Activation Functions Impact on Malware Detection: A Multi-layer Perceptron Approach for Android Applications

Shruti Brahma[1], Shweta Sharma[2], and C. Rama Krishna[3]

[1]Dept. of Computer Science and Engineering, JNTUH University of Engineering, Sultanpur
shrutibrahma@gmail.com
[2] Dept. of Computer Engineering, NIT Kurukshetra
shweta.sharma@nitkkr.ac.in
[3] Dept. of Computer Science and Engineering, NITTTR Chandigarh
rkc@nitttrchd.ac.in

Abstract. Cybersecurity is an evolving landscape and the need to discern malicious and benign applications has become paramount. One method of handling this challenge is by deploying machine learning models such as Multi-Layer Perceptron. A critical aspect of working with this model is the choice of activation functions in the hidden layers of the Multi-Layer Perceptron. The activation functions include the identity, logistic, hyperbolic tangent, rectified linear unit, and leaky rectified linear unit. This study aims to discern whether a given app is benign or malicious, primarily by scrutinizing the permissions and intents it requests during execution. Subsequently, a comprehensive analysis is conducted by combining the activation functions in various ways by considering different combinations for the hidden layers. The experimental results show that Leaky Rectified Linear Unit activation provides the highest accuracy (98.22%) when applied as a common activation function across all hidden layers of the Multi-Layer Perceptron. Additionally, in cases involving varied activation functions, the combination of leaky rectified linear unit, rectified linear unit, and hyperbolic tangent in different hidden layers achieves the best accuracy (97.91%) for detecting malicious applications.

Keywords: Android, activation functions, intents, malicious applications, multi-layer perceptrons, and permissions.

1 Introduction

Smartphones rely on applications to provide various functions and services which makes them an integral part of our daily lives. Alongside this, widespread malicious applications targeting smartphone devices pose a significant threat by compromising sensitive and private information stored on these devices. However, among all operating systems, Android users dominate the smartphone market,

primarily due to its increasing popularity.

Motivation: The Google Play Store boasts more than 2.5 billion applications, offering users a wide range of services, including online shopping, social networking, positioning, and navigation [1]. Consequently, this popularity has made Android a prime target for an escalating number of cyberattacks and malicious applications. The Kaspersky Security Network prevented a total of 5.7 million mobile malware attacks in 2023 [2]. The Quick Heal Annual Threat Report published in the year 2022 revealed the detection of 0.12 million Android malware cases [3]. According to the Kaspersky security report published in the year 2022, mobile malware exceeded 6.4 million attacks on Android-based smartphone devices [4]. Of these, more than 5 million were malicious installation packages, with 53,947 of them linked to mobile banking Trojans.

Figure 1 presents the trend of Android Malware from the year 2015 to 2022 [5]. Considering the vast number of malicious applications, there is an urgent requirement to establish a resilient malware detection framework capable of promptly identifying security threats.

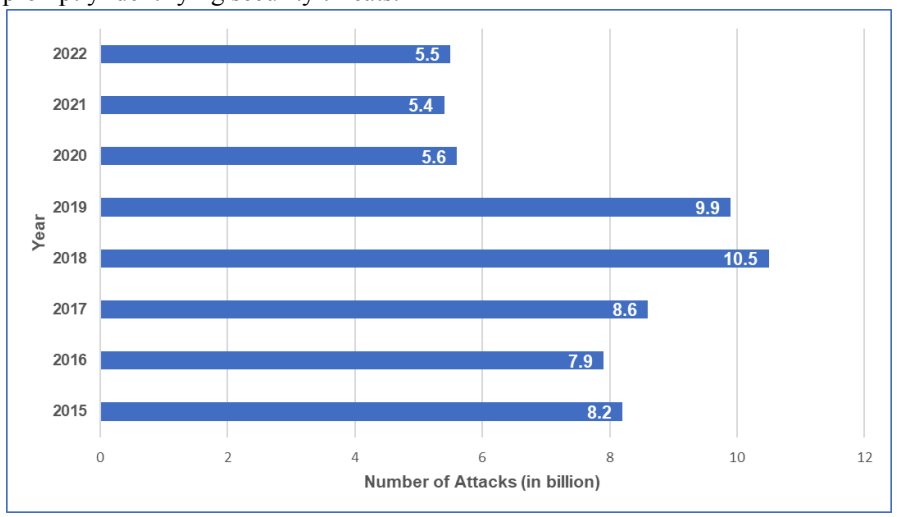

Fig. 1. Annual Number of Android Malware Attacks.

