







Machine Learning-Based Malware Detection Using Frequency Features

Firas Omar Ahmed¹, Helen Farqad Taha², Mustafa Mahmood Yahya^{3*}, Abdulelah Hameed Yaseen⁴

¹ Technical Computer Engineering Department, Technical College of Engineering, Al-Kitab University, Kirkuk 36015, Iraq

² Department of Cybersecurity Techniques Engineering, College of Computer Engineering Technology and Artificial Intelligence - Kirkuk, Northern Technical University, Mosul 41001, Iraq

³ Technical Mechatronics Engineering Department, Technical College of Engineering, Al-Kitab University, Kirkuk 36015, Iraq

⁴ Department of Petroleum Engineering, College of Engineering, Al-Kitab University, Kirkuk 36015, Iraq

Corresponding Author Email: almawalym@uoalkitab.edu.iq

Copyright: ©2025 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/ijssse.150909>

ABSTRACT

Received: 14 August 2025

Revised: 3 September 2025

Accepted: 14 September 2025

Available online: 30 September 2025

Keywords:

malware detection, SVM, RBF, MLP, computer security, dynamic analysis, cybersecurity, PSD

Malware, which mimics an authentic program to perform illicit operations, poses one of the most dangerous threats to cybersecurity. When it comes to detection techniques, the conventional methods comprise primarily the use of signatures, which certainly can work well for known threats but are powerless against the new and more camouflaged threats. While threats are dynamic in nature, a more advanced approach to analysis, known as dynamic behavioral analysis, is more effective in malware detection. This paper proposes a new framework that analyzes the Power Spectral Density (PSD) features of the malware's behavior to create a new set of distinguishing features. We classify these features using three machine learning (ML) models: Radial Basis Function (RBF) networks, Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP). We achieve the following detection accuracies: MLP at 94.9%, SVM at 96.9%, and RBF at 99% along with recall rates of 95.2%, 96.7%, and 98.8%, respectively. It can also be noted that the models presented here achieve high accuracy, which proves the good generalization of the models and their ability to classify software with high accuracy as benign or malicious. Incorporating PSD feature extraction with state-of-the-art machine learning algorithms, this work improves the detection and supplements the fortification of security measures against threat landscapes that have grown increasingly sophisticated.

1. INTRODUCTION

Malware is at present a massive and unrelenting menace to the significant, businesses, and vital assets. Malware is a general term for viruses, worms, Trojan horses, ransomware, spyware, and other software tools with the specific purpose of espionage, system sabotage, or system takeover [1, 2].

Therefore, these attacks pose a significant risk to privacy, financial stability, and national security. Due to the growing ferocity of malware attacks, the ability of hackers to use intricate measures such as elaborate evasion techniques and new zero-day exploits, the need for much more efficient and flexible approaches to identifying malware is now more dire [3].

Conventional virus detection methods primarily concentrate on signatures, identifying viruses by contrasting them with a collection of established signatures [4, 5]. Though it is successful at detecting previously known malware, it is completely reactive and incapable of detecting new, modified, or even hidden malware types [6, 7].

This limitation leaves systems vulnerable to emerging threats that can evade detection by signature methods. Consequently, risks have moved away from being defined by specific characteristics and toward methodical techniques used

in the early detection of suspicious activity [8]. Because malware is spreading with high speed and has a capability to evolve to avoid the defense put in place, there is a need to develop better approaches to analyzing it. Malware typically disguises itself as genuine software and is able to involve itself in unpleasant operations without discernment [9, 10].

Traditional signature detection methods excel in detecting familiar threats, but they fall short in identifying new or disguised threats that employ strategies to evade detection. In an attempt to address these limitations, dynamic analysis has been considered an essential tool in malware detection [11, 12].

This method involves running a program, such as a virus, in a controlled environment, such as a sandbox, and tracking its activities to determine its malicious intents [13]. Dynamic analysis plays a significant role in discerning the behavior of malware and its probable impacts on system resources, network connections, and other software modules, making it pertinent for identifying and responding to modern threats [14].

Indeed, the observation of the behavior of the malware in the environment allows for the receipt of as many behavioral characteristics as possible, which are paramount important when designing detection measures [15, 16]. However,

dynamic analysis is not without its own set of limitations. Today's malware versions include a variety of scripts that allow the malware to identify its operation in a sandbox and avoid detection [17, 18].

Such variants may cease the execution of their actions, postpone the infliction of harm, or even appear harmless during the analysis phase [19].

Moreover, given that the analysis is usually conducted only for a short period, starting with a few minutes, the range of behaviors that the malware produces may not be fully investigated, which hampers the analysis [20, 21].

These challenges highlight the strong need for further sophisticated methods to enhance the dynamic analysis results.

2. LITERATURE REVIEW

Traditional approaches of malware detection were based on signature detection, which, although effective initially, can no longer cope with the new generation of malware, such as zero-day and polymorphic. In references [22, 23], they have provided one of the first works on static analysis in learning, and in these literatures [24, 25], they also used static analysis in their work while also pointing at its weaknesses.

Over time, research has increasingly shifted toward dynamic and behavioral analysis. For instance, studies [26, 27] explained how dynamic analysis performed well in capturing behaviors that are associated with malicious activities at running time. However, this approach is still faced with challenges like high computational cost, complexity, and hence the complexity involved in feature extraction due to the sophisticated behavior of the malware.

İbrahim et al. [28] suggested the use of the static analysis of Android manifests together with a deep learning (DL) algorithm. By using features from 14,079 samples, the new features in addition to those used in the previous experiment yielded binary classification F1-scores of 99.5% and multi-class classification F1-scores of 97%.

Nonetheless, these findings are quite promising and while the approach they have used eliminates the use of heuristics and dynamic analysis of the code, the system remains insecure against dynamic or disguised malware. Moreover, the problem of dataset variation, possible overfitting, and absence of comparison with methods based on dynamic analysis challenge the applicability of the developed model.

In addition, Mohamad Arif et al. [29] also studied Android malware detection through static analysis using permission-based features. Through experimenting with 5,000 Drebin malware samples and 5,000 Androzoo benign samples with different classifiers, they realized that the Random Forest (RF) classification method had the highest accuracy of True Positive Rate (TPR) at 91.6%.

But their focus handicaps the analysis of the dynamic nature of malware within a confined static analysis approach. In addition, permission-based features may not be very effective against malware that utilizes other features of the applications.

Moreover, Sandeep [30] also designed a pre-installation malware detection model using a fully connected DL model. This approach was established to have attained an accuracy of 94.65% and was able to recognize the different malware names, versions, and packages.

However, the innovative approach of using the pre-installation detection does not recognize malware that becomes active after installation. Finally, the study fails to

provide specifics on the diversity of the dataset and actual performance, and for this reason, most of the findings of the study will not be usable in real-world circumstances.

On the other hand, Molina-Coronado et al. [31] provided a comprehensive survey to sum up the problem and potential of the ML-based detection of Android malware using the approaches that pay considerable attention to the reproducibility and comparability of the results. By looking at 10 key works, they also saw problems like sample duplication, lack of balance in datasets, and the absence of common assessment methodologies. However, as their experiment is based solely on historical data, their work is less relevant to more modern and, perhaps, unknown types of malwares. In addition, the counter-quantitative assessment methods make it very difficult to compare models.

Furthermore, Shatnawi et al. [32] proposed a static-based detection method utilizing Android permissions and API calls, tested on the CICInvesAndMal2019 dataset with three machine learning (ML) algorithms: Support Vector Machines (SVM), K-nearest neighbors (KNN), and Naive Bayes (NB). Although it increases the asker's detection scores, this made it only analyze malware distinguishing features based on static analysis, which does not capture dynamic aspects of malware.

Moreover, it was also noted that the chosen algorithms are sensitive to the quality of the data available within the dataset, as well as the methods involved in the selection of the features. Analyzing the possibilities of the combined usage of static and dynamic analysis for the detection of Android malware.

This study [33] found that RF and Decision Tree static methods resulted in 94% accuracy, while dynamic techniques using Principal Component Analysis (PCA) raised SVM and Neural Network (NN) models to an accuracy of 99%. However, there are some limitations in this approach; their approach involves only static and no dynamic analysis that can supplement the former.

