INTERNATIONAL JOURNAL OF INFORMATION SECURITY SCIENCE
A. Karakaya et al., Vol.13, No.4, pp.1-20
https://doi.org/10.55859/ijiss.1510423

Dynamic Malware Detection Approach Based on API Calls: Machine Learning and Ensemble Learning Models

Aykut Karakaya[1], Ahmet Ulu[2]

[1]Department of Computer Technologies, Bülent Ecevit University, Zonguldak, Türkiye
[2] Department of Computer Engineering, Çoruh University, Artvin, Türkiye
Corresponding Author: aykut.karakaya@beun.edu.tr

Research Paper Received: 04.07.2024 Revised: 17.09.2024 Accepted: 03.10.2024

Abstract—The rapid evolution of malware presents significant challenges in cybersecurity. The malware can be detected by using static, dynamic, or hybrid features. Static features are effective for the detection of malware but they are unable to prevent code obfuscation approaches. On the other hand, dynamic features contain code or characters of malware that are obtained while the software is operating on a device. In comparison to static analysis, dynamic analysis is a better method especially to detect the malware containing code obfuscation. The dynamic datasets, contain API calls and permissions, enabling real-time monitoring of malware behavior. Since these datasets are obtained through behavior analysis during execution, methods using dynamic datasets offer a more realistic approach compared to those using static datasets. This study investigates the efficacy of various machine learning and ensemble learning models for malware detection using dynamic analysis. For this purpose, it is used the VirusSample and VirusShare datasets, which consist of API calls and permissions. For both datasets, the Random Forest (RF) model used Principal Component Analysis (PCA) for feature extraction achieved the best results among the machine learning models, with accuracies of 94.83% and 86.27%, respectively. For the VirusSample dataset, the stacking ensemble learning model, which uses RF and Decision Trees (DT) as base classifiers and K-Nearest Neighbors (k-NN) as the meta classifier, achieved the highest accuracy of 94.56% thanks to the use of PCA. In contrast, for the VirusShare dataset, the stacking ensemble learning model, which uses RF, k-NN, and Gradient Boosting (GB) as base classifiers and support vector machine (SVM) as the meta classifier, achieved the highest accuracy of 86.21% with PCA usage for feature extraction. These results underscore the superiority of dynamic analysis and the effectiveness of ensemble methods in enhancing malware detection accuracy. This study contributes to the optimization of machine learning models and the advancement of cybersecurity solutions.

Keywords—information security, malware detection, dynamic analysis, machine learning, ensemble learning.

1. Introduction

With the rapid increase in digitalization today, cybersecurity threats are diversifying and becoming more complex at the same rate. Malware is at the forefront of these threats and can cause damage across a wide spectrum, from individuals to organizational structures. Detecting and classifying malware is critically important in the field of cybersecurity. In this context, machine learning techniques have become a significant tool for malware detection.

Datasets used for malware detection are gener-

ally divided into two main categories: static and dynamic. Static analysis aims to detect malware by examining its code structure, while dynamic analysis analyzes the behavior of the software during its execution [1]. Dynamic datasets provide more comprehensive and reliable results by examining the real-time activities of malware and their impact on the system.

Dynamic analysis has gained prominence due to its ability to capture the behavioral characteristics of malware through API calls, system calls, and other runtime activities, offering a more comprehensive and realistic assessment of threats compared to static analysis [2]. The real-time insights gained from dynamic datasets, which include API calls and permissions, make it possible to monitor malware behavior continuously, thus providing an effective countermeasure against evolving threats. This study aims to address the challenges of malware detection using dynamic analysis by exploring various machine learning and ensemble learning models.

Recent research has demonstrated the efficacy of dynamic analysis in enhancing malware detection accuracy. For example, Rosmansyah et al. proposed a model that integrates API call sequences with traditional machine-learning algorithms [3]. Similarly, Roy et al. introduced a model that reduces feature sets for fast detection and is robust against code obfuscation [4]. They collected a new dataset and proved that a malware detection method using API calls still gives high-accuracy result against code obfuscation. Similarly, Shen et al. employed N-gram features derived from control flow graphs alongside machine learning techniques to identify malware [5]. These studies highlight the potential of dynamic analysis combined with advanced machine learning techniques to effectively identify and mitigate malware threats.

