A COMPREHENSIVE EXPLORATION AND INTERPRETABILITY OF ANDROID MALWARE WITH EXPLAINABLE DEEP LEARNING TECHNIQUES

Diptimayee Sahu, Satya Narayan Tripathy*

Department of Computer Science, Berhampur University
India

*Corresponding Author, e-mail: ds.rs.cs@buodisha.edu.in

Abstract: This study introduces an innovative approach to tackle evolving Android malware threats using explainable artificial intelligence (XAI) methods combined with deep learning techniques. The framework enhances detection accuracy and provides interpretable insights into the model's decision-making process. The research utilizes the CICInvesAndMal2019 dataset for training with Deep Neural Network (DNN) techniques. It incorporates Shapley Additive Explanations (SHAP) and Local Interpretable Model-agnostic Explanations (LIME) XAI techniques to refine the model's features and understand its predictions. This framework uses permissions and intents as static features from Android apps. The proposed framework reduces the execution time, reducing model loss to an impressively lower value of 0.26, and exhibits a commendable accuracy of 97.86%.

Key words: Android, malware detection, DNN, XAI technique.

1. INTRODUCTION

Smartphones are now vital in daily life, with Android being the most recognized and widely accepted operating system (OS), holding over 70 percent of the global market share [1]. The impact of malware on Android devices varies based on its type and the attackers' objectives. For instance, ransomware encrypts data and demands a ransom, adware displays ads to generate revenue, trojans grant unauthorized access to the device, and rootkits give attackers full administrative control [2]. Modern malware employs sophisticated techniques to evade detection, posing a universal threat across all OS platforms. Android's rapid adoption and the concurrent growth of applications underscore its role as a leading smartphone OS. Google Play, a trusted app store, hosts about 3.55 million apps [3]. However, apps from third-party providers may expose devices to malicious content, and studies show Android experiences more attacks than other mobile operating systems [4].

The Android OS uses Android Package Kit (APK) files, which contain essential files for malware analysis. Key files like AndroidManifest.xml and classes.dex are extracted during reverse engineering for static analysis. AndroidManifest.xml includes package-level information such as permissions and metadata, while classes.dex contains

the compiled source code. Android's security measures, such as process isolation and permission-based access control, are designed to protect users. However, the current rate of Android security attacks indicates these measures are insufficient.

The dynamic nature of Android malware exposes the inadequacy of current detection methods. Research in malware detection focuses on machine learning (ML) and deep learning (DL) techniques. Challenges include the scarcity of labeled datasets and class distribution imbalances, which bias model performance. ML and DL models often lack transparency, making it hard to understand their predictions. Enhancing model interpretability is crucial for user trust. XAI offers a solution by providing transparency while maintaining performance. Research should develop techniques that clarify how DL models detect malware and highlight key decision-making features. This study introduces a classification framework for Android malware detection using Deep Neural Network (DNN) techniques aided by SHAP simplified data to improve detection accuracy. In this study SHAP is employed to identify and exclude non-impactful features, simplifying the data for final training and testing. LIME explainable technique is then employed on the model output to visualize prediction confidence and key features impacting that prediction, enhancing understanding and trust in the model.

The subsequent sections of this work are organized as: Section 2 lists all the related literature survey done for this proposed work, Section 3 describes the system process method, Section 4 describes the implementation and result discussion portion, and Section 5 concludes the work.

2. LITERATURE REVIEW

The limited research available on cybersecurity recommender systems was discussed in [5] and it was also pointed out that few studies concentrate on predicting future attacks to shorten response times. Most research emphasizes defensive strategies with quicker solutions. They conclude that more development is needed in this area due to the growing number of cyber-attacks and insufficient handling capabilities.

Various studies on XAI and its implementation in cybersecurity were surveyed in [6]. The survey addressed two significant areas: the use of XAI in cybersecurity and the vulnerabilities of XAI methods to attackers. The survey found that current approaches lack real-world scenarios, highlighting the need to bridge the gap between AI and cybersecurity. Additionally, it noted that XAI methods have vulnerabilities that attackers can exploit and identified a lack of countermeasures. The survey concluded that incorporating XAI could significantly enhance decision-making in various fields.

