Computers, Materials & Continua
DOI: 10.32604/cmc.2023.032984
Article

Tech Science Press

Malware Detection in Android IoT Systems Using Deep Learning

Muhammad Waqar[1], Sabeeh Fareed[1], Ajung Kim[2,*], Saif Ur Rehman Malik[3], Muhammad Imran[1] and Muhammad Usman Yaseen[1]

[1]Department of Computer Science, COMSATS University, Islamabad, 45550, Pakistan
[2]Department of Optical Engineering, Sejong University, Seoul, 143747, Korea
[3]Cybernetica AS, Tallinn, 12618, Estonia
*Corresponding Author: Ajung Kim. Email: akim@sejong.ac.kr
Received: 03 June 2022; Accepted: 09 August 2022

Abstract: The Android Operating System (AOS) has been evolving since its inception and it has become one of the most widely used operating system for the Internet of Things (IoT). Due to the high popularity and reliability of AOS for IoT, it is a target of many cyber-attacks which can cause compromise of privacy, financial loss, data integrity, unauthorized access, denial of services and so on. The Android-based IoT (AIoT) devices are extremely vulnerable to various malwares due to the open nature and high acceptance of Android in the market. Recently, several detection preventive malwares are developed to conceal their malicious activities from analysis tools. Hence, conventional malware detection techniques could not be applied and innovative countermeasures against such anti-detection malwares are indispensable to secure the AIoT. In this paper, we proposed the novel deep learning-based real-time multiclass malware detection techniques for the AIoT using dynamic analysis. The results show that the proposed technique outperforms existing malware detection techniques and achieves detection accuracy up to 99.87%.

Keywords: Android; cyber security; deep learning; internet of things; malware detection

1 Introduction

Recently, the Internet of Things (IoT) concept has emerged, as globalization and connectivity of smart devices have exponentially increased. The IoT devices are used for sensing, monitoring, controlling, intelligently data sharing, and so on without human interaction. The IoT concept has been applied in smart-world systems such as smart grids, smart transportation, smart agriculture, smart cities, smart healthcare, and smart public safety. Most of the devices in the IoT environment are Android-driven due to flexibility, reliability, and hardware support. As a result, the utilization of Android-based IoT (AIoT) has significantly increased in the market [1]. According to [2], nearly 3 billion Android devices are in use and active to access global connectivity through the internet. It is reported that Android-based devices create more network traffic as compared to its competitor Apple's iOS devices [3]. Due to ever increased utilization of Android as an operating system in IoTs, the risk

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

of attacking such devices on the internet is drastically increased. It is possible to bypass the default Google Play Store protection policies and protocols using the malwares [4]. Therefore, rapid evaluation schemes are required to detect the presence of malware on AIoT. On the other hand, it is challenging to identify and block Android malware traffic with high precision using the existing malware detection schemes because existing antivirus and malware software can only identify those harming and spying programs that are programmed in them [5].

In today's communication networks, classifying network traffic in AIoT devices is very critical because of the concurrent transmission and reception of high-speed packets from multiple applications. As a result, hackers are increasingly looking for ways to exploit the vulnerabilities in Android applications. However, the security experts are driven to design sophisticated malware architectures in order to make use of existing operating systems and application flaws [6]. The malicious software such as viruses, spyware, worms, keyloggers, rootkits and ransomware are intended to gain unauthorized access and harm the AIoT systems. The compromise of such devices results in information theft, privacy violation, privilege escalation, SQL injections and remote access. Attackers exploit the dense internet connections to infect smart devices remotely, encrypt sensitive data, install backdoors, and transmit self-propagating programs to infect new systems. There are several instances when the static analysis of malicious code on devices is performed later or at a scheduled time, but the malicious code has already been executed by the virus. Advanced malwares are capable of massive destruction on Android-based IoTs because of their complexity and sophistication [7]. Hence, they have the potential to significantly impact both enterprises and mobile users. Mostly, the undetectable nature of malwares makes them able to get root access of Android devices, making it infeasible to track down.

There are two methods for detecting malware in Android devices such as static analysis and dynamic analysis [8]. The static malware analysis does not use the execution of the application during the testing process and can detect potentially malicious activity in the early stages before the application is executed. This approach employs automated tools to collect technical indicators or parameters which are used to determine whether the application is suitable to use or not. However, static malware detection is not an optimal solution as it needs to decompile the Android Package Kit (APK) and other files for the identification of malwares. Hence, real-time malware identification is an important concern in static malware analysis. Therefore, dynamic malware analysis is a more appealing solution for analyzing the behavior of Android applications and detecting malwares. The dynamic analysis can be done after the execution of code either in a virtual environment or on real devices to analyze the actual behavior of the application [8]. Applications may be malicious or benign, after execution, it demonstrates the actual intents. Hence, real-time malware identification using anti-malware detection applications and programs is highly necessary to protect the Android-driven IoT.

As one of the important requirements of Android is to protect IoT devices from malicious applications especially when devices are connected to the vulnerable internet. In AIoT, malicious applications are prevented and detected using antivirus and malware detecting software. The conventional software can only detect and prevent those malwares which are already fed to them [9]. This means that newly created malware could go undetected during malware inspection in case of dynamic analysis. Machine Learning (ML) has been widely used to identify and classify various Android applications, but it is difficult to train and test large datasets efficiently with ML [9]. The Deep Learning (DL) classifiers perform better in multiple feature datasets as they are capable of automatically selecting the features of a dataset and provide more effective data optimization mechanisms [10]. In this study, we proposed hybrid deep learning classifiers to capture the sophisticated malwares by performing the dynamic analysis on AIoT. The classifier is trained on a labeled dataset with multiple features representing Application Programming Interface (API) calls and network traffic. The method is put to carry out evaluations on a large dataset including both benign and malicious traffics. From these applications, several sets of features are extracted to efficiently identify malicious behavior and multiclass malwares. This study makes the following contributions to the existing body of knowledge.

- To effectively identify the sophisticated malwares in AIoT, it designed the four DL-based models. The purpose is to find the best model for comparison with the existing state-of-the-art schemes for malware detection.
- The performance of these models is compared based on the accuracy, precision, recall, F1-Score, True Positive Rate (TPR), True Negative Rate (TNR), Matthews Correlation Coefficient (MCC), and K-fold cross-validation performed to ensure authentic, unbiased and reliable detections.
- The results show that the hybrid model in which three different deep learning models are integrated together outperforms the other models. This proposed hybrid model is further compared with the state-of-the-art benchmark detection schemes. The results confirm that the proposed hybrid model outperforms existing schemes and is capable of identifying multiclass malwares more effectively in AIoT.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 presents the architectural description of utilized DL techniques. Section 4 discusses the proposed system model. Section 5 provides the details of the experimental setup and Section 6 discusses the results. Finally, the conclusions of this research come in Section 7.