Contributions: In this experimental research work, we proposed a framework for the detection of Android malicious applications using Multi-Layer Perceptron (MLP). The detection involves identifying intricate patterns in diverse features, such as permissions and intents obtained by malicious applications. Therefore, MLPs are able to model non-linear relationships through the activation functions used in the hidden layers which makes them well-suited for capturing intricate patterns [18]. Consequently, this makes MLPs effective for the task of detecting Android malicious applications. Our contributions are twofold:

- First, we individually apply common activation functions—Identity, Hyperbolic Tangent (Tanh), Rectified Linear Unit (ReLU), and Leaky ReLU—across all hidden layers of the MLP. This approach enables a comprehensive comparison and investigation into the impact of each

activation function, determining which one yields the optimal results.

- Second, we explore a configuration by combining different activation functions—namely, Tanh, ReLU, and Leaky ReLU. These combined activations are simultaneously applied across various hidden layers, allowing us to analyze the ensemble that produces the most favorable outcomes.

Outline: Section 2 describes the Literature Survey. Section 3 shows the Dataset Description. Section 4 presents the Proposed Architecture with detailed explanations of all steps. Section 5 shows experimental results with detailed discussions and comparisons with the existing work. Section 6 concludes the paper.

2 Literature Survey

Several researchers worked on the analysis and detection of malicious applications from Android OS as summarized in Table 1. Alomari et al. [6] employed a dense layer model with Long Short-Term Memory (LSTM) using ReLU activation. Hwang et al. [7] presented an architecture by examining the platform-independent binary data. Liu et al. [8] applied a graph-based neural network (GBNN) approach for Android malware detection by analyzing the network traffic after considering node, edge, and different traffic flows and their attributes.

Wang et al. [9] presented a hybrid architecture with Deep Autoencoders (DAE) and Convolutional Neural Networks (CNN) by using the ReLU activation function. A detection method was introduced by Wei et al. [10] by using a one-dimensional CNN to extract network flow features. These features are processed by an independent Recurrent Neural Network (RNN) with a Leaky ReLU activation function.

Bawazeer et al. [11] applied Neural Network algorithms, including Full Order Radial Basis Function (RBF), CNN, and MLP by applying the ReLU activation function within the hidden layers. Sagar [12] used the Deep Belief Network (DBN) and applied Tanh activation functions in hidden layers. Sharma et al. [13] implemented MLP by applying the ReLU activation function in hidden layers to detect Android ransomware. Yadav et al. [14] implemented a CNN-based model using image-based representations of DEX files. Xia et al. [19] and Zhu et al. [20] applied MLP to detect Android malware. Dhanya et al. [21] employed MLP with ReLU activation function for network intrusion detection using the UNSW-NB15 dataset.

The existing literature focuses on applying similar activation functions in the hidden layers. These are a few papers [10] in the literature where researchers use the Leaky ReLU as an activation function in the hidden layers of MLP to detect Android malicious applications. However, no paper in the literature applies different combinations of activation functions in distinct hidden layers of Neural Networks to detect Android malicious applications.

Table 1. Summary of Literature Survey.

Research Paper	Year	Dataset	Malware	Machine Learning	Activation Function
Alomari et al. [6]	2023	Kaggle	5,560	LSTM	ReLU
Liu et al. [8]	2023	CICAndMal2017	5,465	GBNN	ReLU
Dhanya et al. [21]	2023	UNSW-NB15	2 million	MLP	ReLU
Yadav et al. [14]	2022	R2-D2	2 million	EfficientNet-B4 CNN	Softmax
Zhu et al. [20]	2021	Genome, VirusShare	25,577	MLP	ReLU
Bawazeer et al. [11]	2021	MNIST	6,000	CNN	ReLU
Sharma et al. [13]	2021	RansomProber	2,076	MLP	ReLU
Hwang et al. [7]	2020	KISA, VirusShare	50,000	DNN	ReLU
Wei et al. [10]	2020	CICAndMal2017	5,000	RNN	Leaky ReLU
Sagar [12]	2019	BaIoT	7 million	DBN	Tanh
Wang et al. [9]	2017	VirusShare	13,000	DAE-CNN	ReLU