Experiments were conducted by [7] to address how temporal inconsistencies (samples collected at different intervals) can inflate ML model performance to unrealistic levels, such as 99%. These flawed experimental designs mislead model performance, making models unfit for real-world scenarios. The studies employ LIME explainable approaches to investigate ML model learning and suggests that XAI techniques should be implemented to help researchers better understand ML models, thereby building more effective detection systems.

A model was employed by [8], called PAIRED, validating it with three datasets: DREBIN-215, Malgenome-215, and CICMalDroid2020. They conducted feature

selection among 215 features, choosing 35 static features to keep the model lightweight. Achieving an accuracy of 0.9802 with a false negative rate of 0.0090, their approach utilized the SHAP XAI technique to enhance understanding of the model's internal operations and increase trust in its predictions.

The DREBIN Android malware dataset was utilized by [9] for training and validation. They represented features using TF-IDF weighting. Their study employed Support Vector Machine (SVM) and Bidirectional Encoder from Transformers Representations (BERT) classifiers, achieving accuracies of 0.9684 and 0.9591, respectively. They then utilized XAI techniques including Modern Portfolio Theory (MPT) Explainer, SHAP, and LIME to explain classifier predictions. The study highlighted that high feature attribute values indicate that small changes in those features significantly affect model predictions. Comparing explanation capacities, [9] concluded that the MPT explainer outperformed SHAP and LIME, as it evaluates both individual feature contributions and dependencies among features in predictions.

Several ML algorithms were employed by [10] including Extra Trees (ET), Random Forest (RF), and Support Vector Machine (SVM) for real-time malware detection using the TUANDROMD dataset. They conducted four experiments comparing different sampling techniques' effects on algorithm performance. The first experiment used imbalanced data, addressed by RandomOverSampler, SMOTETomek, and RandomUnderSampler methods. RandomOverSampler balanced datasets by adding positive samples, while RandomUnderSampler reduced negative samples. SMOTETomek generated synthetic samples using k-nearest neighbors. ET, RF, and SVM classifiers were evaluated. The study highlighted the ET-RandomOverSampler model's efficiency, achieving 99.53% accuracy in classifying malware and benign samples. Additionally, they employed SHAP and LIME XAI techniques to interpret the decision-making process of their model.

An explainable CNN model was designed for Android malware detection by [11] using the LIME explainable AI technique. They used the Drebin dataset for its balanced splits. They split the data for train and validation of their model into 90% training, 5% validation split, and then 5% for test split and achieved 98% accuracy, 0.98 precision, 0.98 recall, and 0.97 F1 score. Then they used the LIME explainability method to highlight feature importance. They compared CNN and LIME activations on each malware family they considered and spotted that both perform similarly in similar areas. Hence they stated that this method can be used by malware detection systems.

A hybrid detection model was designed by [12] on 3 diverse sets of features. They used static features (Intents and Permissions) for static analysis, DEX byte codes as images for image analysis, and system call sequences for dynamic analysis. Finally, they proposed their hybrid model combining the features of all the 3 models, and passed their result to a DNN classifier. The best result they achieved in their final DNN classifier was 90.9%.

CNN was employed on the CICAndMal2017 dataset by [13] on their image-based detection model. They used NumPy arrays to assume the data as grayscale images. They used different architectures to test the approach and compared their results. They achieved 68.1% accuracy and based on their analysis they profess that their CNN model can detect variants of malware.

3. SYSTEM PROCESS METHOD

There are some preliminary steps followed by a ML or DL model such as data preprocessing and feature selection. They are essential for refining the data and enhancing the model's performance to achieve a preferable outcome.

3.1. Data Preprocessing

To propose an effective framework, preparatory measures are crucial. The collected dataset undergoes data preprocessing to remove redundancies and employs feature engineering to select impactful features. This study uses permissions and intents from the CICInvesAndMal2019 dataset, a comprehensive benchmark provided by the University of New Brunswick, Canadian Institute for Cybersecurity [14]. Both training and test sets from this dataset are utilized based on their significance in detection, identified during a literature survey. Permissions listed in the AndroidManifest.xml file of APK packages, such as READ_PHONE_STATE, WRITE_EXTERNAL_STORAGE, and RECORD_AUDIO, are analyzed to detect unauthorized access by malicious applications. Intents are the objects used for navigation within and between applications that are examined to understand application intentions. The dataset includes 1662 benign samples and 568 malicious samples. Data cleaning is crucial to avoid unreliable outcomes. The dataset undergoes row reduction to eliminate duplicates, followed by data transformation using Python's StandardScaler technique. This technique scales independent features to have zero mean and unit variance, ensuring consistency in feature variance across the dataset and optimizing estimator performance during training. The standard score can be calculated using equation (1).

$$\alpha = \frac{x_i - \mu}{\sigma} \tag{1}$$

In equation (1), 'xi' represents the instances in the dataset, 'μ' is the mean of the instances, and '**σ**' is the standard deviation.