Despite the progress in dynamic analysis, the literature still shows a gap in optimizing machine learning models for malware detection using dynamic datasets. Many existing studies focus on static features, with fewer exploring the full potential of dynamic data. This paper seeks to fill this gap by evaluating the performance of various machine learning and ensemble learning models, including RF, SVM, and stacking methods, on dynamic datasets derived from API calls and permissions. The study aims to provide a detailed comparison of these models and to highlight the benefits of incorporating feature extraction techniques such as PCA to enhance model performance.

1.1. Related works

Research on malware detection and classification constitutes a significant area of investigation in the field of cybersecurity. Studies on malware classification using dynamic datasets have garnered increasing interest in recent years. This section reviews important studies in the literature related to malware detection and classification.

In [6], a method called CTIMD was proposed for dynamic malware detection. This method uses deep learning techniques that integrate threat intelligence obtained from CTIs with API call sequences and runtime parameters. Experiments revealed that CTIMD performed 4.0% to 41.3% better in F1-score compared to existing methods based on raw API call sequences, and 1.2% to 6.5% better compared to state-of-the-art methods considering both API calls and runtime parameters.

In [7], a model named Mal-ASSF was developed for dynamic malware analysis. This model used API2Vec, BiLSTM, TextRNN, and an attention mechanism. As a result, it achieved 3-5% higher accuracy compared to existing methods.

In [8], a similarity-based hybrid malware detec-

tion model called HAPI-MDM was developed. The model was trained using both static and dynamic analysis of API calls with XGBoost and Artificial Neural Network (ANN) algorithms. This model demonstrated superior performance with an accuracy rate of 97.91% and achieved the lowest false positive and false negative rates.

In [9], a new deep learning framework has been developed for dynamic malware detection by utilizing the intrinsic properties of API call sequences. The model uses API call embeddings and convolutional layers to extract semantic information representing categories, actions, and process objects, as well as Bi-LSTM modules to capture the relationships between APIs. As a result, the model achieved 97.31% accuracy and 97.24% F1-score on a large dataset, outperforming existing methods.

In [10], a new framework called DMalNet for dynamic malware analysis based on API feature engineering and graph learning has been developed. It employs a hybrid feature encoder to extract semantic features from API names and arguments, a method to derive an API call graph from the API call sequence, and a graph neural network (GNN) to learn the content and structural information of this graph. This framework achieved 98.43% accuracy in malware detection and 91.42% accuracy in malware type classification.

In [11], the evaluation of supervised machine learning algorithms for malware detection using dynamic API calls has been conducted. Malware samples were executed through Cuckoo Sandbox to collect dynamic API calls, and these data were used to train supervised machine learning algorithms such as Naive Bayes, k-NN, SVM, DT, RF, and Adaboost. As a result of these evaluations, it was noted that the RF algorithm showed the highest performance with an accuracy rate of 99.1%.

In [12], a new method based on permission completion and API calls has been developed to detect Android malware. The feature extraction method using permission completion and API calls, combined with a RF classifier, was employed to detect applications hiding malicious behaviors through the dynamic code loading mechanism. This model achieved an overall accuracy rate of 99.885% on the general dataset and demonstrated high performance in detecting malware using the dynamic code loading mechanism.

Tang et al. proposed a novel deep learning-based method for detecting Android malware [13]. In the proposed method, the executable file of the target malware is read in 8-bit segments and converted into 256x256 grayscale and Markov images. Subsequently, the grayscale and Markov images are combined using transfer probabilities to create a mixed feature image. They have used a ResNet-based network and the network depth is further enhanced by incorporating a channel and spatial attention mechanism (CBAM[14]).