2 Related Work

Recently, researchers have developed many malware detection algorithms and frameworks to safeguard AIoT from advanced and sophisticated malwares. Artificial Intelligence (AI), machine learning, and deep learning-based schemes have been proposed in the literature to improve the malwares detection process in (Android Operating System) AOS and AIoT. A study [11] proposes a malware classification framework consisting of Convolutional Neural Networks (CNNs) and Gated recurrent units (GRUs). This study performs the detection of complicated malware by analyzing the certificate and package data. In this work, for experimentation, around two million samples are used which are collected from VirusTotal [12] and achieved 97.7% detection accuracy. This work utilizes the certificate data for the validation and identification of SMSmalware [13]. However, due to the complex certificate and API validation process, the proposed scheme requires high computational power and memory for detection. The study [14] constructed a GRU-based model to identify the malwares by combining Long Term Short Memory (LSTM), Deep Neural Networks (DNNs) and CNNs. This work utilizes the two datasets such as AMD [15] and Anrozoo [16] to classify the trojans, backdoors and benign. This study claims to achieve a detection accuracy of 98.9% but the speed efficiency of the proposed scheme is not promising and requires reducing the time complexity. The study [17] proposes deep learning-based ConvLSTM2D and CNN to detect multiclass attacks and threats in Android-driven devices. This study achieved accuracy up to 99.2% but was not considered to detect the embedded or bound malware including file infector, riskware, rootkits and adware. The model proposed in [18] categorizes the detection module into two layers and achieves an accuracy of 98.29%. However, the main limitation of this work is the high computational power required to process the dual layers of data.

The study [19] performs android multiclass malware analysis using an Artificial Neural Network (ANN) classifier and reached the detection accuracy up to 80.3%. The proposed work utilizes the network traffic and API system calls to feed the proposed classifier for the identification of different malwares such as spyware, adware, SMSmalware and ransomware. The proposed framework proves infeasible to effectively optimize and validate the large dataset as it employed a single classifier for identification which is not an optimal solution to classify the sophisticated malwares [20]. The study [21] uses AI architecture for malware detection. This scheme established a structured Heterogeneous Information Network (HIN) within an Android application. This work linked the system API calls as nodes and their rich relationships as links. Then uses the meta paths for the detection of malware including the looter, ghost push, RevMob, and achieved accuracy up to 98.6%. The result in this study fluctuates when evaluations are performed on large datasets. The study [22] proposes the implementation of dynamic analysis using Bi-directional Long Term Short Memory (BLSTM) neural networks and achieved 97.22% detection accuracy. The proposed model is validated using the AMD dataset with multiclass malware analysis including DroidKungFu, FakeInst, Downing and BankBot. This scheme requires a high computation time due to the conversion of API call data into n-dimensional space. In the study [23] a novel malware identification model based on the obfuscated and non-obfuscated applications is proposed which employed the Discriminative Adversarial Network (DAN) framework. This study uses the Drebin dataset for experimentation and results in 97% accuracy. The study [24] implemented the Deep Belief Network (DBN) and GRU classifiers to detect the malwares. The data is compiled from VirusShare [25] and PRAGuard [26]. This data is further classified into two groups in such a way that the data collected from PRAGuard samples is named as obfuscated and VirusShare as non-obfuscated data. This work achieved only an accuracy of 96.82%.

The study in [27] uses CNN classifier for the malware identification process. The CNN identifies patterns in malware API call graphs, based on the data which is fetched from the Google Play Store and AMD dataset. This scheme computes the resemblance between the application's API call graphs and the high-weight API call graphs of malwares as a result achieved the detection accuracy of 93.2%. This study also lacks to identify the obfuscated malware. The study [28] uses the Google Play Store based data to detect the malwares using CNN. This study creates the three-channel visual color graphics for Android datasets which comprise benign and malicious applications. This model achieved high accuracy on color images-based dataset but the processes such as opcodes and bytecode conversions take longer time in this scheme which makes this scheme impractical. The work proposed in [29] uses the GRU classifier to develop and validate the model using the CICAndMal2017 dataset [30]. This work achieves a malware detection accuracy of 98.2% and becomes ineffective to detect sophisticated malwares because of the simple model. The study [31] uses the CNN and LSTM classifier for opcode feature extraction from the AMD dataset. The model achieved 95.84% accuracy by converting the dataset into an opcode and selecting key opcode features for the evaluation of malware. However, opcode feature conversion and optimization consume computational power and time, which makes this scheme less effective. The scheme in [32] uses a virtual environment to utilize the behavioral features for the detection of malwares. This work uses the data of 13533 applications from the banking, gambling, and utility sectors as an input to the proposed scheme and achieved an accuracy of 98.08%. Apart from low accuracy, this scheme lacks the detection of obfuscated malwares during real-time processing.

3 Preliminaries

This section discusses the deep learning models which are employed in the proposed work to effectively detect sophisticated malwares in AIoT.

3.1 Gated Recurrent Unit (GRU)

GRU is an advanced development of a standard Recurrent Neural Network (RNN). The GRU is simpler than LSTM because it operates on only two gates. The GRU has an internal contrivance known as gates. The gates regulate the flow of bits or information and decide which information to store or remove from the cell. The stored information is further processed to generate predictions. The GRU uses two main vectors such as the updates gate and the reset gate for deciding which flow of information should be progressed to output. As seen in Eq. (1), the update gate Z_t is derived from the outcomes of input and forget gates [14]. The update gate maintains the aggregate of antecedent memory and new information to be clutch.

$$Z_t = \sigma \left(w_z * [h_{(t-1)},\ x_t]\right) \tag{1}$$

In Eq. (1), the σ represents the sigmoid function that selects the value passing to the output gate, the w_z is the weight matrix for the update gate, the x_t is a contemporary input vector and the value $h_{(t-1)}$ deliberated from the antecedent adjoint layer. In Eq. (2), the R_t is the reset gate that manages the network's short-term memory. At the reset gate, the GRU cell merges contemporary input with previous memory and is used to determine how accurately the equation combines the previous state and new output.

$$R_t = \sigma \left(w_z * [h_{(t-1)},\ x_t]\right) \tag{2}$$

In Eq. (3), the h_t presents the current output, tanh is the hyperbolic tangent function that is used in this model to overcome the vanishing gradient problem in the deep learning classifiers as depicted in Fig. 1. The range of tanh output is between -1 and 1. In Eq. (4), o_t represents the final memory in the current stage of the process and also depends on Z_t which is computed in Eq. (1).

$$h_t = \tanh \left(r_t * [h_{(t-1)},\ x_t]\right) \tag{3}$$

$$o_t = (1 - z_t) * h_{(t-1)} + z_t * h_t \tag{4}$$

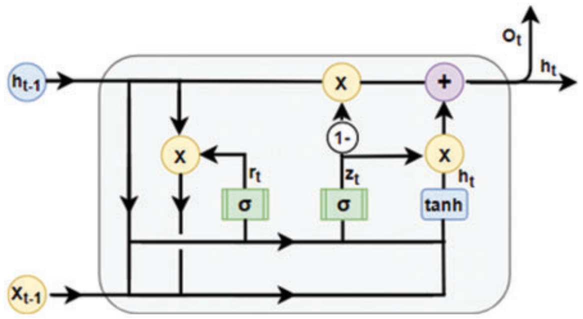

Figure 1: Internal design of GRU [14]

3.2 Bidirectional Long Short-Term Memory (BLSTM)

The BLSTM is derived from the LSTM. The BLSTM classifier circulates the inputs in two directions. Initially, it directs input from backward to forward and then operates it from forward to backward. The BLSTM is used for operations that need a sequence-to-sequence approach. This classifier is used to control the memory cells via three different vectors or gates which compromises of update gate f_t, forget gate i_t, and output gate o_t vectors as represented in Eqs. (5) to (7) respectively.