3.2. Feature Selection

After data cleaning, the next critical step is feature selection to identify relevant features from the dataset. The dataset, containing 8111 features including permissions and intents, undergoes knowledge discovery to select a subset of informative features from the original set 'F_i'. This study employs an entropy-based feature evaluation method, specifically Information Gain (IG), to assess each feature's contribution relative to the target variable. IG measures how much each feature reduces uncertainty about the target, aiding in discarding redundant features and selecting those that enhance classifier performance while minimizing overhead. The IG score for each feature is calculated using equation (2), where 'IG (F_i)' represents the feature's IG score derived from subtracting weighted entropies of respective columns from the target column's entropy. The entropy calculation of target column is employed in Equation (3).

$$IG(F_i) = E(Y) - E(F_i) \tag{2}$$

$$E(Y) = -P_m \log_2 P_m - P_b \log_2 P_b \tag{3}$$

In the above Equations (2) and (3), the target column entropy is denoted as 'E(Y)' for the target column 'Y' and 'E(F_i)' is the entropy for each feature column 'F_i', are

crucial in assessing the predicted probabilities, 'P_m' for malware samples and 'P_b' for benign samples. Figure 1 is an exemplary representation of features and their calculated IG Scores, arranged in descending order. Furthermore, Figure 2 visually represents the IG score ranking, with the X-axis delineating features and the Y-axis indicating IG scores. The selection of features for model training is determined through careful examination of the IG score of the features and their ranking.

.permission.C2D_MESSAGE\n	0.104908
<actionandroid:name="android.app.action.DEVICE_ADMIN_ENABLED"/>\n	0.080790
<actionandroid:name="com.google.android.c2dm.intent.RECEIVE"/>\n	0.080255
android.permission.SYSTEM_ALERT_WINDOW\n	0.080076
com.google.android.c2dm.permission.RECEIVE\n	0.079925
...	
<actionandroid:name="com.myfitnesspal.action.UPDATE_STATUS"/>\n	0.000000
<actionandroid:name="com.xime.latin.lite.PUSH_SKIN"/>\n	0.000000
<actionandroid:name="com.appturbo.nativeo.INSTALL"/>\n	0.000000
<actionandroid:name="com.box.android.EDIT_FILE"/>\n	0.000000
com.flyersoft.moonreader.permission.C2D_MESSAGE\n	0.000000

Length: 8111, dtype: float64

Figure 1. Corresponding IG score of the Features

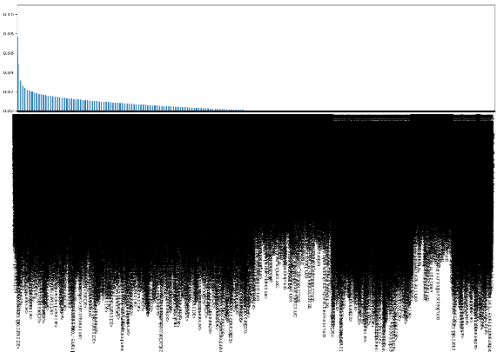

Figure 2. Representation for IG score of the Features

3.3. Detection Framework

The assessment of the suggested framework uses static attributes like permissions and intents for analysis. The framework encompasses distinct stages including data collection, data cleaning, feature selection, model formation and evaluation using the DNN technique, incorporation of SHAP explainable techniques to elucidate the model's performance on chosen features, iterative reconsideration of features based on interpretations, and eventual model generation and evaluation with the refined and simplified set of features and the final performance interpretation through LIME explainable technique. The DNN model consists of input, hidden, and output layers, with performance depending on feature selection and hyperparameter tuning. Input data is propagated through the network, predictions are computed, and weights are adjusted via backward propagation using gradient descent. In this proposed work the model output is again fed to the explainable technique to understand and interpret the feature importance of the individual features on the model prediction. In this work, the SHAP explainable technique is used on the predicted outcome and based on the analysis again the model is trained and tested on a simplified final set of features. The proposed system workflow is shown in Figure 3, and the computations done by the model is shown in Equation (4) and Equation (5).