Zhu et al. proposed a new Convolutional Neural Network (CNN) network named MserNetDroid [15]. The proposed MserNetDroid network consists of two different structures. The first is a residual module where the SENet attention mechanism[16], a channel-based focus mechanism, is applied, and the MserNet module is used for multi-level feature extraction. This approach allows for deeper learning of intrinsic features, which are then weighted using a focus mechanism. To evaluate the effectiveness of the proposed framework, 2,126 malicious applications and 1,061 benign applications collected from VirusShare and the Google Play Store were analyzed. The evaluation results demonstrate that the proposed model successfully detects malware with an accuracy rate of 96.48%.

Ksibi et al. proposed a new CNN model based on an imaginary feature for the detection of Android

malware [17]. They used EfficientNet-based deep learning techniques with limited training data [18]. The method involves converting Android APK files (XML and ARSC) into binary codes and RGB images, which are then used as inputs for deep learning models. The authors extracted features from pre-trained DenseNet169, Xception, InceptionV3, ResNet50, and VGG16 models (by adding them to the beginning of the CNN network) and then adds a 5-layer CNN network containing convolution, pooling, and dropout layers afterward. The proposed methods were evaluated using achieving accuracy rates of 95.24%, 95.24%, and 95.83%. These results demonstrate better performance with lower resource consumption without manual feature engineering.

As a result, there is a vast research area in the literature concerning the detection and classification of malicious software. Studies employing both machine learning and deep learning techniques are presented. Similar to them, we have presented a malware detection method that uses machine learning and feature extraction. Studies using dynamic datasets emphasize the importance of analyzing real-time behaviors of malicious software and developing effective solutions against such threats. This paper contributes to the proliferation of dynamic API call-based datasets and machine-learning models in the field of cybersecurity.

1.2. Contributions

This paper comprehensively evaluates the performance of various machine learning and ensemble learning methods using dynamic analysis data and datasets containing API calls for malware detection. In the existing literature, the number of studies that detect malware using dynamic datasets is limited especially compared studies using static features [19]. In this context, our study makes significant contributions to the literature by presenting new approaches to malware detection and developing models that demonstrate superior performance compared to existing methods. The contributions of this study are as follows:

- *Use of API calls for dynamic malware detection:* This is one of the few studies in the literature that evaluates the performance of various machine learning and ensemble learning methods using datasets containing API calls for dynamic malware detection.
- *Comparison of different machine learning models:* Supervised machine learning algorithms such as Naive Bayes, k-NN, SVM, DT, RF, and Adaboost have been tested on dynamic datasets using API call sequences.
- *Evaluation of ensemble learning methods:* Stacking methods combining various models such as RF, k-NN, and GB have been used, and the performance of these approaches has been compared.
- *Development of high-performance models:* Models obtained with the RF method and using PCA as feature extraction have shown higher accuracy, precision, recall, and F1-score than other machine learning models.
- *High performance on general datasets:* The results obtained emphasize the effectiveness of dynamic analysis methods and selected machine learning models, especially on large and complex datasets.
- *Optimization of machine learning models:* Significant contributions are made to the field of cybersecurity in terms of the use of dynamic datasets and the optimization of machine learning models.

These contributions include the basic points that emphasize the place and importance of our paper in the literature.

1.3. Organization

This paper is organized as follows: Section 2 gives a detailed explanation of machine learning methods and the PCA used for feature extraction while Section 3 describes the features of the datasets, the proposed methods, and the experimental design, including details on data preprocessing and ensemble learning models. Section 4 includes a sensitivity analysis of the highest-performing models, a discussion of the experimental results, a comparison of the results, and potential future research directions. Finally, Section 5 provides a summary of the main findings and contributions, and concludes the paper.

2. Preliminaries

In this section, the machine-learning methods used for malware detection are presented in detail.

2.1. PCA

PCA, originally developed as a statistical technique to transform a set of correlated variables into a new set of independent and uncorrelated variables known as principal components (PCs), is widely employed in data analysis. These components are linear combinations of the initial variables. PCA is particularly effective in addressing multicollinearity among the variables [20]. The PCs are ordered such that the first PC accounts for the most variance in the data, with each following PC explaining the next largest portion of variance not captured by the previous ones, beginning with the first [21]. To clarify the influence of each original variable on the PCs, rotational methods like varimax rotation are often used. Varimax rotation helps each variable to be strongly associated with only one PC while minimizing its correlation with the others.