As shown in Eqs. (5) to (7) that these vectors can be calculated using the weights (such as W_f, W_i and W_o), recurrent connections of previous short-term memory (such as U_f, U_i and U_o) and current inputs of the network (such as b_f, b_i and b_o) [17].

$$f_t = \sigma \left(W_f\, x_t + U_f\, h_{(t-1)} + b_f\right) \tag{5}$$

$$i_t = \sigma \left(W_i\, x_t + U_i\, h_{(t-1)} + b_i\right) \tag{6}$$

$$o_t = \sigma \left(W_o\, x_t + U_o\, h_{(t-1)} + b_o\right) \tag{7}$$

The multi-directional flow of information makes the BLSTM different from the LSTM. In Eq. (8), the layers of BLSTM generate the output vector at time Y< t>, uses forward activation vector af< t>, backward activation vector ab< t>, and assigned weight at the output gate W_y. The final outcome is denoted as Y_T as shown in Eq. (9). Where Y_{T-n} and Y_{T-1} represent the previous output of the backward sequence and $\{^{\rightarrow}h^{\leftarrow}$ represents the sequence of the activation function.

$$y_{\langle t\rangle} = \sigma (W_y[af_{<t>}, ab_{<t>}] + b_y\{^{\rightarrow}h^{\leftarrow}) \tag{8}$$

$$Y_T = [Y_{T-n)}, \ldots,\ Y_{T-1)}] \tag{9}$$

Fig. 2 represents the internal architecture of the BLSTM, where tanh is a hyperbolic tangent, h_t is derived from the function, l_t and f_t are derived from sigmoid functions. The X represents the output sequence at the current state, $x_{(t-1)}$ shows the previous sequence in the gate, and the value h(t−1) deliberated from the antecedent adjoint layer. Furthermore, y< t>, c< t> and a < t−1> represents the outputs in forward direction respectively.

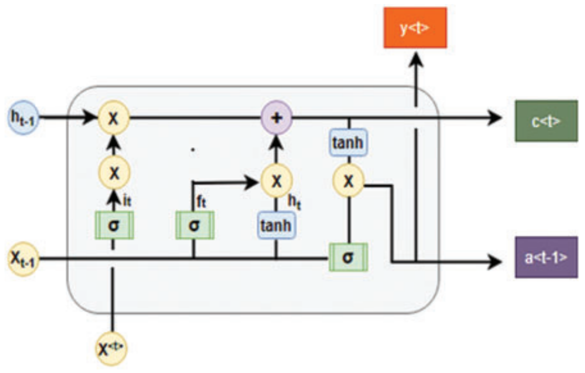

Figure 2: Internal design of BLSTM

4 Proposed Methodology

The proposed concept is illustrated in Fig. 3 which comprises a Graphical Processing Unit (GPU) empowered DNN-BLSTM-GRU hybrid DL model to identify the sophisticated malwares in Android-driven IoT devices. In the data retrieving phase, a single dataset file is compiled by merging multiple CSV files. The generated dataset from the CSV file is fed to the next unit where feature selection, data normalization, data transformation, and label encoding are performed to convert the dataset into a concise format for the classifier. The proposed hybrid DL-driven GPU-accelerated model comprises DNN, BLSTM and GRU which perform the malware detection by labeling them as benign or malware. CUDA is a parallel computing platform that utilizes GPU capabilities to process the computation task very fast. In this work, GPU-accelerated CUDA with a deep neural network library named as cuDNN is employed for developing deep neural networks. As a result, the cuDNN empowered hybrid classifier is capable of effectively and speedily identifying benign and multiple malwares such as SMSMalware, adware, scareware, and ransomware. Finally, performance assessment parameters including accuracy, F1-score, recall, precision, and AUC-ROC (Area Under the Curve-Receiver Operating Characteristics) are generated.

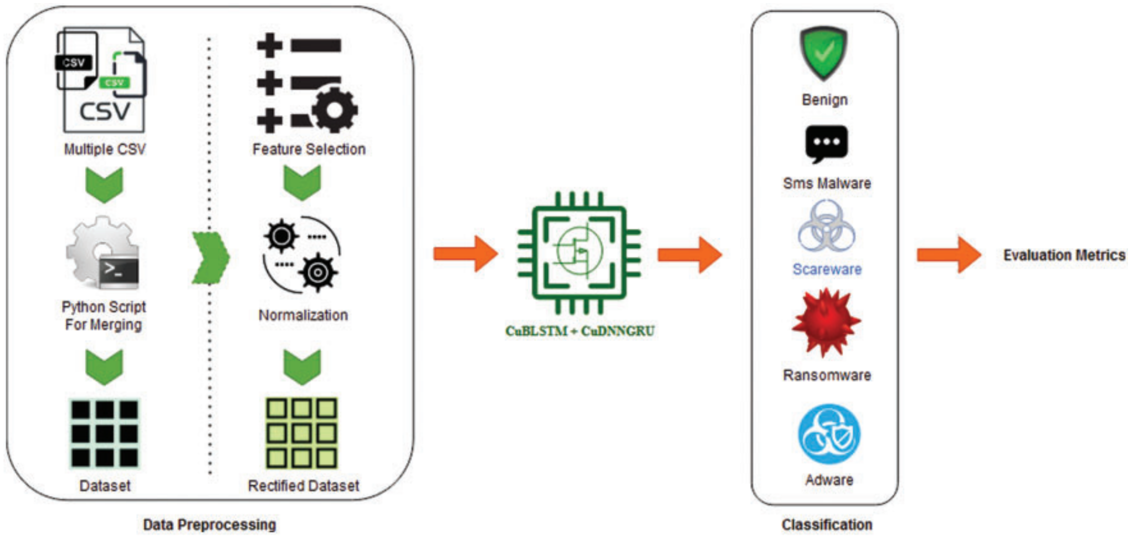

Figure 3: Architecture of the proposed DL-driven hybrid model

4.1 Data Retrieving

The dataset CICAndMal2017 [30] comprises 426 malicious and 1,700 benign applications gathered by the researchers of the University of New Brunswick (UNB) has been utilized. The dataset has been preprocessed to improve the efficacy and performance of our proposed hybrid deep learning approach.