3 Dataset Description

Several datasets are available to perform experimental work for the analysis and detection of Android malware [15]. The Android malware dataset used in this experimental work is sourced from the Drebin dataset, as documented in a study by Arp et al. [16]. It encompasses a collection of Android applications, categorized into two main groups: malicious applications and benign applications. The dataset comprises a substantial volume, with around 5,560 malicious applications and 9,476 benign applications. These applications were gathered over a period extending from August 2010 to October 2012. The specifics of this dataset, including information about its size and its original sources, are provided in Table 2, which likely offers more comprehensive insights into the dataset characteristics and origins for reference within the study.

Table 2. Dataset Specification.

Drebin Dataset	Specification
Size	6 GB
Number of malicious applications	5,560
Number of benign applications	9,476
Collection period	August 2010 to October 2012
Source	"https://www.sec.cs.tu-bs.de/~danarp/drebin/"

4 Proposed Architecture

The proposed architecture for the detection of malicious applications is shown in Figure 2 which is explained as follows:

4.1 Reverse Engineering of APKs

Reverse Engineering of an Android Package Kit (APK) involves systematic analysis and deconstruction of compiled Android applications to understand their internal structure and behavior. This step exposes potential malicious code of Android applications. The Apktool [17] has been used to perform reverse engineering that extracts the program source code from the APKs.

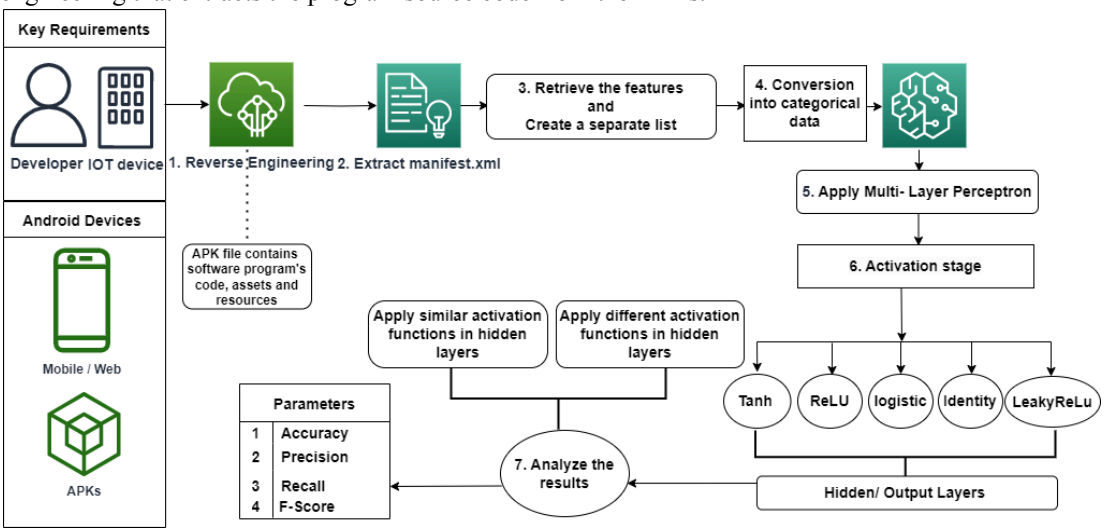

Fig. 2. Proposed architecture for detection of malicious applications.

4.2 Extract manifest.xml

After reverse engineering, the Androidmanifest.xml file has been extracted from the malicious and benign APKs. This step involves parsing the AndroidManifest.xml record which has important metadata such as app permissions and intents. This allows in-depth evaluation of app functionalities and enables finding potential malicious functionalities.