$$Z = \sum_{l=1}^{n} \sum_{p=1}^{q} (f_{Relu}[b + \sum_{i=0}^{m} x_i w_i]) \quad (4)$$

$$f_{out} = f_{sigmoid}(Z) \quad (5)$$

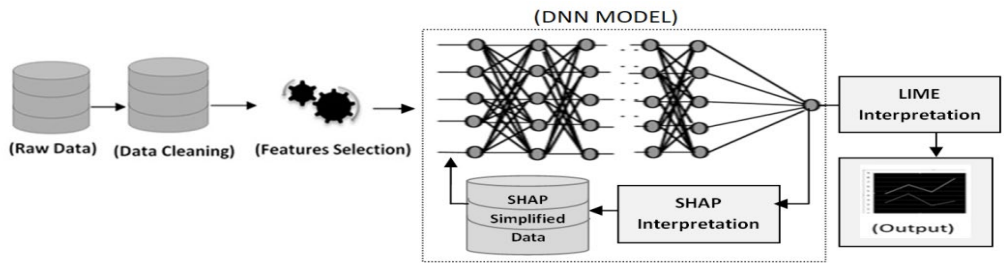

Figure 3. Proposed System Architecture

In Equation (5), ' f_{out} 'is the final output of the model that is calculated by applying sigmoid activation function '$f_{sigmoid}$'on output 'Z' generated after computation of the hidden layers. In Equation (4), '$\sum_{i=0}^{m} x_i w_i$'is the weighted sum of the inputs ranging from '0 to m' at each neuron optimized using RELU activation function 'f_{Relu}' where, 'x_i' is the input, 'w_i' is the connected weight between the neurons, 'b' is the bias added to all the nodes, 'p' represents the number of neurons in a hidden layer that ranges from '1 to q' and 'l represents the number of hidden layer that ranges from '1 to n'.

Optimizing the model's hyper parameters involves several trials to handpick the best ones. The proposed framework includes 5 dense hidden layers for non-linear data transformations using rectified linear activation units (RELU). The output layer has one neuron and uses the sigmoid activation function for classification. During training and testing, network errors are optimized using binary cross-entropy as the loss function. To prevent over fitting, the L2 regularization technique is applied, penalizing each layer parameter. The network minimizes these penalties to reduce loss and improve prediction accuracy.

4. IMPLEMENTATION AND RESULTS

The experimental trials in this investigation were employed in Google Colab notebook, utilizing the 'Tesla T4' GPU, in conjunction with TensorFlow 2.8.0 [15], Keras [16], and Scikit-learn [17] Python libraries. The assessment of experimental results was conducted based on accuracy and loss metrics, computed using Equation (6) and Equation (7), respectively.

$$Accuracy = \frac{True_{positive}+True_{negative}}{True_{positive}+True_{negative}+False_{positive}+False_{negative}} \quad (6)$$

$$Loss = BCE(y.\bar{y}) = -[ylog(\bar{y}) + (1-y)\log(1-\bar{y})] \quad (7)$$

In Equation (7), 'BCE' is binary cross-entropy, '$\bar{y}$' is the predicted value and 'y' is the actual value.

4.1. Result Discussion on Feature Selected Data

This section discusses the experimental results obtained on the feature selected dataset. Based on observations during the feature selection process, the top 40% of total features were considered for further experimentation. Out of the 8111 features in the dataset, 3244 were identified as effective through the IG feature selection technique. These selected features were then used for model training and testing. The dataset was

split into 80% for training and 20% for testing. Multiple trials were conducted to optimize hyper parameters, minimizing model inconsistencies as measured by loss and maximizing model accuracy. The final model architecture includes five densely connected hidden layers, each with 250 neurons, and an output layer with a single neuron. A batch execution technique was selected for the model to process 128 samples per batch. The model underwent training and testing for 600 epochs, incorporating a dropout rate of 0.2 to enhance robustness.