4.2 System Model

In this novel architecture, three different deep learning models are integrated together to form a hybrid classifier for optimal training and prediction. The hybrid classifier learns the feature with an optimized approach. The hybrid models are the next generation of artificial intelligence that combine two or more classifiers from the same or other families to outperform most standard AI classifiers. The GPU-powered, DL-driven DNN-BLSTM-GRU is used for real-time detection of constantly evolving modern cyber threats in Android-based IoT devices. The framework implements GRU and BLSTM to maximize the benefits of advanced deep learning classifiers at the same time. The well-refined data is passed to a hybrid classifier for training and evaluation. The main system model including neurons, activation functions, optimizers, batch size, epochs, and layers is shown in Tab. 1. The proposed framework is designed for the multiclass classification problem and to prevent vanishing gradient problems in it, this study utilizes the ReLU activation function for the hidden layers comprised of CuDNN-BLSTM and CuDNN-GRU. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time and provides the efficient convergence time for the model as required in the proposed model. The Softmax function is used to smooths the output without any data loss and implemented at the last output layer which is the

Dense layer in our scenario. The AdaMax optimizer is used for the optimization of proposed models. The AdaMax systematically adjusts the step size and learning rate for individual parameters during the optimization process and is suitable for the larger dataset, as required by the proposed scheme. Finally, the results are extensively analyzed using established performance criteria for the parameters such as accuracy, precision, recall, F1-score, and AUC-ROC.

Table 1: Models of CUDA-enabled hybrid DL-driven algorithm

Sr. no	Layers	Neurons	Activation Function	Optimizer	Batch size	Epochs	Folds
1	CuDNN-BLSTM	300	ReLu	AdaMax	64	5	10
2	CuDNN-GRU	80, 100, 300	ReLu	AdaMax	64	5	10
3	Dense	5, 50, 70, 80	Softmax	AdaMax	64	5	10

4.3 Dataset Distribution

The dataset contains a total of 120 K records, 100 K records represent the benign family and 20 K records belong to the malware family. The malware family is further divided into multiple subfamilies. Of the 100% of data, 84% contains benign, 4% is ransomware, 4% is scareware, 4% is SMSMalware, and the ratio of adware is 4%. The higher the benign records, the higher the efficacy of the model. The dataset is sliced into different phases including the testing phase and training phase. The model is trained with 90% of the data and the remaining 10% data is used for model evaluation purposes. The training and testing data are split using the sklearn Python library which separates data based on the user's input. In the proposed model, the unnecessary data fields including "nan" values are initially removed from the dataset to improve the model training and optimize the testing without model overfitting problem. As reported in [33], that training the model with more training data improves the prediction accuracy. Hence, the proposed model uses 90% of data for the training which generates the optimal results without overfitting. To further reduce the problem, the K-fold cross-validation is utilized. The dataset is accumulated by collecting the dynamic features in three stages such as at the time of installation, before restarting the device, and after restarting. The multiclass dataset is comprised of data that is compiled after real time execution of Android applications.

5 Experimental Setup

This section discusses the detailed overview of the experimental setup, conventional performance evaluation metrics, and results accompanying the discussion.

5.1 Experimental Setup

The computation system used for the experimentation employed the seven-core Intel processor, 6G GPU, 16 GB RAM, and Windows 10 as an operating system. Experimentation of this framework has been performed using Google's TensorFlow library [34] and Python library Keras [35]. The supervised scheme of machine learning is used for feature learning and prediction process.

5.2 Evaluation Metrics

To evaluate the performance of the DL frameworks, standard evaluation parameters such as accuracy, precision, recall, and F1-score are used. A confusion matrix is employed which uses the True Positive-TP, True Negatives-TN, False Positive-FP, and False Negative-FN to predict the performance of the model. The TP is an outcome in which a system correctly predicts the positive class. The TN is an outcome in which the system correctly predicts the negative class. The FP is an outcome in which the system incorrectly predicts the positive class. The FN is an outcome in which the system incorrectly predicts the negative class. The accuracy is defined as the percentage of correctly labeled records and also represents the detection rate of the utilized algorithm based on Eq. (10). The positive predictive value is referred to as precision and it identifies the number of correctly predicted records based on Eq. (11). The true positive rate is referred to as recall and it presents the correctly predicted records from total records, based on Eq. (12). The F1-score presents a harmonic mean between recall and precision which can be calculated from Eq. (13). The ROC curve shows the plotting of the TP rate and FP rate. AUC-ROC shows how much the model is capable of distinguishing the classes. A higher value of AUC-ROC represents a model having better performance in predicting positive classes as positive and negative classes as negative.

$$\text{Accuracy} = (\text{TP} + \text{TN}) \,/\, (\text{TP} + \text{TN} + \text{FP} + \text{FN}) \tag{10}$$

$$\text{Precision} = \text{TP}/\,(\text{TP} + \text{FP}) \tag{11}$$

$$\text{Recall} = \text{TP}/\,(\text{TP} + \text{FN}) \tag{12}$$

$$\text{F1} - \text{score} = (2 * \text{TP})/(2 * \text{TP} + \text{FP} + \text{FN}) \tag{13}$$

6 Results and Discussion

In the proposed model, the malware and benign classifications are based on the features extracted from API calls and permissions. The metrics such as TPR, TNR, and MCC are used for performance evaluation. Where MCC is an efficient technique to assess the validity of binary and multiclass classifications. The well-refined dataset is fed into the DL-driven four models to perform detections as depicted in Fig. 4. The CuDNN-enabled models comprise LSTM, GRU, BLSTM and a hybrid BLSTM-GRU. The cuDNN-LSTM classifier, attain 99.68% TNR, 99.88% TPR and 99.49% MCC. This is because the LSTM has three operating gates that are input, output, and forget. However, it operates in a single direction, without propagating information in the backward direction. The three operating gates of LSTM make it smooth to process large datasets and achieve optimal accuracy. The CuDNN-GRU touches the lowest at 98.88% TNR, 99.83% TPR and 98.91% MCC. This is because the GRU operates on two gates in the forward direction which swiftly process the dataset. However, GRU is not the best option to process large datasets. The CuDNN-BLSTM achieves 99.62% TNR, 99.84% TPR and 99.37% MCC. The BLSTM adds another LSTM layer that reverses information flow. There are three gates input, output and hidden cell state which are operating in two directions. The BLSTM propagates data in the forward direction and backward direction, beneficial for sequential data. Therefore, BLSTM memorizes data for a long time, resulting in improved prediction. The proposed DL-driven CuDNN-BLSTM-GRU hybrid classifier achieved 99.74%, 99.90%, and 99.57% for TNR, TPR, and MCC respectively. The hybrid classifier utilizes the processing of two different algorithms for balanced performance, BLSTM memorizes the data for the long-term and GRU fasten the process. There MCC is much improved in this classifier and confirms the high quality of multiclass prediction.