Furthermore, the employment of perfect datasets and lack of evaluations that compare with actual evasion strategies decrease the work's applicability.

In contrast, the development of the mechanism used to detect malware has revealed the trends of new designs or modern approaches over the conventional methods, which are more complex. Amer and Zelinka [34] also developed a dynamic analysis method for IoT malware detection (DAIMD) involving convolutional neural networks (CNNs) in a nested cloud environment.

This method transforms behavior data into images that are most functional for CNN classification through examination of the memory, the network, and the system call of IoT malware. The approach can automatically identify both conventional and new IoT malware and lessen the possible harm.

However, it is limited by the type of computational resources needed for real-time monitoring and fails to work with insufficient or noisy data. Thus, it is still questionable what will become of highly obfuscated or polymorphic malware by the program.

Sihwail et al. [35] introduced a consolidated malware detection system that blends the memory analysis with dynamic analysis. This method was found to reconstruct artifacts out of memory and dynamic execution features in conjunction with SVMs, with an accuracy of 98.5% and a 1.7% false positive rate. Thus, eliminating such shortcomings as single-path file execution limitations in dynamic analysis, the approach expands the range of effective malware

detection.

However, memory analysis still suffers from the increased computational requirements and dependency on access to device memory. However, it fails to detect highly obfuscated malware like many other approaches mentioned earlier on this page.

Moreover, Hwang et al. [36] proposed a work under consideration that suggested the ransomware detection model based on two stages with the usage of the Markov model and RF. The tendencies of ransomware are described by the API sequences in a Markov model, and RF minimizes error rates.

The method obtained an accuracy of 97.3% with false positive and negative rates of 4.8% and 1.5%, respectively. One of the drawbacks is that it can be easily fooled by focusing on the sequence of API calls, which can be easily hidden with the help of obfuscation or API hooking by a modern ransomware.

The model's performance, however, decreases with new ransomware samples that are different from the samples used to train the model.

Specifically, using a technique known as "mining sandbox," da Costa et al. [37] proposed a hybrid of static and dynamic analysis on Android malware. This means that through a comparison of API calls in versions of the benign application, any increased, different activities in new versions of the application can be identified as malicious. Static analysis tools like DroidFax added to the dynamic analysis increased its efficiency; it was able to detect 43.75% of malware.

Furthermore, taint analysis performed comparably with dynamic tools as a supplement. However, the approach relies on the competence of static analysis tools, which might fail to detect some or all sorts of unnatural behavior. However, some of this advanced malware with the capability of evading analysis may not be detected in the sandbox.

Furthermore, to remove the complexity out of malware behavior identification, Jeon et al. [38] have proposed a technique of word embedding for API call sequence analysis. Similar traits are grouped using contextual relationships between API functions, while a Markov chain model is created for a semantic transition matrix to differentiate between malware and goodware. This method attained a detection precision of 99% and a false positive rate of 1%, with the accuracy of prediction of the malicious API call sequences being 99.7%.

However, the approach relies strongly on the qualities and variety of the API call sequence datasets. The random-use obfuscation and zero-day attacks may limit its application across the board, and pre-execution data restrict it further.

All of these works together exemplify the progress and future developments that are still issues in malware detection. Even though the means of hybrid approaches that utilize ML combined with dynamic analysis and context awareness increased the detection accuracy, there are some drawbacks: the time needed to compute the result, dependency on the quality of data, and the ability to evade the techniques of the attacks still remain some challenges.

The following should be open in future research focusing on improving malware detection frameworks: Feature extraction techniques should be incorporated into strong detection systems, real-time analysis should also be implemented, and improved, large datasets should also be used.

The traditional type of malware detection technique is divided into dynamic and static analysis [39, 40]. An analysis that is performed without running the software is known as

static analysis, making it rapid and scalable to large input volumes. However, static approaches are not so powerful in detecting such malware, including obfuscated malware, encrypted malware, and polymorphic malware, since these methods rely heavily on the patterns or signatures that have been previously discovered [41, 42]. Dynamic analysis contrastingly monitors a program in its executing state and logs behavioral attributes, such as system calls, file access, and network traffic [43, 44]. This allows more advanced or unknown malware to be identified through more dynamic detection techniques but is computationally more costly and is frequently vulnerable to evasion techniques when malware is aware of a sandboxed environment.

The initial experiments of the field of static analysis have proven to be efficient and scalable but have always claimed poor resistance to obfuscated and polymorphic malware and poor detection of zero-day behavior [22-25]. Later efforts turned to dynamic/behavioral analysis, with a better ability to detect runtime characteristics of malicious activity [26, 27] but also with practical challenges: greater computational cost, complex pipelines to extract features, and sensitivity to evasion in sandbox detection. Android-based static-ML systems (e.g., studies [28, 29, 32]) have demonstrated favorable headline results—e.g., F1-scores of up to 99.5% (binary) and 97% (multi-class) in this study [28], and TPR of 91.6% with RF in this study [29]—but their dependence on manifest/permission features makes them vulnerable to visibility into actual runtime behavior and can be fragile on different datasets or with different attack surfaces. Other static-ML experiments (e.g., pre-installation detection in this study [30], 94.65% accuracy) have drawbacks of applicability to malware execution after installation and various reports of dataset imbalance/duplication and unstable evaluation procedures, which prevent fair comparison and reproducibility [31].

Dynamical ML pipelines and hybrids aim to close such gaps. To illustrate this point as a single example, malware behavior on IoT was converted to CNN images to distinguish new families in this study [34], increasing the probability of distinguishing new families at the cost of high computational load and sensitivity to noisy/suboptimal data. The high accuracy (98.5%) and low FPR (1.7%) memory-augmented dynamic analysis suggested in this study [35] avoided the single-path execution problem, but the resource and memory-access demands were heavy, and highly obfuscated specimens were not well supported. Sequence-based learning (e.g., API-sequence Markov + RF in this study [36]; word-embedding over API calls in this study [38]) was also claimed to achieve high precision (e.g., 99% precision and 1% FPR; 99.7% sequence prediction accuracy in this study [38]) but can be fooled by API hooking/obfuscation and can also fail to generalize to truly novel behavior. Hybrid schemes based on static+dynamic (e.g., the mining-sandbox strategy in this study [37]) also boosted coverage (e.g., 43.75% detection in that environment) but still rely on the quality of the tools used in the static phase and cannot stop sandbox-evasive malware. It is interesting to note that several studies recognize ongoing problems with dataset diversity, balance, and standardized evaluation that make it difficult to claim that one method is better than the other [31, 33].

In order to balance behavior visibility with feature strength and computational feasibility, our work presents a Power Spectral Density (PSD)-based behavioral representation that projects runtime malware activity to the frequency domain,

producing unique, compact features that are insensitive to superficial code-level obfuscations. Combining these PSD properties with various ML classifiers allows our framework (i) to maintain dynamic intuition without having to support prohibitively costly behavior pipelines, (ii) to improve generalization by learning stable spectral patterns of malicious activity but not fragile syntactic signals, and (iii) to improve comparability based on explicit, quantitative reporting (accuracy and recall) and cross-model analysis. Taken together, this design specifically addresses the fundamental areas of weakness that were found in both the static-only ML pipelines (poor behavioral coverage) and the heavy dynamic platforms (cost/complexity and sandbox evasion), providing a balanced, effective approach to modern malware detection.

To address these shortcomings, our work presents a new framework based on PSD features to describe malware behavior in the frequency domain, which has a much more descriptive and unique presentation of malicious activity. Uniting the PSD feature extraction with advanced machine learning models, not only does our approach leads to higher detection accuracy, but it also provides better generalization properties than the traditional methods based on static or dynamic only.

To address these challenges this paper introduces a novel scheme of malware detection and analysis combining PSD and ML models. It is a very powerful tool to be able to analyze frequency components of signals and to be able to apply it to malware behavior we could potentially find patterns that can be derived fairly far from manual methods of inspecting malware. With the PSD features extracted from malware datasets, we can learn subtle variations in behavior of programs from malware datasets which can help discriminate benign software from malicious programs. These features are then classified using the advanced machine learning algorithms: SVM, Radial Basis Function (RBF) Networks, Multi-Layer Perceptrons (MLP), which are well known for their high generalization capabilities.

