Advanced Android Malware Detection through Deep Learning Optimization

Ahmed Alhussen
Department of Computer Engineering, College of Computer and Information Sciences, Majmaah University, Saudi Arabia
aa.alhussen@mu.edu.sa (corresponding author)

Received: 9 April 2024 | Revised: 18 April 2024 | Accepted: 21 April 2024

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | *DOI: https://doi.org/10.48084/etasr.7443*

ABSTRACT

Android stands out as one of the most prevalent mobile operating systems globally, due to its widespread adoption and open-source nature. However, its susceptibility to malware attacks, facilitated by the ability to install third-party applications without centralized control, poses significant security challenges. Despite efforts to integrate security measures, the proliferation of malicious activities and vulnerabilities emphasizes the need for advanced detection techniques. This study implemented and optimized Long Short-Term Memory (LSTM) and Neural Network (NN) models for malware detection on the Android platform. Leveraging meticulous hyperparameter tuning and robust data preprocessing techniques, this study aimed to increase the efficacy of LSTM and NN models in identifying and mitigating various forms of malware. The results demonstrate remarkable performance, with the LSTM model achieving an accuracy of 99.24%, precision of 99.07%, recall of 98.79%, and F1-score of 98.93%, and the NN model attaining an accuracy of 99.18%, precision of 99.02%, recall of 98.84%, and F1-score of 98.93%. By addressing these challenges and achieving such high levels of accuracy and effectiveness, this study contributes significantly to the ongoing endeavor to fortify defenses against cyber threats, thus fostering a safer digital environment for users worldwide.

Keywords-LSTM; deep learning; hyperparameter tuning; Android malware

I. INTRODUCTION

In today's digital landscape, safeguarding computer systems against cyber threats, particularly in the realm of malware detection, is of paramount importance [1, 2]. The widespread usage of mobile operating systems, such as Android, has heightened the significance of this issue, given the platform's susceptibility to various forms of malware. Despite efforts to improve security measures, the persistent emergence of new malware variants underscores the need for continuous improvement in detection techniques. This necessitates a deeper investigation of more advanced approaches that can effectively identify and mitigate the evolving threats. This study responds to the specific need by focusing on optimizing the performance of LSTM models for malware detection on the Android platform by refining its hyperparameters and implementing proper data preprocessing techniques, aimed at enhancing its ability to detect and counteract various types of malware effectively.

This study used Deep Learning (DL) models for Android malware detection, including Long Short-Term Memory (LSTM) and Neural Networks (NN), instead of traditional Machine Learning (ML) methods. LSTMs, a specialized type of Recurrent Neural Networks (RNNs), were chosen for their ability to handle long-term dependencies and capture complex patterns in malware behavior. These DL models automatically learn hierarchical representations from the data, reducing the need for manual feature engineering and enhancing detection accuracy. The main contribution of this study is that it takes advantage of these advanced DL techniques to ameliorate the accuracy and classification of Android malware signatures, addressing the challenges of detecting complex and evolving malware threats.

II. LITERATURE REVIEW

Table I depicts previous studies with various approaches to malware detection. In [3], extensive datasets from both benign and malicious apps were collected to evaluate DroidCollector, which achieved a high detection rate of 98% through ML algorithms. In [4], a deep NN achieved an impressive accuracy of 93.4% on a dataset containing 23 attacks without hyperparameter tuning. In [5], Android application and network layer features were employed in ML models, resulting in detection rates of 99 and 81%, respectively. In [6], static and dynamic artifacts were utilized to classify Android applications and address the challenge of ransomware, surpassing the existing solutions. Meanwhile, other studies' proposed models and methods ranging from system permission features to ensemble ML approaches, achieving accuracies between 94% and 99% [7-12]. Despite their effectiveness, some approaches, such as the Ensemble Deep Restricted Boltzmann Machine [13], demonstrated accuracy below 80%. In [14], previous limitations were addressed by implementing rigorous data preprocessing and hyperparameter tuning using Gridsearch CV.