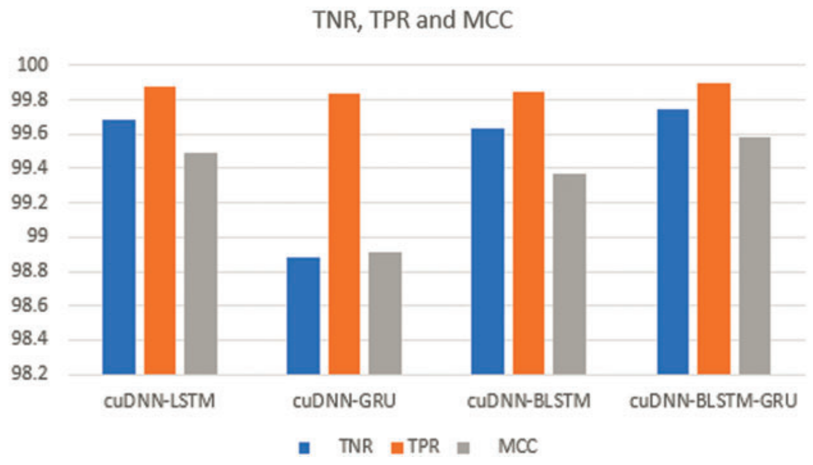

Figure 4: TNR, TPR and MCC values for different models

The cuDNN-LSTM touches 99.84%, 99.88%, 99.88%, and 99.93% of accuracy, F1-score, recall and precision respectively as depicted in Fig. 5. The LSTM layer memorizes the long-term memory, consumes high memory and predicts accurate class. Hence, multiple gates, and detailed processes can make an optimal prediction. The CuDNN-GRU achieves 99.65% accuracy, 99.81% F1-score, 99.83% recall, and 99.77% precision. The GRU has two gates for data processing which updates the value faster and accordingly generates the output. The GRU process the data fast as compared to BLSTM but cannot memorize it for a long time. This is the reason that for the larger datasets alone GRU is not an optimal classifier. The CuDNN-BLSTM markup 99.82% accuracy, 99.84% F1-Score, 99.83% recall and 99.92% precision. The proposed hybrid cuDNN-BLSTM-GRU algorithm surpasses the other deep-learning classifiers. In deep learning, neuron weights are updated based on the batch size. Meanwhile, batch-by-batch, training data is made available to the network as an epoch. Furthermore, the AdamMax optimizer function is used to obtain improved precision. As a result, the total loss is reduced, and the precision is improved. This is the reason that the proposed hybrid DL-driven model achieved a 99.94% precision level, 99.9% recall, 99.87% detection accuracy and 99.9% F1 score.

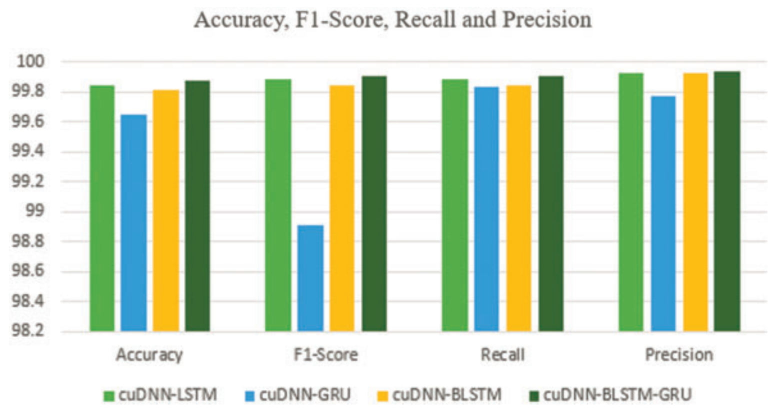

Figure 5: Evaluation parameters for proposed and other DL-driven techniques

A dataset with uneven numbers of observations in single or multiple classes might lead to incorrect classification accuracy. Therefore, calculating a confusion matrix clarifies where the classification model predicts better or makes errors, as shown in Fig. 6. In a multiclass classification problem, an effectual activation function is required to classify each class accurately. Therefore, SoftMax is used to activate class membership on multiclass labels. Furthermore, the true positive rate is improved in the proposed scheme as compared to other models due to detailed data fetching and the convergence of bidirectional BSTM and GRU. The balanced distribution of the dataset makes it simple to depict the stable results in the confusion matrix. The confusion matrix of the proposed framework shows the minimum FP rate of each class.