This research makes the following key contributions to malware detection:

1. Innovative use of PSD: Introduces PSD as a feature extraction method to capture frequency patterns in malware behavior, enhancing detection of advanced and obfuscated threats.
2. Evaluation of ML models: Compares the performance of SVM, RBF, and MLP classifiers for malware detection, achieving high accuracy rates (RBF: 99%, SVM: 96.9%, MLP: 94.9%).
3. Improved detection accuracy: Demonstrates significant accuracy improvements over traditional signature-based methods, addressing challenges like zero-day and polymorphic malware.
4. Enhanced dynamic analysis: Proposes solutions to anti-sandbox evasion techniques and limited observation time, strengthening dynamic malware detection.
5. Scalable real-time application: Offers a practical and computationally efficient approach for real-time malware detection, adaptable to cybersecurity frameworks.

3. METHODOLOGY

3.1 Dataset description

A dataset of the current study is designed to simulate real conditions of the behavior of Android applications [45], and

its specific and stated goal is to test the malware detection mechanisms. It contains a set of samples, and each sample is marked with the data about permission that can be used by the application and a binary class label, which indicates whether the application is benign or malicious.

A sample is represented as a binary feature vector, and the size of the feature vectors will be the dimensions of a specific permission to Android. Examples of such delicate functions include reading SMS or call logs, accessing the device location, internet access, and manipulating contacts or the camera. Every feature accommodates one or two values: 1 when the respective permission is requested or implemented in the application and 0 when the permission is not in action. This code returns an abridged, readable description of the behavior of each application.

The possible number of features in every application: 512 permissions. Features extracted and to be used for classification: 256 (Selected after applying spectral transformation through power analysis).

Output classes: Binary with two classes: Malicious (Abnormal) labeled as 1. Benign (Normal) labeled as 0.

The labels provided to the classes are derived and determined by expert analysis/ground-truth data, thus making the data set eligible to be used in supervised ML.

Past researchers developed and released an original dataset [45], which concentrated on mobile security and malicious behavior profiling. The results were achieved with the help of the static analysis of Android application package files (APKs) when permission metadata was extracted in the applications manifest files. In this genealogy, the permissions feature is a highly sensitive indicator of an application's behavior because malicious applications tend to have more, or more sensitive permissions as compared to their benign counterparts.

In addition, the data in this experiment consists of both malware and benign samples obtained in this study [45] (e.g., Drebin, Androzoo, custom sandbox environment). Overall, there are X malware samples and Y benign software examples, with each type of sample evenly representing its respective group. The malware set includes a wide range of families [e.g., trojans, ransomware, spyware], allowing the models to generalize over a wide variety of threat types and not be confined to a small subset. In addition, we also sampled dynamic execution traces that captured system calls, memory occupancy, and network activity and converted them into PSD features. Such a representation can extract frequency-domain properties that remain stable even with code obfuscation.

3.2 Preprocessing process

To obtain some consistency and enhance the performance of the models, the preprocessing listed below was carried out:

1. Cleaning: Damaged or non-complete samples were discarded.
2. Normalization: Binary information was stored in 0/1 raw form, where it is appropriate to analyze in the frequency domain.
3. Feature extraction: Power spectral analysis was used to convert the vectors to 256 significant features so as to represent the frequency domain properties. Class balancing (where necessary): We ensured that the data did not exhibit excessive bias towards any particular class.

3.3 Significance and rationale

This dataset is one of the best for malware detection for

several reasons:

1. Substitutability and simplicity: Representation of permissions in binary is the simplest and easiest to understand.
2. Real-world fit to attacks: This feature set is highly relevant for most Android malware that exploits abused permissions to function.
3. Indexing with spectral analysis: The binary input is an excellent form of data, as it can be converted to the frequency domain, where deeper patterns can be obtained using power spectral analysis. Using this dataset, along with extracting spectral features and applying ML classifiers, allows for a strong and scalable way to automatically detect malware on mobile devices.

3.4 Proposed method

This section explains the method used in the study, which has two main steps: extracting features using PSD and classifying them with ML algorithms like SVM, RBF network, and MLP.

The major incentive to the use of PSD is that through it, static permission vectors on the Android platform, which are usually binary and high-dimensional, can be converted to frequency-tomogram-like signals to discover hidden patterns and periodic cycles that are not visibly present in raw data. In treating the permission vectors as a one-dimensional signal, PSD furnishes a spectral decomposition that summarizes the power distribution across the frequency.

The transformation facilitates the detection of minor structural differences between unethical and malicious applications.

Three classification models are trained and evaluated using PSD-transformed features. The SVM is used because it works on high-dimensional data that is non-linearly separable by the ability to use kernel functions.

A specific type of artificial neural network, which is called the RBF network, is selected because of its localized receptive fields and excellent generalization. MLP is a popular DL architecture, and it is used to take advantage of its multilayer structure to capture non-linear, complex decision boundaries.

All in all, the idea of this hybridization is to enhance the strength and correctness of the automated Android malware detection performance by integrating signal-processing algorithms with the state-of-the-art machine learning classifiers.

The approach that has been suggested is depicted in Figure 1, which highlights the primary stages of feature extraction and categorization.

3.4.1 Feature extraction using PSD

The particular data that is analyzed is 512-dimensional binary vectors, with each one representing the profile of permission usage in a certain Android application. Every dimension corresponds to a certain type of permission; its value can be equal to 1 in case that particular application requests the related permission or 0 in case otherwise.

Here high-dimensional and inherently sparse dimensions are used, and such an arrangement poses a challenge to the use of standard pattern-recognition techniques since they are not adept at encoding higher-order relationships.

To overcome such challenges and to increase the feature representational space, the PSD transformation is used. The described technique is a projection of the binary permission vectors into the frequency domain, which makes more notable periodic and structured patterns. The obtained spectrograms allow one to find regularities or deviations when making permissions that may be a symptom of malicious activity, which are often hidden in the temporal or binary dimension.

The PSD of a program is often estimated by using a fast Fourier transform (FFT), which efficiently decomposes a binary vector into a combination of sinusoidal components. The behaviour of signals at different frequencies can be explained by their individual components, and the PSD characterizes the division of signal power based on the variety of frequencies. Since the original signal is real and symmetric in the frequency domain, we only retain the first 256 components of the spectrum.

The same process reduces the dimensionality of the representation by half and also retains the most important spectral data that is needed to perform efficient classification. The feature vectors are 256-dimensional PSD feature vectors representing the frequency-context properties of permission usage and entering as the inputs to numerous machine-learning classifiers.

Using PSD to extract features, the model gains the advantage of being able to pick up on finer, frequency-based aspects of malware that might be overlooked by examining the raw binary vectors themselves, potentially helping improve detection quality as well as resilience.

3.4.2 Classification algorithms

To evaluate the discriminative power of the PSD-based feature representation, this paper uses three commonly accepted ML classifiers: SVM, RBF network, and MLP. The classifiers are selected based on their varying architectures that have passed the test of effectiveness in recognizing patterns. The transformed features in the frequency domain are fed to each classifier to determine the difference between malicious and benign Android applications.

To train and evaluate the classifiers, the dataset was randomly divided using a 70/30 split strategy:

- 70% of the data (420 samples) was used for training,
- 30% of the data (180 samples) was used for testing.