TABLE I. LITERATURE REVIEW DETAILS

Reference	Summary
[3]	Collected substantial datasets from both benign and malicious apps (11560). Applied ML algorithms on these datasets and achieved a 98% average detection rate.
[4]	Used a Deep NN, achieving an accuracy of 93.4%. The dataset encompasses 23 attacks. However, a significant limitation was the absence of hyperparameter tuning.
[5]	Leveraged Android application and network layer features in ML models to detect malware applications. The models achieved detection rates of 99% and 81% on open datasets.
[6]	Implemented static and dynamic artifacts from ransomware to classify Android applications as malicious or benign, using an NN for static artifacts and an LGBMClassifier for network traffic, surpassing current solutions.
[7]	Proposed a model utilizing Android system permissions features for malware detection.
[8]	Employed an RNN model, demonstrating exceptional efficiency in detecting potential threats with over 99% accuracy, even with a reduced number of evaluated networking features.
[9]	Introduced DREBIN, a lightweight method for Android malware detection. By conducting extensive static analysis and integrating features into a unified vector space, DREBIN identified malware patterns with 94% accuracy.
[10]	Android malware detection with 96% accuracy using the DREBIN dataset.
[11]	Presented DroidFusion, a fusion classification approach utilizing a multilevel architecture for enhanced accuracy. By training base classifiers and applying ranking-based algorithms, DroidFusion achieved 98.74% accuracy on the DREBIN dataset.
[12]	DCNN was employed for malware detection via binary visualization. The combined approach of DCNN and TFDNN achieved a classification accuracy of approximately 98%.
[13]	Presented an Ensemble Deep Restricted Boltzmann Machine (EDRBM) to classify cybersecurity threats in large-scale networks, particularly focusing on malware. While simulations demonstrated its effectiveness against various malware attacks, its accuracy remained below 80%.
This study	Addressed previous limitations by implementing rigorous data preprocessing and hyperparameter tuning using Gridsearch CV.

III. METHODOLOGY

Figure 1 shows the steps followed.

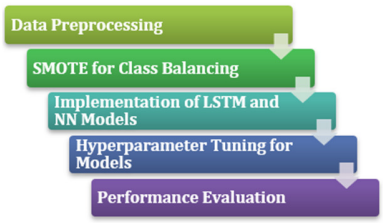

Fig. 1. The overall workflow of this study for malware detection.

A. Data Preprocessing

This investigation used the DREBIN-215 dataset, which is a real Android malware dataset from [9, 11]. The DREBIN-215 dataset is a well-known dataset commonly implemented for Android malware detection research. It comprises Android applications, including both malware and benign applications, collected from various sources. Each sample in the dataset is represented by a feature vector containing a set of static analysis features extracted from the app's APK file. The features in the DREBIN-215 dataset capture various aspects of Android applications, such as permissions requested, API calls made, and other characteristics related to the app's behavior. This rich feature set enables researchers to develop and evaluate machine learning and deep learning models to detect malicious Android apps effectively. A collection of feature vectors comprising 215 attributes sourced from 15,036 applications was assembled. This compilation includes 5,560 instances of malware applications from the Drebin project and 9,476 benign applications. The dataset consisted of 15,036 rows and 216 columns. Each row represented an individual application and each column represented a specific feature of the application. The word cloud visualization technique was followed to get a visually appealing representation of the textual data, as observed in Figure 2, where the size of each word corresponds to its frequency in the corpus.

Fig. 2. The word cloud of the DREBIN dataset used in this study.

Proceeding with the data preprocessing step, the first task involved handling numerical features with missing values. To identify such features, a list called numerical_nan was created, comprising those numerical columns where the count of NaN values exceeded one and the data type was not categorical. Subsequently, the categorical features with missing values were addressed. The dataset was transformed using one-hot encoding to handle the categorical variables. Following this transformation, the categorical features with missing values were identified and stored in the list categorical_nan, where each feature had a nonzero count of NaN values and a data type of 'object'. An attempt was made to understand the relationships between the features of the datasets using a correlation heatmap, as spotted in Figure 3.

B. Class Balancing using SMOTE

Class balancing was employed to address the challenges posed by an imbalanced dataset. Not performing class balancing led to a biased model that favored the majority class, resulting in reduced sensitivity and poor generalization to the minority class. This imbalance also skewed performance metrics, leading to misleading accuracy scores and an increased risk of false negatives. Implementing class balancing techniques aims to improve model robustness, sensitivity, and overall performance in detecting both majority and minority