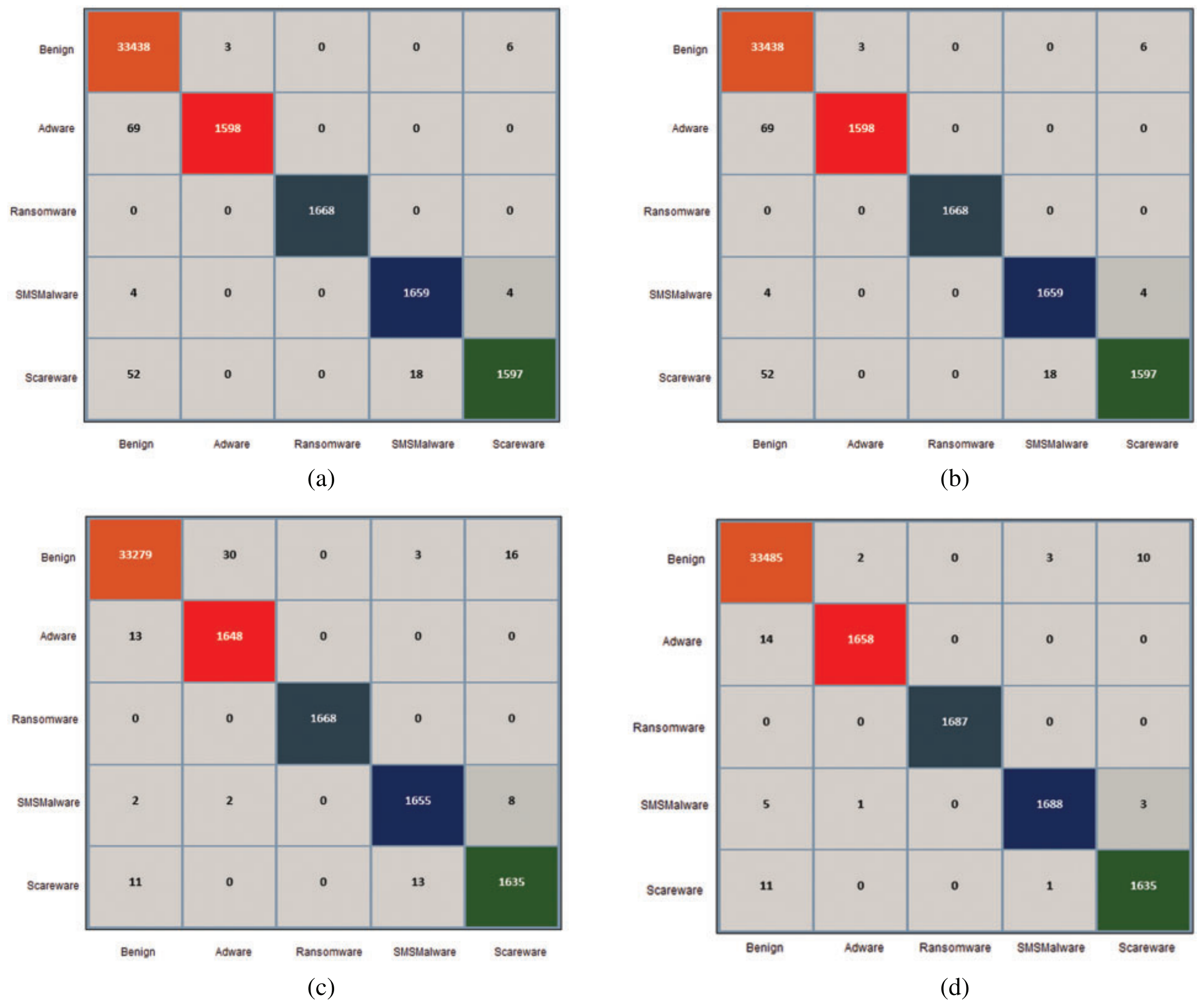

Figure 6: Confusion matrices: (a) CuDNN-LSTM, (b) CuDNN-BLSTM, (c) CuDNN-GRU and (d) CuDNN-BLSTM-GRU

To construct an unbiased model, k-fold cross-validation is introduced in the framework as depicted in Fig. 7. The method, split the dataset into k iterations to achieve unbiased results. Hence, to demonstrate explicitly unbiased results, the 10-fold cross-validation method is used where k = 10.

Furthermore, the data is distributed equally for each fold iteration, 84% for training and 16% for testing. The layered architecture including the model's epoch and batch size remains consistent across folds. The fold eight, achieves high accuracy, on the other hand, at fold five the accuracy is the lowest among all folds. In the end, a collective accuracy of cross-validation is achieved by the hybrid model, showing unbiased and reliable evaluation results.

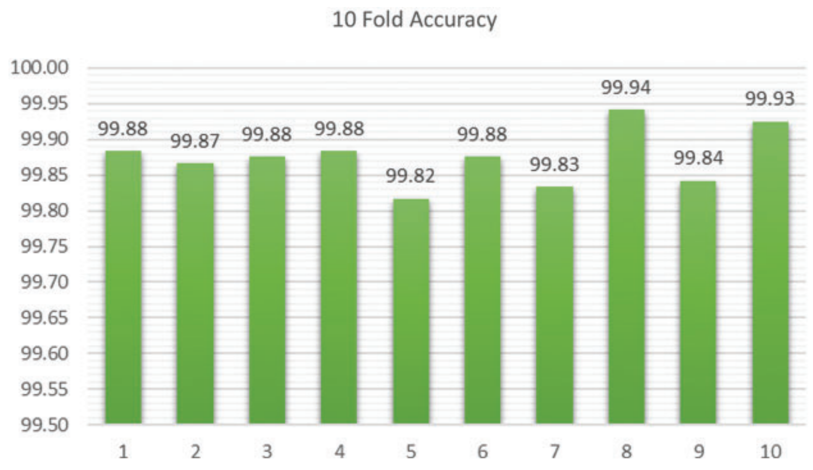

Figure 7: 10-Fold validation: accuracy of each fold

A model constructed using the CuDNN library running on NVIDIA GPU has a faster training time than a model produced with the conventional TensorFlow kernel. As depicted in Fig. 8, the cuDNN-LSTM classifier takes 1390 milliseconds for testing. The CuDNN-GRU classifier takes 1177 milliseconds for malware detection. This is because the GRU has two gates update and forgets, providing a fast convergence rate. However, it takes 1370 for CuDNN-BLSTM to classify the malware and benign. The BLSTM contains three gates and operates in both directions such as, forward and backward. The bidirectional flow of information slows down the process as compared to GRU. The CuDNN-BLSTM-GRU take1380 milliseconds for testing. In this framework, CUDA compatible code is used with a batch size of 64 and an AdaMax optimizer for fast convergence which ultimately achieves optimal training and testing time. There is a compromise between achieved accuracy and testing time realized with the proposed DL-driven model. The speed proficiency of the proposed scheme is not optimal which can be improved in future work.

The performance evaluation is critical in DL and in dealing with a classification issue, it relies on the AUC-ROC curve. As shown in Fig. 9, the AUC-ROC graph is drawn between TPR and FPR while the area under the curve presents the progressive results of the proposed technique. When AUC = 0.5, the classifier cannot identify Positive from Negative class points. The classifier predicts a random or continuous categorization for all data points. At the value of AUC = 1, the classifier properly distinguishes all positive and negative class points. The ROC curve of the proposed approach shows that every class of dataset achieves an AUC of 1, effectively distinguishing between the benign and malware classes.