Algorithm 1 PCA

Require: Data matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ with n samples and p features
Ensure: Reduced dimensionality data matrix $\mathbf{Z} \in \mathbb{R}^{n \times k}$
Standardize the Data:
Compute the mean of each feature: $\mu = \frac{1}{n} \sum_{i=1}^{n} \mathbf{X}_i$
Center the data by subtracting the mean: $\mathbf{X}_{\text{centered}} = \mathbf{X} - \mathbf{1}\mu^T$
Compute the Covariance Matrix:
$\mathbf{C} = \frac{1}{n-1} \mathbf{X}_{\text{centered}}^T \mathbf{X}_{\text{centered}}$
Eigenvalue Decomposition:
Compute the eigenvalues and eigenvectors of the covariance matrix $\mathbf{C}$:
$\mathbf{CV} = \mathbf{V\Lambda}$ ▷ $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues, $\mathbf{V}$ contains the eigenvectors
Select Principal Components:
Sort the eigenvalues in descending order and select the top k eigenvectors:
$\mathbf{W} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k]$
Project the Data:
Project the data onto the new k-dimensional space:
$\mathbf{Z} = \mathbf{X}_{\text{centered}} \mathbf{W}$
return $\mathbf{Z}$

The key outputs of PCA, known as factor loadings, show the degree to which a variable contributes to a specific PC and its similarity to other variables. Variables with higher loadings have a greater impact on the variance explained by that PC. In practical applications, only loadings with absolute values greater than 0.5 are typically used for interpreting the PCs [22]. Furthermore, a PC is generally considered statistically significant if it has an eigenvalue of 1 or higher. The steps of the PCA are presented in detail in Algorithm 1.

2.2. RF

Breiman first introduced the RF algorithm, which has since become a widely used nonparametric tool for classification and regression. It enables the development of prediction rules using various predictor variables without requiring assumptions about their relationships with the response variable [23]. RF is effective for both classification and regression by combining the outputs of multiple

DT into a single prediction. This ensemble learning approach reduces overfitting in DTs.

Tree-based models iteratively split the dataset until a predefined stopping criterion is met. Depending on the criteria, DT can be applied to both classification and regression tasks. However, a common drawback of DT is their tendency to overfit, leading to reduced accuracy. To improve generalization, multiple trees can be constructed from different subsets of the data. Ho introduced the random-subspace method, which Breiman expanded into the random forest algorithm [24]. RFs use an ensemble of trees built on bootstrap samples to mitigate overfitting, as outlined in Algorithm 2.

Algorithm 2 RF algorithm

Initialization: Training data (D), subtrees (B)
for $i \leftarrow 1$ **to** B **do**
    Draw a bootstrap sample of size N from D
    while node size $\neq$ minimum node size **do**
        Randomly select a subset of m predictor variables from total p
        for $j \leftarrow 1$ **to** m **do**
            if jth predictor optimizes splitting criterion **then**
                Split internal node into two child nodes
                break
            end if
        end for
    end while
end for
Output: The ensemble tree of all B subtrees is created.

RFs, which aggregate DT, typically outperform individual trees and offer more accurate error estimates. As the number of trees increases, the error rate decreases. The size of the subset of predictor variables, m, is crucial for controlling tree depth and should be adjusted during model selection.

2.3. DT

A DT is a non-parametric supervised learning method used for classification and regression [25]. It partitions the data into subsets based on the most significant attribute values, using a tree-like model of decisions. The decision-making process involves selecting the attribute that maximizes the Information Gain (IG) or minimizes the Gini Impurity. For classification, the Information Gain is calculated as:

$$IG(T, A) = E(T) - \sum_{v \in Values(A)} \frac{|T_v|}{|T|} \times E(T_v) \quad (1)$$

where T is the entire dataset, E is the entropy, A is the attribute being considered, T_v is the subset of T for which attribute A has value v, and $E(T)$ is defined as:

$$E(T) = -\sum_{i=1}^{n} p_i \log_2(p_i) \quad (2)$$

Here, p_i is the proportion of examples in class i. The Decision Tree continues to split the data until it reaches a stopping criterion, such as maximum depth or minimum number of samples per leaf.