class instances effectively. This study adopted the Synthetic Minority Over-sampling Technique (SMOTE) to address the class imbalance issue. SMOTE was chosen over other methods due to its effectiveness in generating synthetic samples by interpolating between existing minority class instances, thereby increasing the representation of the minority class without duplicating the existing samples. This approach helps to retain the original distribution of the minority class and reduces the risk of overfitting. The Drebin 215 dataset originally contained 5,560 instances of malware and 9,476 instances of benign applications. Class imbalance poses a challenge for ML algorithms, as they may become biased towards the majority class, leading to suboptimal performance in detecting the minority class. SMOTE was applied to augment minority-class samples by generating synthetic instances that are similar to the existing minority-class samples [14]. This process effectively balanced the class distribution, resulting in an equal number of instances for both malware and benign applications after SMOTE, as shown in Figure 4. The balanced dataset enables more accurate and reliable training of ML models for malware detection, contributing to improved overall performance and effectiveness in cybersecurity applications.

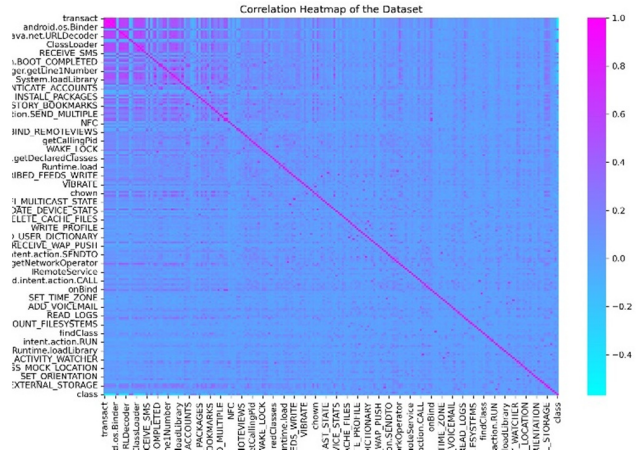

Fig. 3. Heatmap of the dataset features.

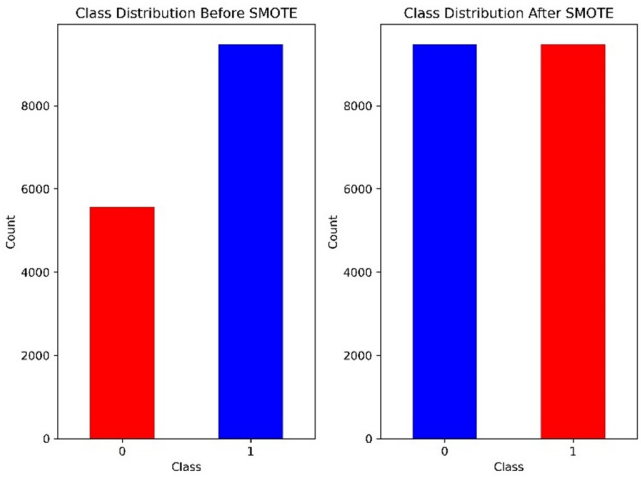

Fig. 4. Class distribution before and after implementing smote.

C. Implementation of the LSTM and NN models

The LSTM model was applied for Android malware detection, featuring a series of layers optimized to process input data and generate output predictions. The model architecture begins with an input layer to accommodate sequential data, structured as a 3-dimensional tensor. This initial layer seamlessly transitions into an LSTM layer, strategically engineered to capture and retain long-term dependencies inherent in the input sequences, thereby enhancing the model's capacity to discern intricate patterns indicative of malware presence. To avoid overfitting, dropout layers are judiciously integrated following the LSTM layer, serving to systematically deactivate a portion of neural units during training iterations, thereby fortifying the model's generalization capabilities. After this, two dense layers, each utilizing Rectified Linear Unit (ReLU) activation functions, are deployed to instill non-linearity into the model, fostering its ability to delineate complex relationships within the data. Concluding the architecture is a single dense layer with a sigmoid activation function, culminating in binary classification. This layer computes the probability of an input sequence being classified as malware, lending the model its predictive capacity. The model compilation is executed meticulously, employing the Adam optimizer with a learning rate set at 0.001, while utilizing binary cross-entropy loss and accuracy as performance metrics. Hyperparameter tuning was carried out via GridSearchCV, optimizing parameters, such as learning rate, batch size, number of epochs, and optimizer. Following optimization, the model was trained on the training data and evaluated on the test set to assess its performance. Finally, the architecture for the optimized model was visualized deploying the plot_model function from TensorFlow.keras.utils. The model encompasses 74,881 trainable parameters, crucial for learning from input data and making precise predictions on new instances. Figure 5 portrayes the architecture of the LSTM model implemented.