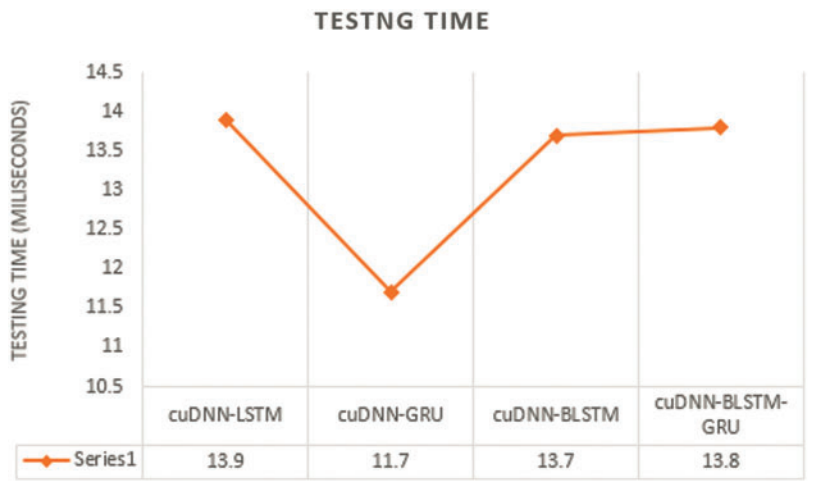

Figure 8: Testing time of four DL models

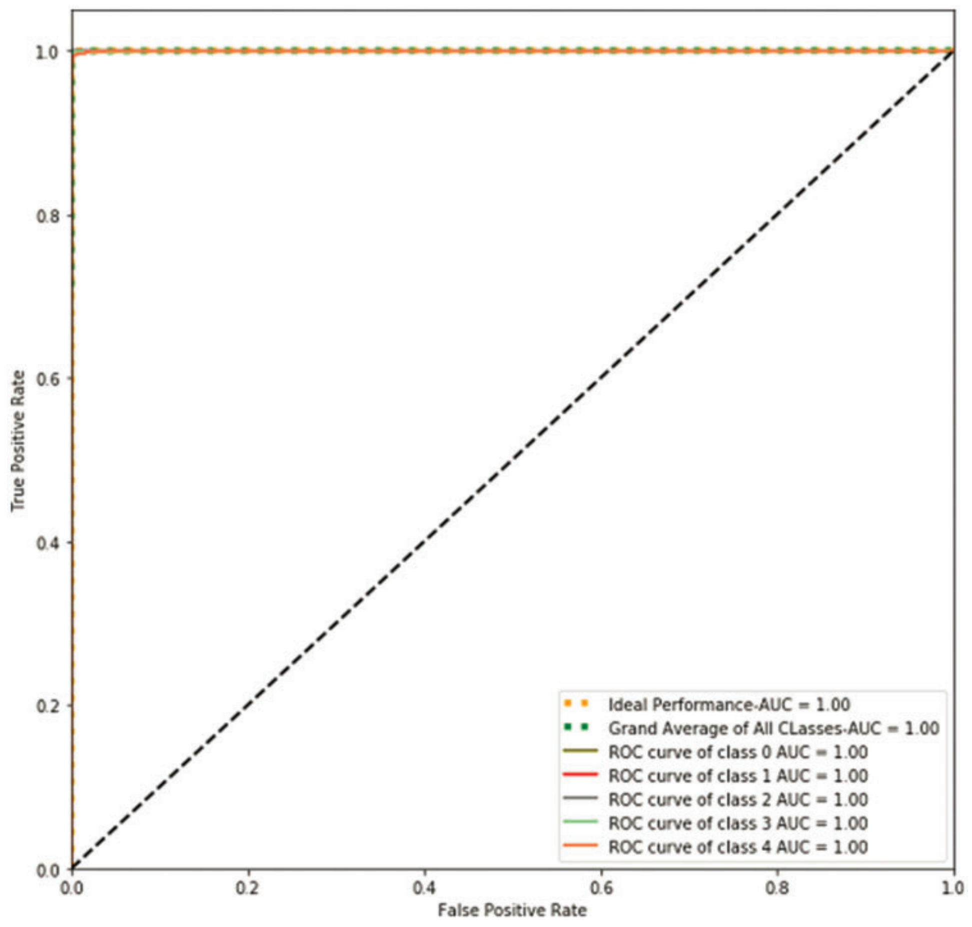

Figure 9: AUC-ROC curve for the proposed model

The newly constructed hybrid classifier is also compared with benchmark schemes as shown in Tab. 2. The experimental results confirm that the proposed classifier results the higher accuracy, F1 score, recall and precious as compared to other frameworks. The results also demonstrate that the proposed hybrid model outperforms other frameworks in terms of reliability. The framework [18] employed the Variational Auto Encoder (VAE) based LSTM model for selecting the feature, performing the training, and estimating the results. This scheme is effective only for sequential data and requires high computation power for processing. In our scheme, although the BLSTM requires memory, but a smaller number of iterations are used to balance the memory consumption and performance. The study [19] uses ANNs for

Table 2: Comparison with existing state-of-the-art frameworks

Ref.	Schemes	Accuracy	Precision	Recall	F1 Score
[18]	LSTM-VAE	97.29	94.63	96.07	95.34
[19]	DeepAMD	80.3	82.2	80.3	80.5
[22]	Droidetec	97.22	98.21	98.38	98.2
Proposed	CuDNN-BLSTM-GRU	99.87	99.94	99.9	99.9

detecting multiclass malware as the ANNs are very resilient to noise during training. However, due to the nature of ANNs, they need processors with parallel processing capacity. The ANNs may originate less reliable results in the neural network, which causes false-positive results. The framework [22] utilize the BLSTM classifier for the dynamic detection of sophisticated malwares. This study employed to weight distribution strategy to fetch the semantic vectors from the API calls. The BLSTM operates in dual sequence as it operates in forwarding and backward propagation which causes high memory consumption.

The proposed hybrid solution cuDNN-BLSTM-GRU outperforms the benchmark frameworks. The BLSTM learns the previous and next patterns in the neural network and takes less time in traversing data in both backward and forward directions. The GRU operates in the forward direction and learns the pattern with a fast-learning rate. The cuDNN speed up the training process, testing process, and validation process. The combination of BLSTM and GRU with cuDNN proved optimal to learn and predict the malware patterns in AIoT as verified through experiments. The cross-fold validation further improved the results and ensure the unbiased, reliable, and accurate detection of sophisticated malwares.

The proposed model performed real-time malware identification by analyzing multiple network traffic features. The proposed model detects sophisticated malwares with high Accuracy, F1-Score, recall, precision, and as compared to existing frameworks. The framework implemented the 10-fold validation which shows authentic and unbiased results. The proposed model has a few limitations such as it results in high testing time due to its complexity. The model is trained using a single dataset to classify the multiple traffic types which can be further improved by training the model on multiple datasets. The model is not designed to classify malicious traffic that is in encrypted form. It requires both API calls and network traffic to effectively identify the sophisticated malwares.

7 Conclusion

The rising popularity and demand of Android-based IoT devices make them vulnerable to emerging and complex security attacks. The detection and classification of multiclass sophisticated malwares in AIoT devices are indispensable. This paper presented a novel framework in which three different deep learning models are integrated to form a hybrid classifier for optimal training and prediction. The proposed model is compared with the existing benchmark detection schemes. The results confirm that the proposed model reaches higher detection accuracy for sophisticated malwares in AIoT compared to the existing detection models with the trade-off in computation time. In future work, we intend to propose a solution to reduce the complexity, computation time of the proposed model and train the model on multiple datasets to improve its performance.

Funding Statement: This work was supported by the MSIP and National Research Foundation of South Korea under Grant 2018R1D1A1B07049877.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar *et al.,* "A multimodal malware detection technique for android IoT devices using various features," *IEEE Access*, vol. 7, no. 1, pp. 64411–64430, 2019.

[2] J. T. McDonald, N. Herron, W. B. Glisson and R. K. Benton, "Machine learning-based android malware detection using manifest permission," in *Proc. ICSS*, Hawai, USA, pp. 6976–6985, 2021.

[3] G. Stergiopoulos, D. Gritzalis, E. Vasilellis and A. Anagnostopoulou, "Dropping malware through sound injection: A comparative analysis on android operating systems," *Computers & Security*, vol. 105, no. 11, pp. 34–58, 2021.

[4] F. A. Alharbi, A. M. Alghamdi and A. S. Alghamdi, "A systematic review of android malware detection techniques," *International Journal of Computer Science and Security*, vol. 15, no. 1, pp. 01–19, 2021.