2.4. SVM

SVM is a supervised learning algorithm used for both classification and regression tasks [26]. The core idea of SVM is to find the hyperplane that best separates the data into classes. For a linearly separable dataset, the SVM aims to maximize the margin between the closest points of the two classes, known as support vectors. Mathematically, the optimization problem for a linear SVM can be formulated as:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad (3)$$

subject to the constraint:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \forall i \quad (4)$$

where $\mathbf{w}$ is the weight vector, b is the bias term, $y_i \in \{-1, 1\}$ represents the class labels, and $\mathbf{x}_i$ are the feature vectors. The decision function is then given by:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{5}$$

For non-linearly separable data, the kernel trick is employed to project the data into a higher-dimensional space where a linear separation is possible.

2.5. k-NN

In k-NN classification, the distance between each data point in the dataset is calculated. However, only the k closest points are considered when determining the classification of a given data point. These k points are the ones nearest to the point in question compared to all other points. The value of k is predetermined; selecting a k that is too large can result in different data being incorrectly grouped into the same class, while a k that is too small can cause similar data to be classified into different groups.

Algorithm 3 k-NN Algorithm

1: **Initialization:** Training data (X); class labels (Y); number of nearest neighbors (K)
2: **for** each sample x in the test data **do**
3: Calculate the distance: $d(x, X) = \sqrt{\sum_{i=1}^{n}(x_i - X_i)^2}$
4: Classify x based on the majority class: $C(x_i) = \arg\max_k \sum_{X_i \in k-NN} C(X_j, Y_K)$
5: **end for**
6: **Output:** Class of a test sample x

The k-NN algorithm works by first determining a method for measuring distance, with the Euclidean distance being the most commonly used. The k nearest neighbors are identified, and the class with the majority representation among these neighbors is assigned to the test point. The general structure of the algorithm is outlined in Algorithm 3. The performance of the k-NN classifier is highly dependent on the value of k [27]. Typically, the optimal k is found through empirical testing.

2.6. GB

GB is an ensemble learning technique that builds models sequentially, where each model attempts to correct the errors of the previous one [28]. The algorithm optimizes a loss function by adding new models in a greedy manner. The general objective of GB can be expressed as:

$$F_m(x) = F_{m-1}(x) + \alpha h_m(x) \tag{6}$$

where $F_m(x)$ is the updated model, $F_{m-1}(x)$ is the previous model, α is the learning rate, and $h_m(x)$ is the new model that minimizes the loss function. The loss function $L(y, F_m(x))$ is minimized using gradient descent:

$$h_m(x) = -\gamma \nabla_{F_{m-1}} L(y, F_{m-1}(x)) \tag{7}$$

where γ is the step size and $\nabla_{F_{m-1}} L(y, F_{m-1}(x))$ is the gradient of the loss function with respect to the predictions of the previous model. GB can be used for both classification and regression tasks, with the loss function tailored to the specific problem (e.g., mean squared error for regression, log loss for classification).

2.7. Adaptive Boosting (Adaboost)

Ensemble learning encompasses three main approaches: bagging, stacking, and boosting. In bagging, the dataset is divided into training and testing sets, and random samples from the training data are used to train multiple models. The final decision is

made by averaging or voting on the model outputs. Boosting also involves data splitting and sampling, but it prioritizes misclassified data for subsequent models, thereby focusing on difficult cases. Stacking, on the other hand, combines the outputs of several classifiers using a meta-classifier to make the final decision, capitalizing on the strengths of each model in the feature space [29].

AdaBoost is a fundamental boosting algorithm that increases the influence of misclassified data by adjusting their weights in successive iterations. Unlike other methods that use deep DT, AdaBoost typically employs shallow trees, with final decisions made through weighted majority voting. This technique is particularly effective in scenarios requiring improved accuracy and reduced overfitting, such as in fog computing and IoT-based health and tactical analysis frameworks [30], [31]. The steps of the AdaBoost method are presented in detail in Algorithm 4.