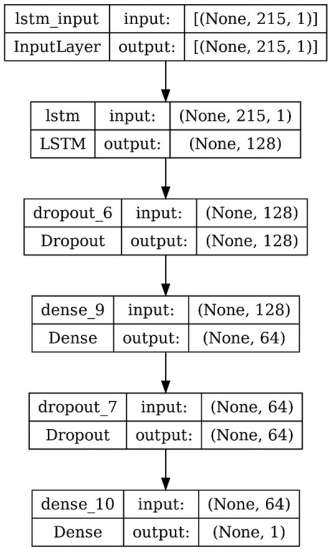

Fig. 5. Architecture of the LSTM model used.

The second model was a sequential NN consisting of three layers: two dense interleaved with a dropout layer for regularization. The first dense layer contains 128 neurons, each activated by a ReLU function, which yields an output shape of (None, 128). This layer contributes 27,648 parameters to the model. Following the dense layer, a dropout layer is inserted, having the same output shape, which randomly deactivates neurons with a dropout rate of 0.3 to prevent overfitting. The second dense layer comprises 64 neurons activated by ReLU, resulting in an output shape of (None, 64), and contributing 8,256 parameters. Another dropout layer follows with the same output shape, maintaining the dropout rate of 0.3. Lastly, a single neuron dense layer with a sigmoid activation function is appended for binary classification, producing an output shape of (None, 1) and adding 65 parameters. The total number of trainable parameters in the model amounts to 35,969, encompassing all the weights and biases across the layers, while there are no non-trainable parameters. Figure 6 illustrates the architecture of the NN model utilized.

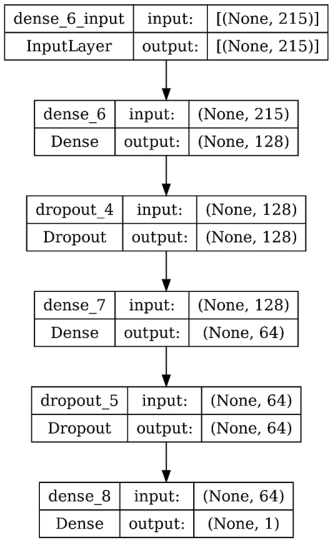

Fig. 6. Architecture of the NN model used.

D. Hyperparameter Tuning

To ensure that the models work well and do not focus too much on the training data, GridSearchCV was used for hyperparameter tuning, as noticed in Table II. GridSearchCV was favored for hyperparameter tuning due to its exhaustive search capability, systematically evaluating all possible combinations of hyperparameter values. Its simplicity made it straightforward to implement and understand, while its integration with cross-validation techniques ensured robust evaluation across different data subsets, reducing overfitting. Despite its computational intensity, GridSearchCV's scalability allowed for parallelized evaluations, optimizing model performance by identifying the optimal hyperparameter values across various ML algorithms and libraries.

E. Cross-Validation

Cross-validation was deployed to evaluate model performance robustness and reduce overfitting. This provides a reliable estimate of the model's generalization ability by reviewing performance across multiple data subsets and enhancing predictive accuracy on unseen data. A 10-fold cross-validation approach was employed.

TABLE II. HYPERPARAMETER TUNING FOR THE LSTM AND NN MODELS

Hyperparameter	Meaning	Values
units	Number of units in the layer	10, 20, 30
optimizer	Optimization algorithm for training	Adam, RMSprop
epochs	Number of training epochs	5, 10, 20
batch_size	Number of samples per gradient update	32, 64, 128

IV. PERFORMANCE EVALUATION OF THE MODEL

The performance of the LSTM and NN models was evaluated using four metrics: recall, F1-score, precision, and accuracy. Accuracy measures the proportion of the correctly identified malware applications out of all the applications classified as malware. Recall calculates the proportion of the correctly identified malware applications to the total actual malware samples, while precision measures the percentage of the correctly identified malware applications out of all the applications classified as malware. The formulas for accuracy, recall, and precision are detailed in (1)-(3).

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

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3)$$

where *TN* denotes true negatives, *TP* denotes true positives, *FP* denotes false positives, and *FN* denotes false negatives. The result for the F1-score was derived by (4) to assess the effectiveness of the models.