[5] A. Gaurav, B. B. Gupta and P. K. Panigrahi, "A comprehensive survey on machine learning approaches for malware detection in IoT-based enterprise information system," *Enterprise Information Systems*, vol. 16, no. 6, pp. 1–25, 2022.

[6] D. O. Sahın, S. Akleylek and E. Kilic, "Linregdroid: Detection of android malware using multiple linear regression models-based classifiers," *IEEE Access*, vol. 10, no. 1, pp. 14246–14259, 2022.

[7] Z. Ren, H. Wu, Q. Ning, I. Hussain and B. Chen, "End-to-end malware detection for android IoT devices using deep learning," *Ad Hoc Networks*, vol. 101, no. 1, pp. 102098–102111, 2020.

[8] A. Razgallah, R. Khoury, S. Halee and K. Khanmohammadi, "A survey of malware detection in android apps: Recommendations and perspectives for future research," *Computer Science Review*, vol. 39, no. 3, pp. 100358–100375, 2021.

[9] S. Millar, N. McLaughlin, J. M. Rincon and P. Miller, "Multi-view deep learning for zero-day android malware detection," *Journal of Information Security and Applications*, vol. 58, no. 1, pp. 102718–102732, 2021.

[10] E. B. Karbab, M. Debbabi, A. Derhab and D. Mouheb, "MalDozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, no. 5, pp. 48–59, 2018.

[11] W. Y. Lee, J. Saxe and R. Harang, "Seqdroid: Obfuscated android malware detection using stacked convolutional and recurrent neural networks," *Deep Learning Applications for Cyber Security*, vol. 1, no. 1, pp. 197–210, 2019.

[12] P. Peng, L. Yang, L. Song and G. Wang, "Opening the blackbox of VirusTotal: Analyzing online phishing scan engines," in *Proc. IMC*, Amsterdam, Netherlands, pp. 478–485, 2019.

[13] K. Hamandi, A. Chehab, I. H. Elhajj and A. Kayssi, "Android SMSmalware: Vulnerability and mitigation," in *Proc. AINAW*, Barcelona, Spain, pp. 1004–1009, 2013.
[14] I. Bibi, A. Akhunzada, J. Malik, J. Iqbal, A. Musaddiq *et al.,* "A dynamic DL-driven architecture to combat sophisticated android malware," *IEEE Access*, vol. 8, no. 1, pp. 129600–129612, 2020.
[15] F. Wei, Y. Li, S. Roy, X. Ou and W. Zhou, "Deep ground truth analysis of current android malware," in *Proc. DIVMA*, Saclay, France, pp. 252–276, 2017.
[16] K. Allix, T. F. Bissyande, J. Klein and Y. L. Traon, "Androzoo: Collecting millions of android apps for the research community," *IEEE/ACM MSR*, Austin, *Texas*, pp. 468–471, 2016.
[17] I. U. Haq, T. A. Khan, A. Akhunzada and X. Liu, "MalDroid: Secure DL-enabled intelligent malware detection framework," *IET Communications*, vol. 16, no. 1, pp. 1160–1171, 2021.
[18] M., Liu, S. Wei and P. Jiang, "A hybrid modeling of mobile app dynamics on serial causality for malware detection," *Security and Communication Networks*, vol. 21, no. 10, pp. 1–10, 2021.
[19] I. S. Imtiaz, S. Rehman, A. R. Javed, Z. Jalil, X. Liu *et al.,* "DeepAMD: Detection and identification of android malware using high-efficient deep artificial neural network," *Future Generation Computer Systems*, vol. 115, no. 1, pp. 844–856, 2021.
[20] S. Ardabili, A. Mosavi and A. R. Varkonyi-Koczy, "Advances in machine learning modeling reviewing hybrid and ensemble methods," in *Proc. ICGRM*, Gomel, Belarus, pp. 215–227, 2020.
[21] S. Hou, Y. Ye, Y. Song and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proc. SIGKDD*, Halifax, Canada, pp. 1507–1515, 2017.
[22] Z. Ma, H. Ge, Z. Wang, Y. Liu and X. Liu, "Droidetec: Android malware detection and malicious code localization through deep learning," *Cryptography and Security*, vol. 3, no. 1, pp. 3595–3608, 2020.
[23] S. Millar, N. McLaughlin, J. M. Rincon, P. Miller and Z. Zhao, "DANdroid: A multi-view discriminative adversarial network for obfuscated android malware detection," in *Proc. CODASPY*, New Orleans LA USA, pp. 353–364, 2020.
[24] T. Lu, Y. Du, L. Ouyang, Q. Chen and X. Wang, "Android malware detection based on a hybrid deep learning model," *Security and Communication Networks*, vol. 20, no. 6, pp. 1–11, 2020.
[25] H. D. Menendez, S. Bhattacharya, D. Clark and E. T. Barr, "The arms race: Adversarial search defeats entropy used to detect malware," *Expert Systems with Applications*, vol. 118, no. 4, pp. 246–260, 2019.
[26] S. Sen, E. Aydogan and A. I. Aysan, "Coevolution of mobile malware and anti-malware," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2563–2574, 2018.
[27] J. Kim, Y. Ban, E. Ko, H. Cho and H. H. Yi, "MAPAS: A practical deep learning-based android malware detection system," *International Journal of Information Security*, vol. 21, no. 4, pp. 725–738, 2022.
[28] I. Almomani, A. Alkhayer and W. El-Shafai, "An automated vision-based deep learning model for efficient detection of android malware attacks," *IEEE Access*, vol. 10, no. 1, pp. 2700–2720, 2022.
[29] O. N. Elayan and A. M. Mustafa, "Android malware detection using deep learning," *Procedia Computer Science*, vol. 184, no. 2, pp. 847–852, 2021.
[30] A. H. Lashkari, A. F. Kadir, L. Taheri and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *Proc. ICCST*, Montreal, Quebec, Canada, pp. 1–7, 2018.
[31] N. Lu, D. Li, W. Shi, P. Vijayakumar, F. Piccialli *et al.,* "An efficient combined deep neural network-based malware detection framework in 5G environment," *Computer Networks*, vol. 189, no. 6, pp. 107932–107943, 2021.
[32] V. Sihag, M. Vardhan, P. Singh, G. Choudhary and S. Son, "De-lady: Deep learning based android malware detection using dynamic features," *Journal of Internet Services and Information Security*, vol. 11, no. 2, pp. 34–45, 2021.

[33] X. Ying, “An overview of overfitting and its solutions,” *Journal of Physics: Conference Series*, vol. 1168, no. 2, pp. 001–006, 2019.

[34] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis *et al.,* “Tensorflow: A system for large-scale machine learning,” in *Proc. SOSDI*, Savannah, GA, USA, pp. 265–283, 2016.

[35] F. Chollet, Q. S. Zhu, F. Rahman, T. Lee, G. D. Marmiesse *et al.,* “Keras,” *GitHub Repository*, 2015. [Online]. Available: https://github.com/fchollet/keras.