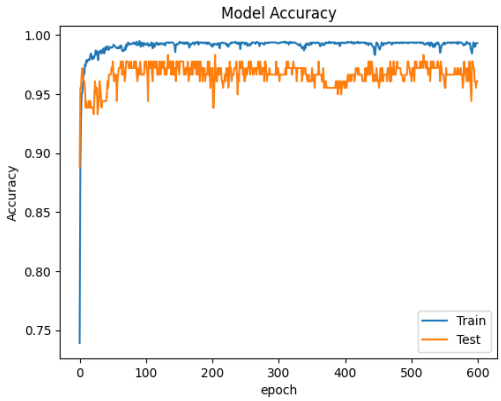

Figure 4. Model Accuracy for Malware Detection with Feature Selection

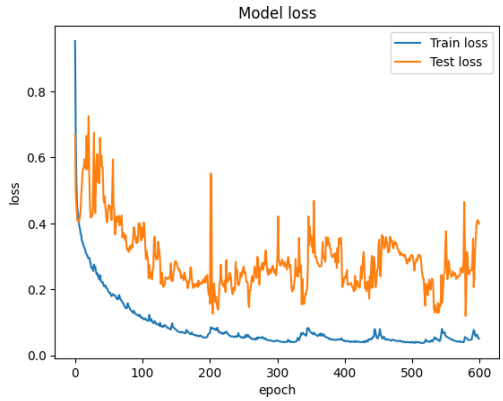

Figure 5. Model Loss for Malware Detection with Feature Selection

The framework's performance was measured using accuracy and loss metrics. Figure 4 illustrates the accuracy generated by the model, while Figure 5 represents the loss. In Figure 4, the x-axis represents the number of epochs, and the y-axis displays accuracies during the training and testing process at intervals of 0.05. Similarly, Figure 5 shows the number of epochs on the x-axis and the model's loss on the y-axis at intervals of 0.2. The model achieved an average accuracy of 96.67% and an average loss of 0.29 across epochs on the testing set of the feature-selected dataset, completing all 600 epochs in 325 seconds.

4.2. SHAP Explanations

DNN models are often complex and opaque, raising trust issues among users. In contrast, XAI aims to make models transparent and help researchers in understanding their model's results and take measures to improve efficacy. This study uses the SHAP library to calculate SHAP values, representing each feature's contribution to model predictions. These values provide a global explanation of feature importance. SHAP uses game theory concept to determine each feature's impact by comparing the model's performance with and without the feature. Higher SHAP values indicate greater feature contributions to predictions. The SHAP values help visualize and understand each feature's usefulness in the model's predictions.

Figure 6 represents the top 20 features sorted by their SHAP values, with the x-axis representing SHAP value distribution and the y-axis listing the features. Figure 7

presents a density scatter plot of these SHAP values, making it easy to identify each feature's impact on model output. In this plot, the x-axis represents prediction performance, and the y-axis represents individual features. High feature values indicate a significant influence on predictions, while low values indicate a lesser impact. The dots on the plot correspond to individual observations. The algorithm (ALGORITHM: SHAP EXPLAINER) explains the process of selecting the final simplified set of features based on the generated Shap_values.