This split was applied consistently across all three

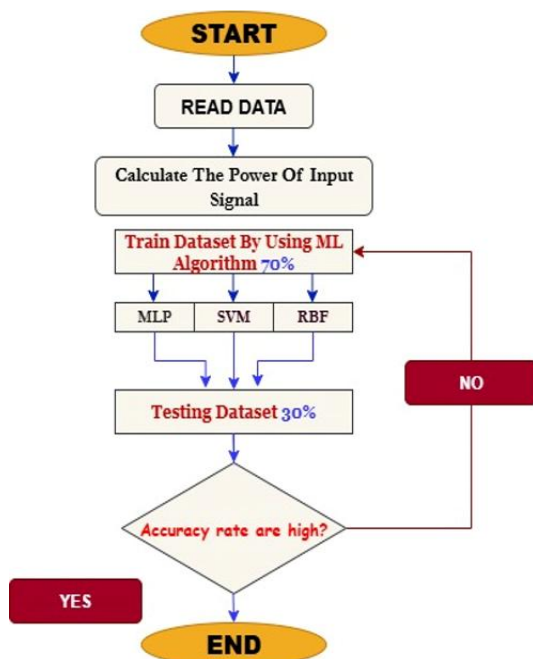


Figure 1. The suggested approach

classifiers—SVM, RBF network, and MLP—with stratified sampling used to maintain the original distribution of both classes in each subset.

3.5 SVM

SVM is a supervised learning model that has often been touted to be robust in a binary classification environment, especially in the event that such data is high-dimensional and non-linearly separable. The main idea underlying the model is to map the decision boundary, i.e., the hyperplane in this case, which will have the greatest margin between the two classes, i.e., malicious binaries and benign applications.

Using 256-dimensional PSD feature vectors as its input, the SVM does the following chain of operations:

- The kernel function maps the features in a higher-dimensional space and helps in distinguishing complicated and non-linear patterns. Radial basis functions or polynomials are some of the kernel functions.
- The support vectors, data points right next to the decision boundary, are found since they are the most important to the determination of the optimal hyperplane.
- The geometric margin between the support vectors of each class (or classifier) is maximized, thereby adding more strength to the capacity for generalization on unseen data.

Particularly, the methodology is particularly well fit in this setting because high-dimensional data and sparse contributions are typical of PSD-transformed binary vectors. Furthermore, the fact that it operates over a small number of support vectors to perform the classification operation means that not only is this a compact model, but it is computationally efficient as well, which makes it an extremely useful feature when finding malware in large-scale schemes.

Given the PSD features in Eq. (1) the SVM aims to:

$$\min_{w,b} 1/2 \|W\|^2 \text{ subject to } y_i (W^T X_i + b) \geq 1 \quad (1)$$

where, X_i are the PSD feature vectors and $y_i \in \{-1,1\}$ are class labels.

Mapping of the dataset to the frequency domain drastically increases the adequacy of the classes, making the classification by the SVMs to work better. In cases where non-linear boundaries may be complex then kernel functions, most conspicuous being the Radial Basis Function kernel could be used to aid non-linear separation.

3.6 RBF network

An RBF network is a specific type of feedforward artificial neural network that works well for difficult problems with complicated and curved decision boundaries. Its canonical form consists of three layers: an input layer, a single hidden layer of radial-basis activation units (typically implemented as Gaussian functions), and an output layer either through a linear regression or logistic regression, making use of the activations of the hidden layer.

RBF networks sort data by transforming input vectors (like the PSD-transformed feature vectors used in this research) into a higher-dimensional space where they create localized bumps. Each of the hidden nodes will evaluate a radial basis function that measures the similarity (or distance) of the input vector and the corresponding prototype (center). The Gaussian kernel can be given by Eq. (2):

$$\phi(x) = \exp\left(-\frac{\|x - c\|^2}{2\sigma^2}\right) \quad (2)$$

where,

- x is the input vector,
- c is the center of the RBF neuron,
- σ controls the width (spread) of the function.

The current study has employed an RBF neural network to distinguish the Android apps as malicious and benign. The frequency-domain patterns encoded in the PSD vectors extracted from each application often cluster in distributions across the feature space, which is a common characteristic. The localized receptive field of the RBF neuron is therefore appropriate to the task.

Training the network entails tuning the RBF centers, the associated widths (sigma), and the output-layer weights in a way that minimizes the prediction error. Such a two-stage learning process allows the network to generalize properly, even in case the separating surface is not regular.

In comparison to their more conventional linear alternatives, RBF networks have a number of potential benefits in the context of malware detection: specifically, their localized response to input can enable the robust performance necessary in the detection of malware when faced with conditions that are likely to confound their more simplistic peers, and since they offer a flexible model of non-linearly mapping input to output without an overabundance of parameters to diffuse or dilute, they are particularly useful in high-dimensional data spaces like those that are typically realized by means of PSD-based malware analysis.

3.7 MLP

In this particular study, a simple feedforward neural network, namely an MLP, is used to classify benign and malicious applications running on the Android platform based on their PSD features using their vector of 256-dimension descriptors. The network contains an input layer and an output layer or more than two layers (two hidden layers). At every layer, the neurons are sequentially layered, and each neuron is networked with the other neurons of the layer, computing a weighted sum of the input and subsequently applying a non-linear activation function to the outputs. Since the MLPs can be used in modeling highly nonlinear and non-linear complex mappings between features and labels in the input and output sets, it is a beneficial classification scheme to use them when dealing with high-dimensional spaces of features.

To optimize the MLP, we implement the backpropagation algorithm, an iterative process of learning named optimization, which looks to minimize the known form of a loss function through small adjustments in the network's weights in a direction of a gradient. The most popular loss used here is the mean squared error (MSE), which measures the average squared difference between the displayed outputs and the respective real labels and is given by the Eq. (3):

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (3)$$

where, y_i is the actual label and \hat{y}_i is the predicted output.

Training of the MLP involves initializing the network's weights, moving the inputs forward through the hidden nodes to produce the outputs, calculating the loss, and then

propagating the error backward to the weights. The depth and the width can be manipulated, as well. The depth is the number of hidden layers, whereas the width is the amount of neurons per layer, so that the compromise between generalization performance and the model complexity is that of the model complexity and generalization performance.

Since it can be used to implement universal approximation, the MLP will be used as the solid baseline in the classification in this present work. The MLP can usually understand complicated patterns and decision boundaries that simpler, linear classifiers often miss when it learns from features derived through principal components.

4. RESULTS AND DISCUSSION

A package of experiments was developed to evaluate the potential of a proposed malware detection framework using a set of machine-learning classifiers. The testing included the comparisons between the classification accuracy, the stability of the models, and the capabilities to distinguish between malicious and harmless Android applications. The experiments utilized the official MATLAB R2018, which served both as a development interface and as an actual application environment for extracting features and training the classifier.

The input files consisted of Android applications in the form of binary permission vectors that contained 512 features. PSD transformation was performed on these vectors to get features in the frequency domain. A bias node was added, giving the classification system 513 nodes as the inputs. The labels given to each application were either malicious (1) or benign (0). This scenario defines a binary classification problem.

The feature vectors, which were improved using the PSD, were used to train and test the classifiers: SVM, RBF network, and MLP. The data was split using the 70/30 method whereby 70% of the data was utilized in training and the rest (30%) was left to test, and it was made sure that the classifiers were trained using a heterogeneous set of data and at the same time retained a separate validation set.

All experiments were repeated several times to reduce the variance coming about as a result of random initialization. The accuracy scores and decision boundaries that have resulted were analyzed systematically to compare the performance of classifiers. The structure and distribution of the input features is illustrated in Figure 2, which allows one to get a sense about how heterogeneous the input space can be before the classification.