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

V. RESULTS AND DISCUSSIONS

For class 0, representing benign samples, both NN and LSTM achieved high accuracy, precision, recall, and F1-score, all 0.99, indicating robust performance in correctly identifying benign samples. For class 1, representing malware samples, both models demonstrated high accuracy and F1-score, with slight variation in precision and recall. The NN model achieved a slightly higher precision score for class 1 compared to the LSTM model, 0.99 versus 0.98, suggesting its effectiveness in correctly identifying malware samples while maintaining high overall accuracy. Therefore, the NN model appears to perform slightly better in detecting malware samples in this Android malware detection task, as shown in Table III. Figures 7-12 illustrate the accuracy and loss curves, confusion matrix, and ROC curves for both the LSTM and NN models. The ROC curve plots the TPR against the FPR, manifesting the classifier's discrimination ability. The AUC quantifies this performance by calculating the area under the ROC curve.

TABLE III. PERFORMANCE OF PROPOSED LSTM AND NN

Class	Accuracy	Precision	Recall	F1-Score	Model
0	0.99	0.99	0.99	0.99	NN
1	0.99	0.99	0.98	0.98	
0	0.99	0.99	0.99	0.98	LSTM
1	0.99	0.98	0.98	0.98	

This study used Tensor Processing Units (TPUs) v2.8, which are specialized circuits engineered by Google to accelerate training tasks in AI models. Featuring eight cores and 64 GB of memory, these TPUs accelerate the computational processes involved. Utilizing the TPU v2.8 configuration, the LSTM model underwent training for 10 epochs, completing the process in 23 s, which is significantly shorter compared to existing studies. In the future, there is a potential to integrate advanced DL architectures, including attention mechanisms or hybrid models that combine LSTM with convolutional layers. These integrations could improve model performance and adaptability, particularly to address evolving malware threats. Furthermore, exploring ensemble techniques, transfer learning approaches, or other ML approaches tested in different fields could be valuable in improving model generalization and scalability [15-18]. These avenues offer promising directions for extending the capabilities and robustness of malware detection systems.

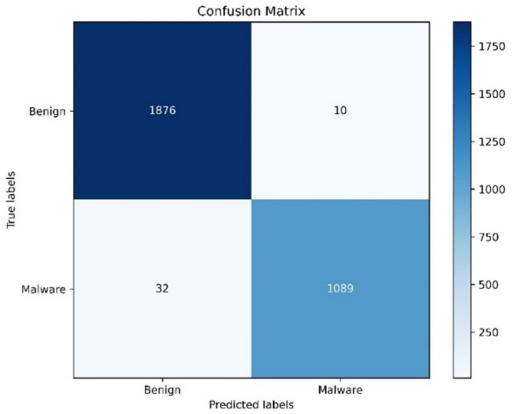

Fig. 7. Confusion matrix for the LSTM model.

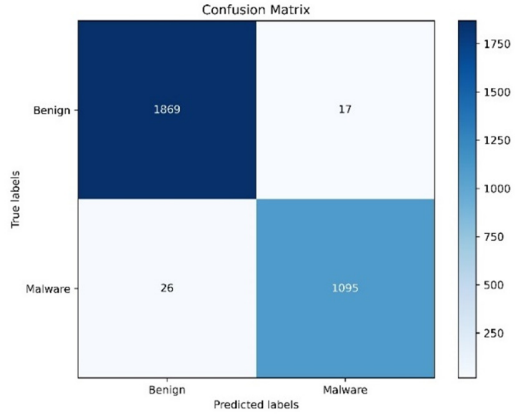

Fig. 8. Confusion matrix for the NN model.

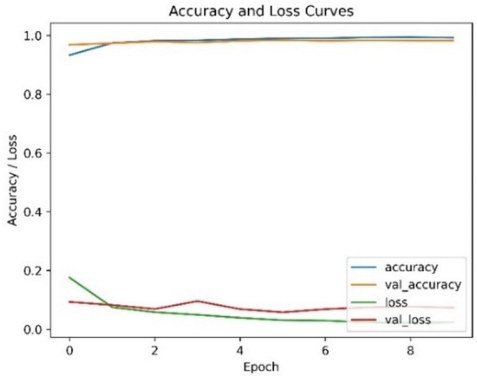

Fig. 9. Accuracy and loss curve for the LSTM model.