4.3 Retrieve the features and create a separate list

This step involves the creation of a distinct listed catalog of features such as permissions and intents in a CSV file. A total of 214 features are incorporated to offer a comprehensive representation of malware behavior and characteristics. This curated list facilitates in leveraging the divergence between benign and malicious permission patterns. This demarcating between legitimate and potentially malevolent requests helps in creating robust machine learning models.

4.4 Conversion into categorical data

This step involves transforming a vast dataset of app permissions and intents into binary format. Each permission and intent corresponds to a specific action on the mobile device. When an application requests permission or intent to access a feature, it is represented as “1”, otherwise it is marked “0”.

4.5 Apply Multi-Layer Perceptron

MLP is a neural network where connected nodes assist in the detection of malware [13]. The process starts by entering app-related features such as permissions and intents into a suitable format (in binary values: 1 for allowed and 0 for denied). Subsequently, the model undergoes intense training on the extracted features, facilitating it to learn and adapt to malware behavior. Once trained, MLP is capable of effectively predicting the maliciousness of new applications based on previously learned data and patterns.

4.6 Activation Stage

The activation functions applied in multiple hidden layers and output layer of MLP hold a profound significance in the arena of malware detection. During the forward pass of the neural network, the input data undergo linear transformations that involve weights and biases. The resulting values are then passed through an activation function, which identifies significant nonlinearities in the model. This

nonlinearity ensures that the network can capture complex relationships in the data, which in this case is a characteristic difference between malware and benign applications. Several activation functions including Identity, Logistic, Tanh, ReLU, and Leaky ReLU serve a vital role in this stage which are discussed as follows:

Identity Activation Function

It is a linear activation function where the activation is proportional to the input. Mathematically, it is defined in equation 1 where *x* is the input feature.

$$f(x) = x \qquad (1)$$

The identity activation function is applied in the hidden layers to maintain linearity between the output and the input. However, it fails when any form of non-linearity is introduced [18].

Logistic Activation Function

It is also known as the Sigmoid activation function which is significantly known for its ability to combine non-linearity into the network. It is defined in equation 2 where *x* is the input features.

$$f(x) = \frac{1}{\left(1+e^{-x}\right)} \qquad (2)$$

In the output layer, the logistic activation function is applied for tasks involving binary classification. It effectively predicts the probability of an input, with output values naturally constrained within the range of 0 to 1. However, it suffers from a limited output range and is not applicable for multi-classification [18].

Tanh Activation Function

It is a transformative bridge between input signals and subsequent neuron outputs. It is mathematically represented in equation 3. Here *x* is considered as the input feature.

$$f(x) = \left(\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\right) \qquad (3)$$

The tanh activation function is applied in the hidden layers as it offers a zero-centered output and ranges between -1 and 1 which results in balanced output and faster convergence. However, the Tanh is zero-centered but the gradient isn't, it is always positive and varies between 0 and 1 [18]. This discrepancy can result in undesired behaviors during training.

ReLU Activation Function

It is the widely used activation function in neural networks. It exhibits a behavior where, for non-positive inputs (negative or zero), the output is consistently 0, while for positive inputs, the output mirrors the input value exactly. This can be expressed mathematically as equation 4, where *x* represents the input feature.

$$f(x) = max(0, x) \qquad (4)$$

The ReLU activation function is used in the hidden layers because of its range, spanning from 0 to infinity. It addresses the gradient issue and does not saturate

for positive inputs. However, for all negative input values, the output is 0 which leads to the Dying ReLU problem, where the output of the neurons is consistently zero, resulting in zero gradients [18].

Leaky ReLU Activation Function
It is an adaptation of the standard ReLU function, intended to mitigate the issue of "Dying ReLU." It is mathematically defined in equation 5, where *x* represents the input features, and 'a' is typically a small negative value, often chosen between 0.01 and 0.1.

$$f(x) = max(a * x, x) \qquad (5)$$

In the hidden layers, the leaky ReLU activation function is utilized, characterized by its range extending from - infinity to + infinity. It prevents the "Dying Relu" problem and improves the performance of deep learning models [18].