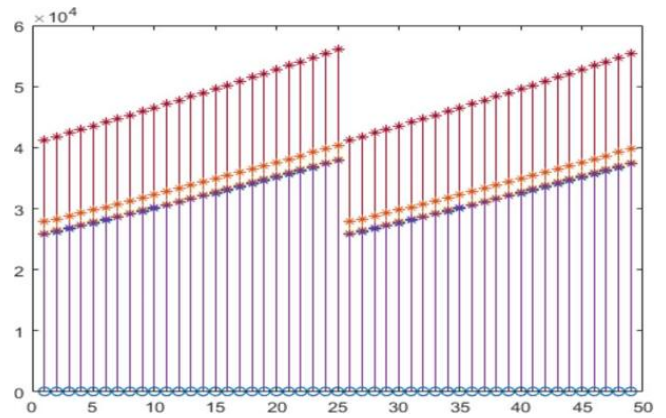


Figure 2. Input feature representation

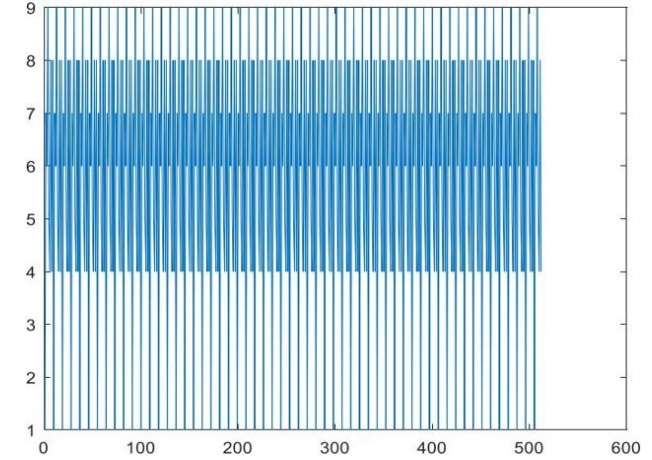


Figure 3. Extracted features from a single sample case

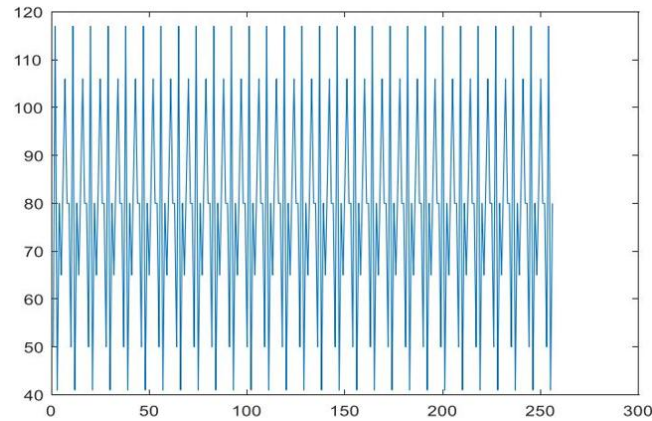


Figure 4. The input data subsequent to feature extraction

The considered dataset contains a set of applications gathered in two of the largest repositories. The data is then subjected to a two-phase feature engineering process: (1) binary feature extraction to capture the application behavior and (2) feature dimensionality reduction by invoking PSD methods.

Figure 3 presents the description of one of the features of the application. The applications are originally characterized by 512 binary permission features, which code the general behavior and request profile of an application.

Operations on PSD are employed to enhance the accuracy of the classification and alleviate the curse of dimensionality. This process results in 256 frequency-domain features derived from the original 512 binary permission features. The resulting spectral components exaggerate the behavioral difference that leads to malware identification shown in Figure 3.

Figure 4 represents the structure of the data after feature extraction. By this step the raw input has been transformed to a numerically organized representation that includes only the most relevant features that are essential to classify it.

This cleaned-up data is the feature for the following ML algorithms, and since it minimizes redundant or noisy variables, it will not only improve the prediction accuracy but also save the machine's work.

In the first experiment, an output of the PSD method was plugged into an RBF neural network. The raw data were changed into frequency-domain features using the PSD technique, which helped identify the main patterns of signals linked to normal and harmful behavior. This classification was done by the RBF network into the normal and attack classes

based on these features extracted.

The RBF network was selected because of the fact it takes care of non-linearity in terms of classification and due to its rapid convergence in the training process. It operates using radial functions as activation functions in the hidden layer, which indicate the similarity between input data and a series of prototype vectors. The RBF network goes through three main steps during training: choosing center points (often done using clustering methods like K-means), deciding how wide the radial functions should be, and adjusting the output weights through supervised training.

Figure 4 displays the training process whereby inputs will be mapped out by the features, followed by the transformations in hidden layers and final output classifications. This experiment acts as a baseline to measure the capability of PSD-based features when combined with a classical RBF-based classifier in classifying between attack and benign patterns.

The results of the experiment portrayed in Figures 5 and 6 indicate that the classifier is 99.0% effective in distinguishing between the attacks and the normal cases. This result shows how well the combination of PSD feature extraction and RBF classifier works.

For the overall evaluation of the classification performance, besides accuracy, other performance measures were also determined to include precision, recall, F1-score, and specificity. All of these measures shed light on the performance of the model in detecting true positives (attacks) and true negatives (normal cases). Table 1 presents the entire series of statistical performance criteria.

In the second experiment, the result of the feature extraction process using the PSD technique was analyzed, and the classification engine of SVM was used. Being a supervised learning algorithm, SVM is most appropriate where the task to be classified is a binary classification problem, and thus, the problem entails maximizing the gap between classes via locating an optimal hyperplane.

Figure 7 shows the classification performance of SVM and gives a confusion matrix that tells the true positives, true negatives, false positives, and false negatives. This matrix forms the foundation on which the ability of the classifier to identify malicious and benign data is analyzed.

Other quantitative metrics, such as accuracy, precision, recall, F1-score, and specificity, were also determined to get additional evaluation of the effectiveness of the classifier (Table 2).