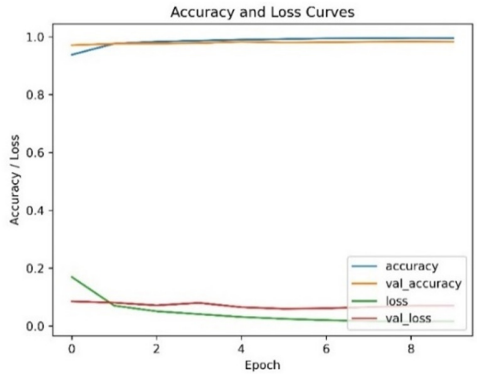

Fig. 10. Accuracy and loss curve for the NN model.

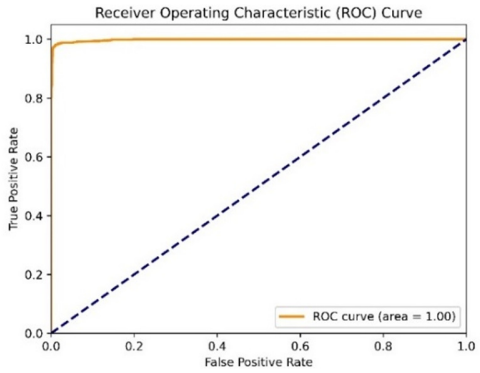

Fig. 11. ROC curve for the LSTM model.

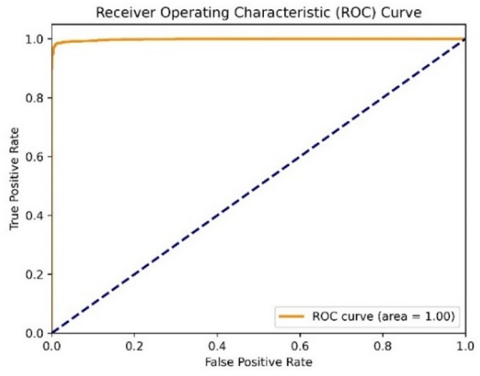

Fig. 12. ROC curve for the NN model.

VI. CONCLUSION

This study explored the development and evaluation of DL models for malware detection, with a particular focus on the LSTM and NN architectures. The analysis commenced with comprehensive data preprocessing techniques, including handling missing values and feature engineering, to effectively prepare the dataset for model training. In particular, oversampling techniques such as SMOTE were employed to address class imbalance, ensuring a more representative training dataset. Subsequently, two LSTM and NN models were constructed with multiple layers, leveraging the network's ability to process sequential data, particularly Android application features, for accurate malware classification. The model architectures were refined through proper hyperparameter tuning utilizing techniques, such as GridSearchCV, enhancing its performance and robustness. The evaluation of the models using key metrics revealed impressive results, with the NN model achieving accuracy, precision, recall, and F1-score of 0.99 for class 0, whereas the LSTM model attained 0.99 accuracy, precision, and recall for the same class. Notably, both models demonstrated high precision and effectiveness in distinguishing between benign and malicious applications. Also, the study underscored the importance of employing cross-validation and rigorous evaluation methods to ensure the reliability and robustness of model assessment. Regarding future scope, further research could explore the integration of advanced deep learning architectures, such as attention mechanisms, or hybrid models combining LSTM with convolutional layers, to potentially improve model performance and adaptability to evolving malware threats. Finally, investigating the applicability of ML approaches in various fields could be a worthwhile avenue for exploration.

ACKNOWLEDGMENT

Ahmed Alhussen like to acknowledge the Deanship of Postgraduate Studies and Scientific Research at Majmaah University for supporting this work under Project No. R-2024-1070.

REFERENCES

[1] C. S. Yadav *et al.*, "Malware Analysis in IoT & Android Systems with Defensive Mechanism," *Electronics*, vol. 11, no. 15, Jan. 2022, Art. no. 2354, https://doi.org/10.3390/electronics11152354.

[2] A. Al-Marghilani, "Comprehensive Analysis of IoT Malware Evasion Techniques," *Engineering, Technology & Applied Science Research*, vol. 11, no. 4, pp. 7495–7500, Aug. 2021, https://doi.org/10.48084/etasr.4296.

[3] D. Cao *et al.*, "DroidCollector: A High Performance Framework for High Quality Android Traffic Collection," in *2016 IEEE Trustcom/BigDataSE/ISPA*, Tianjin, China, Aug. 2016, pp. 1753–1758, https://doi.org/10.1109/TrustCom.2016.0269.