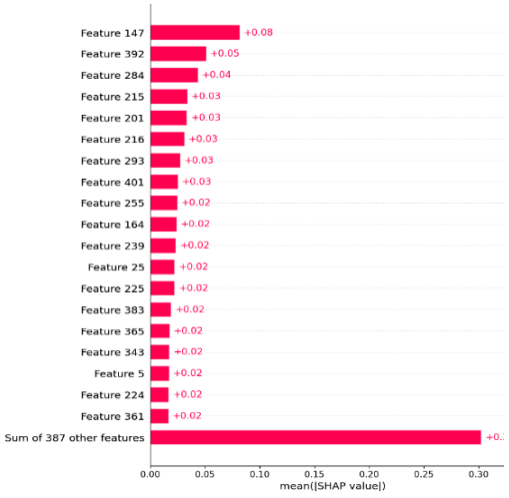

Figure 6. Summary plot based on Shap_values

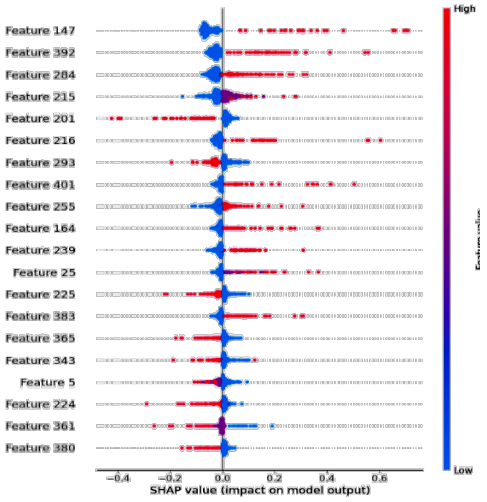

Figure 7. Scatter Plot of Sorted Features

ALGORITHM: SHAP EXPLAINER		
Input	**:**	**Data(SHAP simplified)**
Output	**:**	**val_accuracy and val_loss**
Step 1	:	Perform interpretation using: **SHAP($[DNN]_{output}$)**
Step 2	:	**SHAP simplified data:** 1. (Shap_values)$_i$= explainer.shap_values(X_i) 2. **Select:** (Shap_value)$_i$ >0
Step 3	:	1. Evaluate **Model(DNN)** with: Input layer (Input dimension = (Shap_value)$_i$) + Hidden layers (Relu+Hyperparameters) + Output layer (sigmoid) 2. Return (val_accuracy, val_loss)

4.3. Result Discussion on SHAP Simplified Data

In this section, the experiments are conducted on the SHAP simplified data. Based on the observation from the SHAP explainer and the generated Shap_values, out of 3244 selected features, only 406 features were detected to have significant Shap_values, while the remaining features showed negligible impact on the model's output. Therefore, the final feature selection includes 406 features for the proposed framework, enabling

experiments focused on enhancing model performance while reducing execution time, thereby making the framework more lightweight.

The model parameters remained consistent with 5 dense hidden layers, each containing 250 neurons, and 1 neuron in the output layer with a 0.2 dropout rate, akin to the previous experiment. The framework trained for 600 iterations. The framework demonstrated an impressive average accuracy of 97.86% on the test set of the SHAP simplified data, as depicted in Figure 8, with an impressively low average loss of 0.26, as shown in Figure 9. In both figures, the x-axis represents the number of epochs; in Figure 8, the y-axis represents the accuracy score at each epoch, and in Figure 9, it represents the loss generated at each epoch. The proposed model takes 132sec to execute 600 epochs. The duration of both experiments are detailed in Table 1.

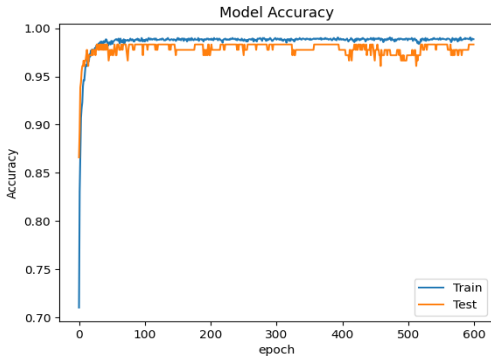

Figure 8. Model Accuracy for Malware Detection after XAI Interpretation

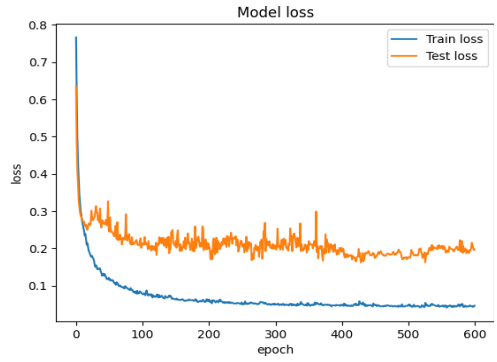

Figure 9. Model Loss for Malware Detection after XAI Interpretation

Table 1. Comparison of Experiment Durations

Experiments	***Time to execute 600 epoch***	***Accuracy***	***Loss***
On feature selected data	*325sec*	*96.67%*	*0.29*
On SHAP simplified data	*132sec*	*97.86%*	*0.26*

4.4. LIME Explanations

LIME is useful for interpreting complex models like DNNs, where understanding the decision-making process can be challenging. While high accuracy is important, it is not the only measure of trust. Providing explanations for individual predictions enhances transparency and trust in the model's efficacy. LIME explains the classifier's confidence that a particular instance belongs to a specific class and identifies the feature values that increase the likelihood of that classification. For example, in Table 2, X_test(0) is an instance with an actual output of "1 (malware)." LIME indicates a 98% confidence that X_test(0) belongs to class "1 (malware)." The Table 2 lists five instances tested by the model, comparing their actual class labels with the predicted labels and the confidence percentage that each instance belongs to the predicted class. Instead of just knowing the actual and predicted values, understanding the model's prediction confidence and the role of specific features can build user trust.