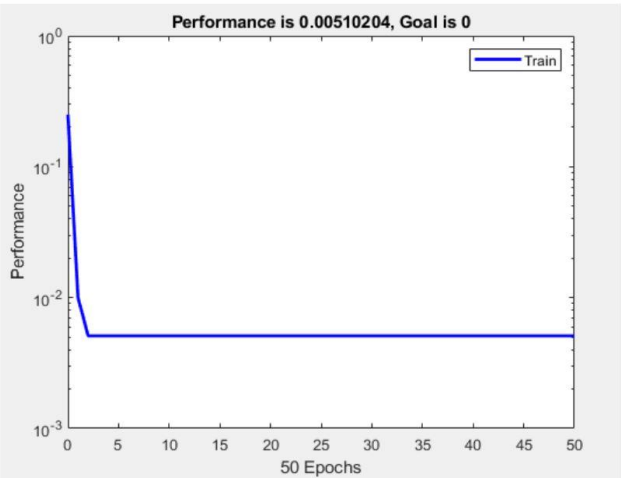


Figure 5. Radial basis function training procedure

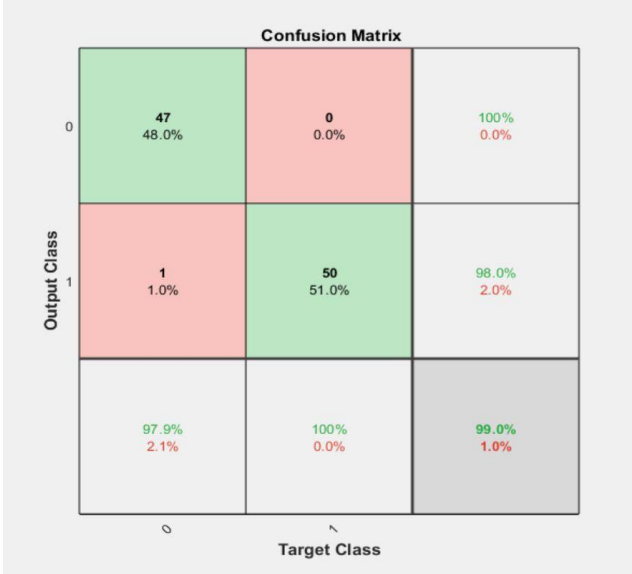


Figure 6. Radial basis function training procedure

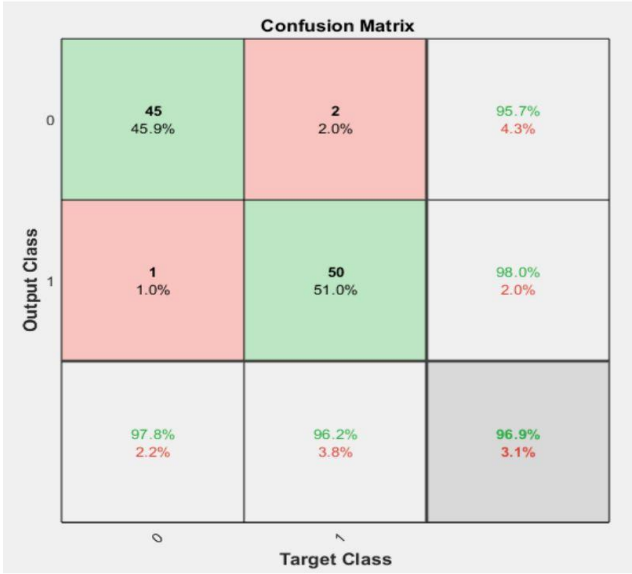


Figure 7. Classification results of SVM in confusion matrix format

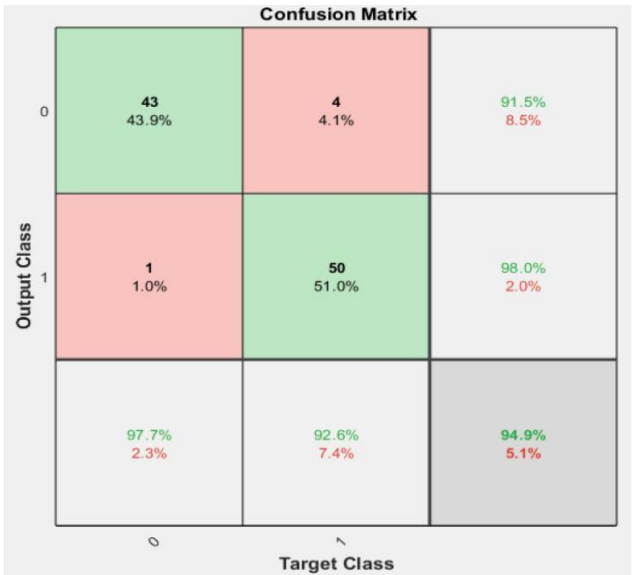


Figure 8. Confusion matrix for neural network classifier

Table 1. Evaluation metrics for RBF model

Accuracy	Sensitivity	Precision	Specificity	F1-Measure
99.0%	97.9%	100%	100%	98.9%

Table 2. Evaluation metrics for SVM model

Accuracy	Sensitivity	Precision	Specificity	F1-Measure
96.9%	97.8%	95.7%	96.0%	96.7%

Table 3. Evaluation metrics for NN model

Accuracy	Sensitivity	Precision	Specificity	F1-Measure
94.9%	97.7%	91.4%	92.3%	94.5%

In the last experimental stage, the features extracted by the PSD feature extraction technique were used as input to a neural network classifier, which had been engineered to distinguish between patterns of attacks and those of normal operations. This neural net utilizes several layers and non-linear activation functions, and, thus, it strives to detect the complex patterns in the data rather than to find simpler classification schemes that can miss them.

Figure 8 presents the confusion matrix concerning the performance of the network, which clearly defines the values of true positive, true negative, false positive, and false negative rates, and, therefore, allows for conducting a deep evaluation of how effectively the network distinguishes between malicious behavior and normal activity.

At the same time, some of the most evident statistical measures of performance have been computed, i.e., accuracy, precision, recall, F1-score, and specificity, to draw a full-scale quantitative portrait of the capabilities of the classifier. These results are provided in Table 3, which provides a quantitative summary of general results of the neural network.

Experimental evidence indicates that all three family members of the classifier, namely RBF, SVM, and MLP, have high detection accuracy, with the highest performance being RBF (99% accuracy) as compared to the SVM (96.9%) and MLP (94.9%) classifiers. This higher performance of the RBF model is due to a number of factors. One of them is that RBF networks can be quite practical in the modelling of non-linear relationships within the feature space and of large dimensions that can accommodate the underlying complexities of the PSD feature representation.

Second, the RBF kernel can create localized decision boundaries surrounding clusters of malware behavior in the frequency domain, thus enabling it to resolve fine grains in the difference between good and bad software that might be collocated in other classifier spaces. In fact, despite its power and capability to handle non-linear separation using kernel functions, SVM may not be as adaptive to localized changes in the spectrum of malware behavior, leading to slightly reduced accuracy.

Although MLP models are undoubtedly incredibly powerful in the settings where they are used in the context of complex pattern modeling, additional data is necessary, and model hyperparameters must be optimized to avoid overfitting or underfitting. The PSD features together with MLP performed well in our experiments, although the RBF model had an

advantage of naturally exploiting local clusters in the PSD space.

Furthermore, the feature extraction performance of PSD is core to the performance improvement of all classifiers. PSD features emphasize repeating patterns of activity by converting behavioral features into the frequency domain and thus are less affected by superficial modifications of code or by obfuscation methods. This leads to a more stable and discriminative set, which improves the capacity of the classifiers to generalize between malware families.

The high recall and low false-positive rates noted with all models further suggest that the framework is adequately represented to detect common and subtle behavioral traces of malware. Overall, the interaction of localized decision boundaries (RBF), frequency-domain feature representation (PSD), and exhaustively trained training data is the reason behind the performance improvements resulting and the reason the RBF model works better than either SVM or MLP in this experiment.

4.1 Comparison with state-of-the-art approaches

A relative comparison was conducted with the aim of assessing the effectiveness of the recommended techniques in the field of malware detection. Three models, namely PSD+RBF, PSD+SVM, and PSD+Neural Network, were devised with frequency-domain features that were derived using PSD. The findings indicate that these models can perform strongly as far as classification is comprehended with regard to attack and normal behavior.

In particular, the PSD+RBF model recorded the highest accuracy score of 99.00%, indicating that it is the most effective at mapping complex and non-linear relationships among the extracted features. The PSD+SVM model had an equal performance compared to the original even though its accuracy was slightly lower; however, there was significant improvement in the generalization ability, and the false positives were low. The PSD+Neural Network classifier provided a middle-ground trade-off between the learning capability and the prediction stability, especially in cases where complex input distributions arise.

Compared to results provided in more recent investigations, the presented framework proves to be notably more effective compared to a range of signature-based and static-analysis methods, most of which are known to perform poorly in most tasks due to inherently low accuracy, the lack of detecting zero-day threats, and manual feature engineering. It was found that the inclusion of PSD at the stage of extracting the features played a major role in enhancing the discriminating power of all classifiers.

The obtained results therefore affirm the affirmation that the frequency-domain analysis and the machine-learning-based classifiers jointly have a substantial and scalable framework that can be used in real-time to detect malware. The effectiveness of the suggested methodology is highlighted through a comparative assessment of how practical it can be and whether it will become a passing alternative to the existing detection systems. A detailed report of all the performance measures is shown in Table 4, where it can be noticed that the PSD+RBF model achieved the best accuracy (99.00%), and it outmatched all other traditional systems in most performance metrics.

Table 4. Performance comparison against recent models

Model	Feature Extractio-N	Classifier	Accuracy (%)	Precision (%)	F1-Measure
[46]	Genetic algorithm-based feature selection	SVM	94.0%	93.8%	93.9%
[47]	Static PE features (n-grams, imports, header fields) + PCA	SVM	97.5%	97.0%	97.2%
[48]	Static Android features (permissions, intents, API calls) + Hybrid FS: wrapper-based RF + greedy stepwise	RF, C5.0 Decision Tree, SVM RBF	DT: 91.8%; RF: 91.32%; SVM RBF: 82.33%	DT: 92.1%; RF: 91.34%; SVM RBF: 81.0%	DT: 91.1%; RF: 91.0%; SVM RBF: 81.1%
[49]	PCA + Min-Max Scaling	LGBM, RF, ET, SVM, KNN, ANN	97.16%	96.5%	96.5%
[50]	Multiple SVMs with feature selection	Multiple SVMs	97.5%	97%	97%
[51]	Feature selection optimized by Particle Swarm Optimization (PSO) applied on ransomware datasets	RF+ SVM, Decision Tree, KNN	98.38%	97.9%	98.0%
Our Methods	PSD+RBF,	RBF,	99.0%	100%	98.9%
	PSD+SVM	SVM	96.9%	95.7%	91.4%
	PSD+NN	NN	94.9%	96.7%	94.5%

5. CONCLUSIONS

The current study introduces an ML methodology for malware identification founded on the PSD tool. Its approach is to analyze malware datasets using multiple autoencoders to extract high-level features and then classify them using SVM, RBF, and multi-MLP. The outcomes of experiments demonstrate a high level of performance, as the general accuracy is 99.00 percent. In this context, SVM comes up with an accuracy of around 96.5, RBF of around 99.0, and MLP of around 94.9. Such results show that the hybrid method works well in extracting discriminatory features and in malware detection when compared to the previous methods.