[4] T. Gueye, Y. Wang, M. Rehman, R. T. Mushtaq, and A. Hassan, "Machine Learning for Control Systems Security of Industrial Robots: a Post-covid-19 Overview." Sep. 06, 2022, https://doi.org/10.21203/rs.3.rs-2022709/v1.

[5] C. C. U. López, J. S. D. Villarreal, A. F. P. Belalcazar, A. N. Cadavid, and J. G. D. Cely, "Features to Detect Android Malware," in *2018 IEEE Colombian Conference on Communications and Computing (COLCOM)*, Medellin, Colombia, May 2018, pp. 1–6, https://doi.org/10.1109/ColComCon.2018.8466715.

[6] L. Arora and K. Kumar, "Android Ransomware Detection Toolkit," in *2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST)*, Delhi, India, Dec. 2022, pp. 1–5, https://doi.org/10.1109/AIST55798.2022.10064946.

[7] N. J. Ratyal, M. Khadam, and M. Aleem, "On the Evaluation of the Machine Learning Based Hybrid Approach for Android Malware Detection," in *2019 22nd International Multitopic Conference (INMIC)*, Islamabad, Pakistan, Aug. 2019, pp. 1–8, https://doi.org/10.1109/INMIC48123.2019.9022790.

[8] M. Woźniak, J. Siłka, M. Wieczorek, and M. Alrashoud, "Recurrent Neural Network Model for IoT and Networking Malware Threat Detection," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5583–5594, Dec. 2021, https://doi.org/10.1109/TII.2020.3021689.

[9] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," in *Proceedings 2014 Network and Distributed System Security Symposium*, San Diego, CA, USA, 2014, https://doi.org/10.14722/ndss.2014.23247.

[10] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An Efficient Android Malware Detection System Based on Method-Level Behavioral Semantic Analysis," *IEEE Access*, vol. 7, pp. 69246–69256, 2019, https://doi.org/10.1109/ACCESS.2019.2919796.

[11] S. Y. Yerima and S. Sezer, "DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection," *IEEE Transactions on Cybernetics*, vol. 49, no. 2, pp. 453–466, Oct. 2019, https://doi.org/10.1109/TCYB.2017.2777960.

[12] K. Aldriwish, "A Deep Learning Approach for Malware and Software Piracy Threat Detection," *Engineering, Technology & Applied Science Research*, vol. 11, no. 6, pp. 7757–7762, Dec. 2021, https://doi.org/10.48084/etasr.4412.

[13] J. Kumar and G. Ranganathan, "Malware Attack Detection in Large Scale Networks using the Ensemble Deep Restricted Boltzmann Machine," *Engineering, Technology & Applied Science Research*, vol. 13, no. 5, pp. 11773–11778, Oct. 2023, https://doi.org/10.48084/etasr.6204.

[14] M. A. Haq, "Smotednn: A novel model for air pollution forecasting and aqi classification," *Computers, Materials and Continua*, vol. 71, no. 1, pp. 1403–1425, 2022, https://doi.org/10.32604/cmc.2022.021968.

[15] S. Merugu, K. Jain, A. Mittal, and B. Raman, "Sub-scene Target Detection and Recognition Using Deep Learning Convolution Neural Networks," in *ICDSMLA 2019*, 2020, pp. 1082–1101, https://doi.org/10.1007/978-981-15-1420-3_119.

[16] A. Bathula, S. Muhuri, S. kr. Gupta, and S. Merugu, "Secure certificate sharing based on Blockchain framework for online education," *Multimedia Tools and Applications*, vol. 82, no. 11, pp. 16479–16500, May 2023, https://doi.org/10.1007/s11042-022-14126-x.

[17] M. Suresh, A. S. Shaik, B. Premalatha, V. A. Narayana, and G. Ghinea, "Intelligent & Smart Navigation System for Visually Impaired Friends," in *Advanced Computing*, 2023, pp. 374–383, https://doi.org/10.1007/978-3-031-35641-4_30.

[18] S. Merugu, M. C. S. Reddy, E. Goyal, and L. Piplani, "Text Message Classification Using Supervised Machine Learning Algorithms," in *ICCCE 2018*, 2019, pp. 141–150, https://doi.org/10.1007/978-981-13-0212-1_15.