4.7 Analyze the results

At the concluding stage of analysis, we meticulously investigate the impact of distinct activation functions, namely Tanh, Relu, Identity, and Leaky Relu, separately in each hidden layer of the model. Additionally, we explore a configuration in which we combine the Tanh, ReLU, and Leaky ReLU as activation functions within different hidden layers. We select the logistic activation function for the output layer. This comprehensive evaluation has been done to gain insights into which combination of activation functions optimizes the model's ability to effectively distinguish between benign and malicious applications. The model's performance is assessed by using four vital parameters, which are Accuracy in equation(6), Precision in equation (7), Recall in equation(8), and F-score in equation (9).

$$Accuracy = \frac{True\ Positives + True\ Negatives}{True\ Positives + False\ Positives + True\ Negatives + False\ Negatives} \qquad (6)$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \qquad (7)$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \qquad (8)$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \qquad (9)$$

5 Experimental Results

The proposed model has been implemented in Google Colaboratory with Python programming. The research process adopts a two-step approach, involving

training and testing. To ensure effective model training, the dataset is divided into two primary components: Features (referred to as *X*) and the corresponding labels (referred to as *y*), representing benign and malicious applications. The *X* component constitutes the feature vector upon which the model will be trained and tested. For a reliable model evaluation, 70% of the data has been allocated to the training set, with the remaining 30% for testing.

The core of the proposed work centers around exploring various activation functions in the MLP classifier. We systematically vary and apply different combinations of activation functions in the hidden layers of MLP classifiers. Specifically, in this paper, the impact of activation functions, including Identity, Tanh, Relu, and LeakyRelu are examined on the model's performance. Following the training phase, the MLP classifier undergoes a thorough evaluation using performance metrics, which encompass accuracy, precision, recall, and the F1 score.

5.1 Hyper-parameters Analysis

Table 3 describes the hyper-parameters used in the MLP which has 3 hidden layers. It's scheduled for training over 200 times. The learning rate is set to 0.01, which regulates the pace of model updates. The model uses Stochastic Gradient Descent (SGD) as its optimization technique. Unlike traditional methods that update weights after processing the entire dataset, SGD tweaks them after each individual data point. This aids in bypassing local minima. The Binary Cross-Entropy (BCELoss) loss function is used in this model for binary classification. It measures the difference between the true labels and the predicted probabilities.

Table 3. Hyper-parameters.

Hyper-Parameters	Values
Activation Functions	Identity, Tanh, ReLU, LeakyReLU
No. of Hidden Layers	3
No. of Iterations	200
Learning Rate (Alpha)	0.01

5.2 Comparative Analysis of Activation Functions in Experimental Results

Table 4 shows the experimental results of applying common activation functions such as Identity, Tanh, Relu, and Leaky ReLU in all hidden layers of MLP. Overall, each configuration produced high-performance metrics, with accuracy rates all above 97%. However, the Leaky ReLU activation function led to the highest accuracy of 98.22% and F-Score of 97.52%. This is because it avoids the "dying ReLU" issue to ensure that neurons continue to learn by introducing a small, non-zero gradient for negative inputs.

Table 4. Experimental Results with Common Activation Functions in Hidden Layers.

Activation Functions (in MLP)	Accuracy	Precision	Recall	F-Score
Identity+Identity+Identity	0.9791	0.9714	0.9702	0.9707
Tanh+Tanh+Tanh	0.9807	0.9727	0.9733	0.9730
ReLU+ReLU+ReLU	0.9816	0.9739	0.9745	0.9742
LeakyReLU+LeakyReLU+Leaky ReLU	**0.9822**	0.9716	0.9789	**0.9752**

Table 5 shows the experimental outcomes of employing different combinations of activation functions in an MLP with three hidden layers of MLP. It has been observed that applying the Leaky ReLU in the first hidden layer, ReLU in the second hidden layer, and Tanh activation function in the third hidden layer exhibits their best overall performance in terms of accuracy (97.91%) and F-Score (97.08%), making it the strongest combination for detection of malicious applications.