This is done using the PSD, which finds important features, so the suggested method mixes signal-processing techniques with advanced data-mining algorithms to greatly improve classification accuracy. We later feed the learned features to supervised classifiers, which then learn to differentiate malware. In addition to that, PSD has a satisfactory degree of flexibility and robustness, which supports the possibility of its integration with diverse types of classifiers required to perform different kinds of classifications. This is because compared to the old ways of feature extraction, like convolutional features, PSD has a greater ability to extract the required signal features. Such flexibility allows the approach to be adapted beyond malware detection into areas like EEG and ECG signal classification in the healthcare sector, disease detection, and face and fingerprint biometric recognition systems, among others, with small modifications to the parameters, such as input features, network architecture, and output classes.

In spite of these encouraging findings, a number of weaknesses still exist. To begin with, although the representations of PSD are robust, the approach still makes use of adequate dynamic execution traces, which in some real-world contexts or highly obfuscated malware are not always available. Second, feature extraction and training a model can be computationally challenging and difficult to execute when it is done on a large scale or in real time. Finally, the assessment is performed on a specific dataset; it must be retested on a larger set of unknown malware families and system environments.

In future work, the idea of combining neuron-based feature extraction, namely PSD methods, with deep neural architectures, in particular convolutional neural networks and transformer models, in the extraction of more advanced and hierarchically organized features should be explored, which promises to increase the accuracy of detection and the performance of generalization. A similar next step should be the creation of real-time malware detection systems, which will maximize computational performance and minimize latency, making the strategy viable to implement in a practical environment. The ability to extend the design of the current binary classification to a multi-class would also help in the discrimination of specific malware types and families.

To ensure reliability under attack conditions, we also need to evaluate the strength of the suggested methodology against hackers and malware code obfuscation methods.

Finally, the validity and overall applicability of the PSD-based approach can be proved with its utilization in other areas, including medical signal analysis, speech recognition, and environmental monitoring, and thus the applicability of this approach to a broader application can be displayed and proved.

REFERENCES

- [1] Aslan, Ö.A., Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE Access*, 8: 6249-6271. <https://doi.org/10.1109/ACCESS.2019.2963724>
- [2] Ucci, D., Aniello, L., Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. *Computers & Security*, 81: 123-147. <https://doi.org/10.1016/j.cose.2018.11.001>
- [3] Aboaoja, F.A., Zainal, A., Ghaleb, F.A., Al-Rimy, B.A.S., Eisa, T.A.E., Elnour, A.A.H. (2022). Malware detection issues, challenges, and future directions: A survey. *Applied Sciences*, 12(17): 8482. <https://doi.org/10.3390/app12178482>
- [4] Salih, A., Zeebaree, S.T., Ameen, S., Alkhyat, A., Shukur, H.M. (2021). A survey on the role of artificial intelligence, machine learning and deep learning for cybersecurity attack detection. In 2021 7th International

- Engineering Conference "Research & Innovation amid Global Pandemic"(IEC), Erbil, Iraq, pp. 61-66. <https://doi.org/10.1109/IEC52205.2021.9476132>
- [5] Botacin, M., Ceschin, F., Sun, R., Oliveira, D., Grégio, A. (2021). Challenges and pitfalls in malware research. *Computers & Security*, 106: 102287. <https://doi.org/10.1016/j.cose.2021.102287>
- [6] Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., Liu, H. (2020). A review of android malware detection approaches based on machine learning. *IEEE Access*, 8: 124579-124607. <https://doi.org/10.1109/ACCESS.2020.3006143>
- [7] Alsumaidae, Y.A.M. (2024). Advanced transfer learning technique for enhanced detection and classification of damaged solar cells. *Ingénierie des Systèmes d'Information*, 29(6): 2335-2343. <https://doi.org/10.18280/isi.290621>
- [8] Kurnaz, S., Ahmed, Y. (2019). Automatic malware detection using data mining techniques based on Power Spectral Density (PSD). *International Journal of Computer Science and Mobile Computing*, 8(3): 27-30. <https://www.academia.edu/download/106675787/V8I3201906.pdf>
- [9] Easa, H.K., Saber, A.A., Hamid, N.K., Saber, H.A. (2023). Machine learning based approach for detection of fake banknotes using support vector machine. *Indonesian Journal of Electrical Engineering and Computer Science*, 31(2): 1016. <https://doi.org/10.11591/ijeecs.v31.i2>
- [10] Akhtar, M.S., Feng, T. (2022). Malware analysis and detection using machine learning algorithms. *Symmetry*, 14(11): 2304. <https://doi.org/10.3390/sym14112304>
- [11] Shaukat, K., Luo, S., Varadharajan, V. (2023). A novel deep learning-based approach for malware detection. *Engineering Applications of Artificial Intelligence*, 122: 106030. <https://doi.org/10.1016/j.engappai.2023.106030>
- [12] Mahindru, A., Sangal, A.L. (2021). MLDroid—framework for Android malware detection using machine learning techniques. *Neural Computing and Applications*, 33(10): 5183-5240. <https://doi.org/10.1007/s00521-020-05309-4>
- [13] Qamar, A., Karim, A., Chang, V. (2019). Mobile malware attacks: Review, taxonomy & future directions. *Future Generation Computer Systems*, 97: 887-909. <https://doi.org/10.1016/j.future.2019.03.007>
- [14] Faruk, M.J.H., Shahriar, H., Valero, M., Barsha, F.L., Sobhan, S., Khan, M.A., Wu, F. (2021). Malware detection and prevention using artificial intelligence techniques. In 2021 IEEE international conference on big data, Orlando, FL, USA, pp. 5369-5377. <https://doi.org/10.1109/BigData52589.2021.9671434>
- [15] Al-Tahee, M., Easa, H.K., Jabbar, K.A., Hussein, L., Alataba, S.R., Mohammed, M.A. (2024). Artificial intelligence assisted cyber-Physical systems with intelligent cyber security using deep learning. In 2024 International Conference on Smart Systems for Electrical, Electronics, Communication and Computer Engineering (ICSSEECC), Coimbatore, India, pp. 689-694. <https://doi.org/10.1109/ICSSEECC61126.2024.10649428>
- [16] Hadiprakoso, R.B., Kabetta, H., Buana, I.K.S. (2020). Hybrid-based malware analysis for effective and efficiency android malware detection. In 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), Jakarta, Indonesia, pp. 8-12. <https://doi.org/10.1109/ICIMCIS51567.2020.9354315>
- [17] Ahmed, S.R., Mohamed, S.J., Aljanabi, M.S., Algburi, S., Majeed, D.A., Kurdi, N.A., Tawfeq, J.F. (2024). A novel approach to malware detection using machine learning and image processing. In *Proceedings of the Cognitive Models and Artificial Intelligence Conference*, pp. 298-302. <https://doi.org/10.1145/3660853.3660931>
- [18] Hussain, S.J., Ahmed, U., Liaquat, H., Mir, S., Jhanjhi, N.Z., Humayun, M. (2019). IMIAD: Intelligent malware identification for android platform. In 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, pp. 1-6. <https://doi.org/10.1109/ICCISci.2019.8716471>
- [19] Chaganti, R., Ravi, V., Pham, T.D. (2022). Image-based malware representation approach with EfficientNet convolutional neural networks for effective malware classification. *Journal of Information Security and Applications*, 69: 103306. <https://doi.org/10.1016/j.jisa.2022.103306>
- [20] Akhtar, M.S., Feng, T. (2023). Evaluation of machine learning algorithms for malware detection. *Sensors*, 23(2): 946. <https://doi.org/10.3390/s23020946>
- [21] Manzil, H.H.R., Naik, S.M. (2024). Detection approaches for android malware: Taxonomy and review analysis. *Expert Systems with Applications*, 238: 122255. <https://doi.org/10.1016/j.eswa.2023.122255>
- [22] Moser, A., Kruegel, C., Kirda, E. (2007). Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pp. 421-430. <https://doi.org/10.1109/ACSAC.2007.21>
- [23] Sharif, M., Yegneswaran, V., Saidi, H., Porras, P., Lee, W. (2008). Eureka: A framework for enabling static malware analysis. In *European Symposium on Research in Computer Security*, pp. 481-500. https://doi.org/10.1007/978-3-540-88313-5_31
- [24] Sung, A.H., Xu, J., Chavez, P., Mukkamala, S. (2004). Static analyzer of vicious executables (save). In *20th Annual Computer Security Applications Conference*, Tucson, AZ, USA, pp. 326-334. <https://doi.org/10.1109/CSAC.2004.37>
- [25] Wagener, G., State, R., Dulaunoy, A. (2007). Malware behaviour analysis, extended version. *Journal in Computer Virology*, 4(4): 279-287. <https://doi.org/10.1007/s11416-007-0074-9>
- [26] Vamvatsikos, D., Cornell, C.A. (2002). Incremental dynamic analysis. *Earthquake Engineering & Structural Dynamics*, 31(3): 491-514. <https://doi.org/10.1002/eqe.141>
- [27] Alsumaidae, Y.A.M., Yahya, M.M., Yaseen, A.H. (2025). Optimizing malware detection and classification in real-time using hybrid deep learning approaches. *International Journal of Safety & Security Engineering*, 15(1): 141-150. <https://doi.org/10.18280/ijss.150115>
- [28] Ibrahim, M., Issa, B., Jasser, M.B. (2022). A method for automatic android malware detection based on static analysis and deep learning. *IEEE Access*, 10: 117334-117352. <https://doi.org/10.1109/ACCESS.2022.3219047>
- [29] Mohamad Arif, J., Ab Razak, M.F., Awang, S., Tuan Mat, S.R., Ismail, N.S.N., Firdaus, A. (2021). A static

