603. <https://doi.org/10.1631/FITEE.1601745>
- [40] Zhu, R., Li, C., Niu, D., Zhang, H., & Kinawi, H. (2018). Android malware detection using large-scale network representation learning. *arXiv preprint arXiv:1806.04847*.
- [41] Zou, K., Luo, X., Liu, P., Wang, W., & Wang, H. (2019, October). ByteDroid: Android malware detection using deep learning on bytecode sequences. In *Chinese Conference on Trusted Computing and Information Security* (pp. 159-176). Singapore: Springer Singapore. https://doi.org/10.1007/978-981-15-3418-8_12

Authors Biography



Dr. Rafid Sagban is currently Dean of the Technical Engineering College at Al-Ayen Iraqi University (Thi-Qar, Iraq) and also affiliated with the University of Babylon, Hilla, where he previously studied and taught. He holds a B.Sc. in Computer Science from University of Babylon (2001), a Higher Diploma in Data Security (Baghdad, 2002), and both his M.Sc. and Ph.D. in Information Technology from Universiti Utara Malaysia (UUM), completed in 2011 and 2015 respectively. Dr. Sagban's research and development experience spans over 15 years in academia and industry, combining algorithm analysis, swarm intelligence, business intelligence, internetworking, web optimization, and computational intelligence. He has authored more than 50 publications, notably in metaheuristic algorithms

(e.g., ant-colony optimization, evolutionary computation), data mining, and optimization techniques, often applied to problems such as combinatorial optimization, classification, and network security. Dr. Sagban is also actively involved with academic-professional bodies: he is (or has been) member of the IEEE Computational Intelligence Society (Malaysia Section), part of the Machine Intelligence Research Labs (USA), the Iraqi Association for IT Specialists, and serves on the editorial board of the Journal of University of Babylon (among others).



Dr. Rana Hikmet Tobia Saloom is affiliated with the Ministry of Higher Education and Scientific Research, Iraq. Based on the affiliation you provided, she collaborates with the scientific research community in Iraq. At present, I did not find publicly accessible detailed academic profile or verified photo for Dr. Rana Saloom in open sources. This suggests that her academic presence online may be limited or not indexed in major research databases. If you like, I can run a deeper search (including Arabic-language sources) to attempt to locate her CV and photograph, but I cannot guarantee success.



Dr. Naseer Ali Hussien is an Assistant Professor at the Scientific Research Center, Al-Ayen Iraqi University, Thi-Qar, Iraq, and the Department of Computer Science, College of Education for Pure Science, University of Thi-Qar, Nasiriyah, Iraq. He holds a Ph.D. in Computer Science, focusing on algorithms and computational methods used in data processing and optimization. Dr. Hussien's primary research interests lie in computational intelligence, data mining, and the development of optimization algorithms for real-time data analysis. His research has contributed to the improvement of various computational models and algorithms, with a particular emphasis on optimization techniques for complex systems. He has explored how these methods can be used to enhance decision-making processes in fields such as transportation systems, energy management, and software engineering. Dr. Hussien has published numerous articles and research papers in well-regarded academic journals and conferences. He has actively contributed to the growth of computational intelligence research in Iraq and has collaborated with researchers worldwide to advance knowledge in the optimization of large-scale data systems.