Algorithm 4 AdaBoost

Require: Training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$ where $y_i \in \{-1, 1\}$
Require: Number of boosting rounds T
Ensure: Final strong classifier $H(\mathbf{x})$
Initialize weights:
$w_i^{(1)} = \frac{1}{n}, \quad \forall i = 1, 2, \ldots, n$
for $t = 1$ to T **do**
  Train weak classifier $h_t(\mathbf{x})$ using weights $w_i^{(t)}$
  Compute the weighted error:

$$\epsilon_t = \sum_{i=1}^{n} w_i^{(t)} \cdot \mathbf{I}(h_t(\mathbf{x}_i) \neq y_i)$$

  Compute the classifier weight:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

  Update weights for next iteration:

$$w_i^{(t+1)} = w_i^{(t)} \exp\left(-\alpha_t y_i h_t(\mathbf{x}_i)\right)$$

  Normalize the weights:

$$w_i^{(t+1)} = \frac{w_i^{(t+1)}}{\sum_{j=1}^{n} w_j^{(t+1)}}$$

end for
Construct final strong classifier:

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \right)$$

return $H(\mathbf{x})$

3. Proposed Models

This section includes dataset features, proposed method and model, experimental design and results.

3.1. Dataset

The dataset is provided as comma-separated values (CSV) file. Each row contains the name of the malware, API call, and class (i.e., malware type) [32]. For example, 7ff49f2f0912352416b05c010f35f402cc79feed, "IntersectRect, GetCurrentProcess, GetVersion",Virus.

- In each ZIP file obtained from VirusSamples and VirusShare, malware samples are represented with their MD5 hash codes. MD5 hash codes allow searching and analyzing these files without uploading. For every sample in the ZIP files, the hash codes are written into a text file in groups of 500. Moreover, each group is numbered from 0 to n.
- The hash codes in text files are sent to the VirusTotal system to determine their malware family. Then the analysis results are written to a file. If any of the hash code files do not exist in the VirusTotal system, the response can be empty.
- VirusTotal returns approximately 70 malware detection application results. These 70 application results are checked; the most frequently repeating name gives the malware family. It can be spyware, trojan, virus, ransomware, etc. The hash codes are grouped by their malware family.

- After grouping the malware hashes, the next step is to extract the malwares from the ZIP file. According to their malware family output from the VirusTotal query, it generates malware family folders, and each of them contains live malware with password protection "infected." The Python module PEfile is used to extract the API calls from the grouped malwares.
- Datasets with three sections: the hash codes of malware files, API calls from the PEFile library in Python, and the malware type from the VirusTotal API, are built in CSV format.

Machine learning and ensemble learning models are proposed using two datasets, VirusSample and VirusShare, obtained from VirusTotal. The datasets from [32] are converted into binary data as a result of the processing. Each API call or permission information is treated as a feature. Then, the dataset undergoes processes to handle missing data and eliminate insignificant data with very few samples. Afterward, feature extraction and the application of machine learning and ensemble learning models are carried out. The schema of the data preprocessing for machine learning is shown in Figure 1.

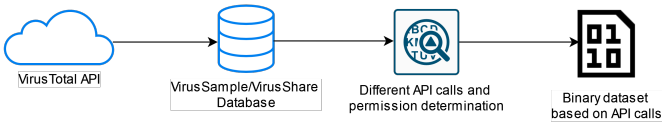

Figure 1. Data preprocessing for machine and ensemble learning.

3.2. Preprocessing

In this paper, machine learning and ensemble learning models are proposed using two different datasets obtained from VirusTotal: VirusSample and VirusShare. The VirusSample dataset contains 7966 different permissions and 9795 records, while the VirusShare dataset contains 22792 different permissions and 14616 records. These datasets include API calls and permission data for dynamic malware detection. Each row in the datasets includes the name of the malware, API calls, and the malware class.