Figures 10, 11, and 12 illustrate the top 10 features from the LIME analysis for instances X_test(0), X_test(1), and X_test(2), respectively, showing feature values and their impact on each prediction. Figure 10 displays the top 10 features and their contributions to predict X_test(0) as "1 (malware)." Similarly, Figures 11 and 12 present the top 10 features and their contributions for instances X_test(1) and X_test(2), respectively.

Table 2. Analysis of Model Outputs with LIME

Test Samples	***X_test(0)***	***X_test(1)***	***X_test(2)***	***X_test(3)***	***X_test(4)***
Actual Output	*1*	*1*	*0*	*0*	*0*
Predicted Output	*1*	*1*	*0*	*0*	*0*
% of prediction by LIME	0 0.02 1 0.98	0 0.05 1 0.93	0 0.88 1 0.07	0 0.92 1 0.05	0 1.00 1 0.00

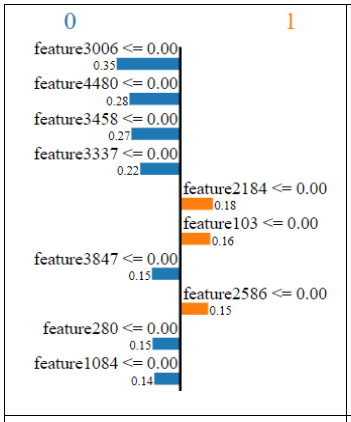

Figure 10. Feature Analysis of X_test(0)

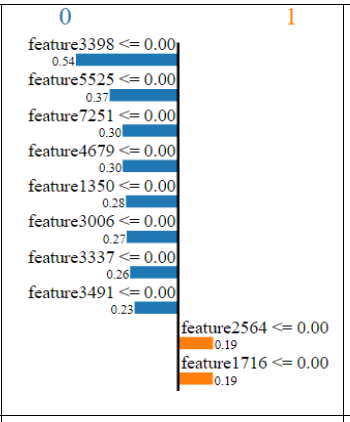

Figure 11. Feature Analysis of X_test(1)

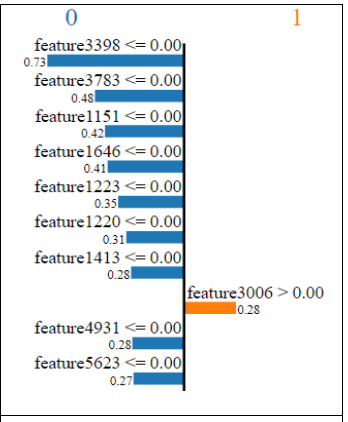

Figure 12. Feature Analysis of X_test(2)

This innovative framework is specifically designed for comprehensive malware detection in Android applications, aiming to deepen our insights into Android malware behavior. The time metrics in Table 1 highlight the lightweight nature of the proposed framework, with Table 2 demonstrating its efficiency and better performance compared to other approaches.

5. CONCLUSION

The intensity of Android malware attacks can vary significantly, from relatively benign adware and spam to sophisticated malware capable of stealing sensitive information, such as personal data, financial details, and login credentials. Depending on its capabilities, malware can cause various adverse outcomes, including financial losses, identity theft, privacy breaches, device malfunctions, and even the compromise of entire networks or systems. This proliferation of Android malware poses significant risks not only to individual users but also to organizations, governments, and critical

infrastructure systems, underscoring the importance of robust cybersecurity measures and proactive threat mitigation strategies.

Detecting malware is crucial, but building user trust is equally essential. Interpreting black-box models provides transparency to users and assists researchers in understanding and refining model inputs to enhance predictions. This proposed framework uses DNN techniques on static features extracted from Android packages to efficiently detect malware. The model output is analyzed using the SHAP XAI technique, and based on the output; the features are reconsidered and further simplified to make the model lightweight.

Experimental results demonstrate that the proposed system is both lightweight and efficient in malware recognition, achieving an impressive accuracy of 97.86% with a minimal loss of 0.26. In addition to detailing the model's accuracy, the study explains the model's confidence in its individual predictions and the contribution of each feature to those predictions using the LIME explainable technique. This comprehensive understanding of the model's capabilities and limitations helps build user trust.