The reason for the high accuracy and F-Score in the "Leaky ReLU+ReLU+Tanh" combination is attributed to the unique range each activation function operates within. Leaky ReLU has a range of -infinity to +infinity, allowing it to retain a broad spectrum of values, while ReLU has from 0 to infinity, and Tanh from -1 to 1. When Tanh is used as the first-layer activation function, it can potentially constrict the range of results. However, having Leaky ReLU in the first layer followed by ReLU in the second layer ensures a vast range, possibly contributing to better outcomes.

Table 5. Experimental Results with Varied Activation Functions Combinations in Hidden Layers.

Activation Functions (in MLP)	Accuracy	Precision	Recall	F-Score
Tanh+ReLU+Leaky ReLU	0.9776	0.9695	0.9677	0.9686
Leaky ReLU+ReLU+Tanh	**0.9791**	0.9702	0.9714	**0.9708**
Leaky ReLU+ Tanh + ReLU	0.9784	0.9713	0.9683	0.9698

5.3 Results of the proposed framework compared with existing frameworks

In this section, we conduct a comparison between the outcomes of our proposed framework and several existing frameworks used for the detection of Android malicious applications. Figure 3 illustrates that our proposed framework outperformed the alternatives, achieving the highest accuracy of 98.22% when compared to the existing frameworks such as MKDIL [19], DAE-CNN [9], EfficientNet-B4 [14], and SEDMDroid [20] in the task of malicious application detection.

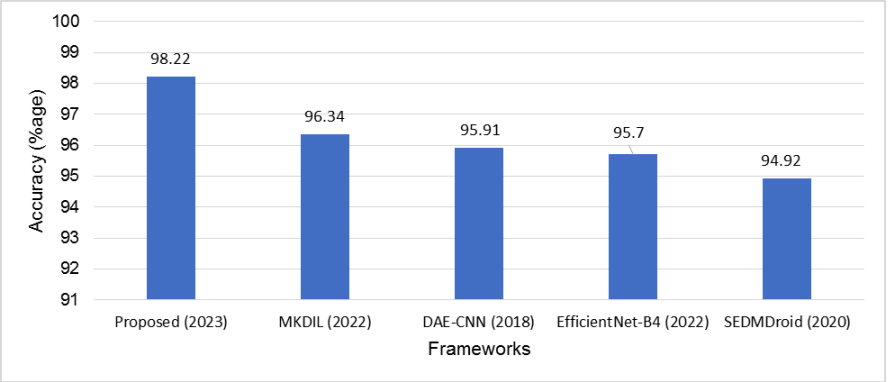

Fig. 3. Comparison of the proposed framework with the existing frameworks.

6 Conclusion

In this research paper, a framework based on the MLP model was introduced for the detection of malicious Android applications. The efficacy of the framework was assessed by experimenting with various combinations of activation functions in the hidden layers of the MLP. The experimental findings consistently demonstrated that using Leaky ReLU in all hidden layers of the MLP outperformed other activation functions, including Identity, Tanh, and ReLU. More precisely, it was consistently observed that Leaky ReLU produced the best outcomes, delivering an accuracy of 98.22% and an F-Score of 97.52%.

However, when considering diverse activation functions in different layers, a combination of Leaky ReLU in the first layer, ReLU in the second layer, and Tanh in the third layer (referred to as the Leaky ReLU+ReLU+Tanh ensemble) produced the best results, with an accuracy of 97.91% and an F-Score of 97.08%.

Overall, the common activation function is the most effective and gives noteworthy results over the performance of mixed combinations. Our findings will be instrumental in choosing the correct activation function that can affect the susceptibility of the network to certain features of malicious applications.

Moreover, our proposed framework will be significant for smartphone security and could have far-reaching implications in safeguarding users' devices from malicious applications. The future scope of our research involves optimizing Android malware detection through diverse activation functions, including Exponential Linear Unit (ELU), Scaled Exponential Linear Unit (SELU), Parametric Rectified Linear Unit (PReLU), and Gaussian Error Linear Unit (GELU).