- analysis approach for Android permission-based malware detection systems. *PloS One*, 16(9): e0257968. <https://doi.org/10.1371/journal.pone.0257968>
- [30] Sandeep, H.R. (2019). Static analysis of android malware detection using deep learning. In 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, pp. 841-845. <https://doi.org/10.1109/ICCS45141.2019.9065765>
- [31] Molina-Coronado, B., Mori, U., Mendiburu, A., Miguel-Alonso, J. (2023). Towards a fair comparison and realistic evaluation framework of android malware detectors based on static analysis and machine learning. *Computers & Security*, 124: 102996. <https://doi.org/10.1016/j.cose.2022.102996>
- [32] Shatnawi, A.S., Yassen, Q., Yateem, A. (2022). An android malware detection approach based on static feature analysis using machine learning algorithms. *Procedia Computer Science*, 201: 653-658. <https://doi.org/10.1016/j.procs.2022.03.086>
- [33] Raghuraman, C., Suresh, S., Shivshankar, S., Chapaneri, R. (2019). Static and dynamic malware analysis using machine learning. In First International Conference on Sustainable Technologies for Computational Intelligence: Proceedings of ICTSCI 2019, pp. 793-806. https://doi.org/10.1007/978-981-15-0029-9_62
- [34] Amer, E., Zelinka, I. (2020). A dynamic windows malware detection and prediction method based on contextual understanding of API call sequence. *Computers & Security*, 92: 101760. <https://doi.org/10.1016/j.cose.2020.101760>
- [35] Sihwail, R., Omar, K., Zainol Ariffin, K.A., Al Afghani, S. (2019). Malware detection approach based on artifacts in memory image and dynamic analysis. *Applied Sciences*, 9(18): 3680. <https://doi.org/10.3390/app9183680>
- [36] Hwang, J., Kim, J., Lee, S., Kim, K. (2020). Two-stage ransomware detection using dynamic analysis and machine learning techniques. *Wireless Personal Communications*, 112(4): 2597-2609. <https://doi.org/10.1007/s11277-020-07166-9>
- [37] da Costa, F.H., Medeiros, I., Menezes, T., da Silva, J.V., da Silva, I.L., Bonifácio, R., Ribeiro, M. (2022). Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification. *Journal of Systems and Software*, 183: 111092. <https://doi.org/10.1016/j.jss.2021.111092>
- [38] Jeon, J., Park, J.H., Jeong, Y.S. (2020). Dynamic analysis for IoT malware detection with convolution neural network model. *IEEE Access*, 8: 96899-96911. <https://doi.org/10.1109/ACCESS.2020.2995887>
- [39] Casso, I., Morales, J.F., López-García, P., Hermenegildo, M.V. (2020). Testing your (static analysis) truths. In International Symposium on Logic-Based Program Synthesis and Transformation, pp. 271-292. https://doi.org/10.1007/978-3-030-68446-4_14
- [40] Menard, K.P., Menard, N. (2020). *Dynamic Mechanical Analysis*. CRC Press. <https://doi.org/10.1201/9780429190308>
- [41] Guo, Z., Tan, T., Liu, S., Liu, X., Lai, W., Yang, Y., Zhou, Y. (2023). Mitigating false positive static analysis warnings: Progress, challenges, and opportunities. *IEEE Transactions on Software Engineering*, 49(12): 5154-5188. <https://doi.org/10.1109/TSE.2023.3329667>
- [42] Thomson, P. (2021). Static analysis: An introduction: The fundamental challenge of software engineering is one of complexity. *Queue*, 19(4): 29-41. <https://dl.acm.org/doi/pdf/10.1145/3487019.3487021>
- [43] Afianian, A., Niksefat, S., Sadeghiyan, B., Baptiste, D. (2019). Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys*, 52(6): 1-28. <https://doi.org/10.1145/3365001>
- [44] Cui, Y., Sun, Y., Lin, Z. (2023). DroidHook: A novel API-hook based Android malware dynamic analysis sandbox. *Automated Software Engineering*, 30(1): 10. <https://doi.org/10.1007/s10515-023-00378-w>
- [45] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.E.R.T. (2014). Drebin: Effective and explainable detection of android malware in your pocket. In NDSS, 14(1): 23-26. https://media.telefonicatech.com/telefonicatech/uploads/2021/1/4915_2014-ndss.pdf
- [46] Fatima, A., Maurya, R., Dutta, M.K., Burget, R., Masek, J. (2019). Android malware detection using genetic algorithm based optimized feature selection and machine learning. In 2019 42nd International Conference on Telecommunications and Signal Processing (TSP), Budapest, Hungary, pp. 220-223. <https://doi.org/10.1109/TSP.2019.8769039>
- [47] Al-Kasassbeh, M., Mohammed, S., Alauthman, M., Almomani, A. (2020). Feature selection using a machine learning to classify a malware. In Handbook of Computer Networks and Cyber Security: Principles and Paradigms, pp. 889-904. https://doi.org/10.1007/978-3-030-22277-2_36
- [48] Smmarwar, S.K., Gupta, G.P., Kumar, S. (2022). A hybrid feature selection approach-based Android malware detection framework using machine learning techniques. In Cyber Security, Privacy and Networking: Proceedings of ICSPN 2021, pp. 347-356. https://doi.org/10.1007/978-981-16-8664-1_30
- [49] Hasan, R., Biswas, B., Samiun, M., Saleh, M.A., Prabha, M., Akter, J., Abdullah, M. (2025). Enhancing malware detection with feature selection and scaling techniques using machine learning models. *Scientific Reports*, 15(1): 9122. <https://doi.org/10.1038/s41598-025-93447-x>
- [50] Turukmane, A.V., Devendiran, R. (2024). M-MultiSVM: An efficient feature selection assisted network intrusion detection system using machine learning. *Computers & Security*, 137: 103587. <https://doi.org/10.1016/j.cose.2023.103587>
- [51] Gurukala, N.K.Y., Verma, D.K. (2024). Feature selection using particle swarm optimization and ensemble-based machine learning models for ransomware detection. *SN Computer Science*, 5(8): 1093. <https://doi.org/10.1007/s42979-024-03454-4>