The implementation of XAI in DNN models for Android malware detection is beneficial for timely locating vulnerabilities and taking necessary actions. Future experiments will explore a broader range of XAI techniques to enhance our understanding and transparency in the decision-making processes of our models, aiming to implement these approaches using deep learning techniques for Android malware detection.

REFERENCES

[1] Android Market Share. Accessed on 12.01.2023. Link: https://techjury.net/blog/android-market-share/.

[2] Android Malware. Accessed on 04.04.2024. Link: https://www.cyber.nj.gov/threat-landscape/malware/mobile-malware/android-malware.

[3] Number of apps available in leading app stores as of 3rd quarter 2022. Accessed on 15.02.2020. Link: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[4] Mobile Security: Android vs iOS. Accessed on 10.07.2023. https://www.kaspersky.com/resource-center/threats/android-vs-iphone-mobile-security.

[5] Ferreira, L., Silva, D. C., Itzazelaia, M. U. Recommender Systems in Cybersecurity. *Knowledge and Information Systems*, vol. 65, 2023, pp.5523–55, Springer

[6] Charmet, F., Tanuwidjaja, H. C., Ayoubi, S., Gimenez, P., Han, Y., Jmila, H., Blanc, G., Takahashi, T., Zhang, Z. Explainable artificial intelligence for cybersecurity: a literature survey. *Annals of Telecommunications*. https://doi.org/10.1007/s12243-022-00926-7. 2022

[7] Liu, Y., Tantithamthavorn, C., Li, L., Liu, Y. Explainable AI for Android Malware Detection: Towards Understanding Why the Models Perform So Well?. *arXiv:2209.00812v1 [cs.CR]*. 2022

[8] Alani, M., Awad, A.I.: PAIRED: An Explainable Lightweight Android Malware Detection System. *IEEE Access*, vol. 10, 2022, pp.73214-73228. doi: 10.1109/ACCESS.2022.3189645.

[9] Z. Lu and V. L. L. Thing. How Does It Detect A Malicious App?" Explaining the Predictions of AI-based Malware Detector. *2022 IEEE 8th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, Jinan, China. 2022. pp. 194-199, doi: 10.1109/BigDataSecurityHPSCIDS54978.2022.00045.

[10] Aslam, N., Khan, I.U., Bader, S.A., Alansari, A., Alaqeel, L.A., Khormy, R.M., AlKubaish, Z.A., Hussain, T. Explainable Classification Model for Android Malware Analysis Using API and Permission-Based Features. *Computers Materials & Continua*, vol.76, no.3, 2023, pp.3167-3188. doi: 10.32604/cmc.2023.039721

[11] Kinkead, M., Millar, S., McLaughlin, N., OKane, P. Towards Explainable CNNs for Android Malware Detection. *Procedia Computer Science,* vol. 184, 2021, pp. 959-965. https://doi.org/10.1016/j.procs.2021.03.118

[12] Oliveira, S.A., Sassi, J.R. Chimera: An Android Malware Detection Method Based on Multimodal Deep Learning and Hybrid Analysis. *TechRxiv*. 2020. https://doi.org/10.36227/techrxiv.13359767.v1.

[13] Lekssays, A., Falah, B., Abufardeh, S. A Novel Approach for Android Malware Detection and Classification using Convolutional Neural Networks. *SCITEPRESS – Science and Technology Publications, Lda*. (2020). doi: 10.5220/0009822906060614

[14] L. Taheri, A. F. A. Kadir and A. H. Lashkari, "Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls," *2019 International Carnahan Conference on Security Technology (ICCST)*, Chennai, India. 2019. pp. 1-8, doi: 10.1109/CCST.2019.8888430.

[15] TensorFlow. Accessed on 13.06.2020. https://www.tensorflow.org/.

[16] Keras. Accessed on 13.06.2020. https://keras.io/getting_started/.

[17] Scikitlearn, Machine Learning in Python. Accessed on 13.06.2020 https://scikit-learn.org/stable/.

Information about the authors:

Mrs. Diptimayee Sahu is working as a research scholar at Berhampur University, Odisha, India. Her research area is Deep Learning.

Dr. Satyanarayan Tripathy is working as Assistant Proffesor in Department of Computer Science, Berhampur University, Odisha since 2011. He is an active member of Life Member of Computer Society of India (LMCSI), Life Member of Odisha Information Technology Socity(LMOITS).

Manuscript received on 4 July 2024