References

[1] “Google Play Store: number of apps 2023,” *Statista*. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-

google-play-store/ (accessed Oct. 14, 2023).
[2] A. Kivva, “IT threat evolution in Q2 2023. Mobile statistics,” *Kaspersky*, Aug. 30, 2023. Accessed: Nov. 02, 2023. [Online]. Available: https://securelist.com/it-threat-evolution-q2-2023-mobile-statistics/110427/
[3] Quick Heal Technologies Limited, “Quick Heal Annual Threat Report 2022.” https://www.quickheal.co.in/documents/threat-report/Quick-Heal-Annual-Threat-Report-2022.pdf (accessed Oct. 18, 2023).
[4] T. Shishkova, “IT threat evolution in Q1 2022. Mobile statistics,” *Kaspersky*, May 27, 2022. [Online]. Available: https://securelist.com/it-threat-evolution-in-q1-2022-mobile-statistics/106589/
[5] “Number of malware attacks per year 2022,” *Statista*. https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/ (accessed Nov. 01, 2023).
[6] E. S. Alomari *et al.*, “Malware Detection Using Deep Learning and Correlation-Based Feature Selection,” *Symmetry*, vol. 15, no. 1, p. 123, Jan. 2023.
[7] C. Hwang, J. Hwang, J. Kwak, and T. Lee, “Platform-Independent Malware Analysis Applicable to Windows and Linux Environments,” *Electronics*, vol. 9, no. 5, p. 793, May 2020.
[8] T. Liu, Z. Li, H. Long, and A. Bilal, “NT-GNN: Network Traffic Graph for 5G Mobile IoT Android Malware Detection,” *Electronics*, vol. 12, no. 4, p. 789, Feb. 2023.
[9] W. Wang, M. Zhao, and J. Wang, “Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, Apr. 2018.
[10] S. Wei, Z. Zhang, S. Li, and P. Jiang, “Calibrating Network Traffic with One-Dimensional Convolutional Neural Network with Autoencoder and Independent Recurrent Neural Network for Mobile Malware Detection,” *Security and Communication Networks*, vol. 2021, pp. 1–10, Feb. 2021.
[11] O. Bawazeer, T. Helmy, and S. Alhadhrami, “Malware Detection Using Machine Learning Algorithms Based on Hardware Performance Counters: Analysis and Simulation,” *Journal of Physics: Conference Series*, vol. 1962, no. 1, p. 012010, Jul. 2021.
[12] G. V. R. Sagar, “Malware Detection Using Optimized Activation-Based Deep Belief Network: An Application on Internet of Things,” *Journal of Information & Knowledge Management*, vol. 18, no. 04, p. 1950042, Dec. 2019.
[13] S. Sharma, R. Krishna, and R. Kumar, “An Ensemble-based Supervised Machine Learning Framework for Android Ransomware Detection,” *The International Arab Journal of Information Technology*, vol. 18, no. 3A, 2021.
[14] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham, “EfficientNet convolutional neural networks-based Android malware detection,” *Computers & Security*, vol. 115, p. 102622, Apr. 2022.
[15] S. Sharma, N. Kumar, R. Kumar, and R. K. Challa, “The Paradox of Choice: Investigating Selection Strategies for Android Malware Datasets Using a Machine-learning Approach,” *Communications of the Association for Information Systems*, vol. 46, no. 1, pp. 619–637, 2020.
[16] 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*, 2014.
[17] C. Tumbleson, “Apktool,” *Apktool*. https://apktool.org/ (accessed Nov. 01, 2023).
[18] “Neural Networks and Deep Learning,” *Coursera*.

https://www.coursera.org/learn/neural-networks-deep-learning/ (accessed Oct. 20, 2023).

[19] M. Xia, Z. Xu, and H. Zhu, “A Novel Knowledge Distillation Framework with Intermediate Loss for Android Malware Detection,” in *2022 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, Dec. 2022.

[20] H. Zhu, Y. Li, R. Li, J. Li, Z. You, and H. Song, “SEDMDroid: An Enhanced Stacking Ensemble Framework for Android Malware Detection,” *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 984–994, Apr. 2021.

[21] K. A. Dhanya, S. Vajipayajula, K. Srinivasan, A. Tibrewal, T. S. Kumar, and T. G. Kumar, “Detection of Network Attacks using Machine Learning and Deep Learning Models,” *Procedia Computer Science*, vol. 218, pp. 57–66, 2023.