During the data preprocessing phase, the one-hot encoding method is used. In this method, each distinct API call or permission information is considered as a feature. These features are transformed into binary data. In other words, the dataset is organized such that if the permission or API call is present in the corresponding row, it is marked as 1, and if not, it is marked as 0. This approach enables the machine learning models to produce more efficient and accurate results in malware detection. The binary data transformation allows for easier processing and analysis of complex data during the training and testing phases of the models. The original column containing the API calls is removed, leaving only the newly created features and the class label columns. As a result of these preprocessing steps, the dataset is processed similarly for thousands of rows, making it suitable for machine learning models. The transformation of the dataset is shown in Equation (8).

$$f(i,j) = \begin{cases} 0 & \text{if } j \notin i \\ 1 & \text{if } j \in i \end{cases} \tag{8}$$

Here, the values of $f(i,j)$ are defined as follows:

- i: Represents the row number (e.g., 1, 2, 3, ... n) or an observation in the dataset. In other words, each i value corresponds to a different example/row in the dataset.
- j: Represents the API call or feature (e.g., _wfopen, feof, fscanf, ...) column. In other words, each j value corresponds to a different API call.

As a result, the transformed dataset sample is shown in Figure 2.

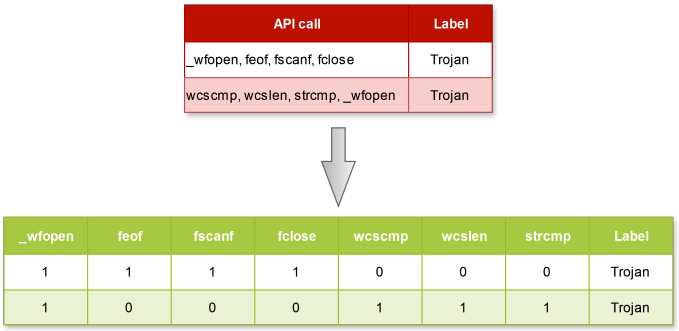

Figure 2. Transformation to binary dataset (with sample data).

After performing one-hot encoding and normalization, 7966 API calls and permissions for VirusSample and 22792 for VirusShare were obtained, each corresponding to a feature. Since the number of features beyond a certain point does not result in significant changes in the model, feature extraction is necessary. Therefore, feature extraction is carried out using PCA.

The organized datasets were tested using various machine learning and ensemble learning models. Supervised machine learning algorithms such as Naive Bayes, k-NN, SVM, DT, RF, and Adaboost, as well as various ensemble models using stacking methods, were evaluated. The performance of these models was compared based on accuracy, precision, recall, and F1-score, and the best-performing models were identified. The results demonstrate that API calls and permission data can be effectively used for dynamic malware detection.

3.3. Details of Proposed Machine Learning and Ensemble Learning Models

The dynamic datasets named VirusSample and VirusShare, which include API calls and permissions, are being tested on machine learning and ensemble models. Analyses conducted using dynamic datasets yield highly effective results in malware detection. Dynamic analysis enables the detection of malicious activities by observing the operations performed by the software and its impact on the system during its execution. In this context, dynamic datasets are crucial for understanding the real-time behaviors of malware and how it utilizes system resources.

Using dynamic datasets containing API calls and permissions for malware detection offers several advantages over traditional static analysis methods. API call data shows which system functions the malware calls and the frequency of these calls, while permission data reveals which system resources the software requests access to. This information helps machine learning algorithms classify malware more accurately. For instance, the abnormal frequency of a particular API call or permission request can indicate malicious activity. Therefore, analyses conducted using dynamic datasets play a significant role in the detection and classification of malware.

Utilizing dynamic analyses in this way enhances the performance of machine learning models, achieving higher accuracy rates in malware detection. Additionally, when combined with ensemble learning models, more reliable and comprehensive detection mechanisms are developed by leveraging the strengths of different algorithms. Thus, this article provides significant contributions to the field of cybersecurity by demonstrating the effective use of dynamic datasets and machine learning techniques.

In machine learning, models are developed using classification techniques such as SVM, HGB, and k-NN, RF without using PCA, and RF with using PCA (RF+PCA). These methods enable the categorization of data based on specific features, and each

offers different advantages in various situations. RF is a method formed by combining multiple DT and is generally known for its high accuracy rates. RF is particularly effective when the dataset contains noise or numerous features [33]. SVM aim to find the hyperplane that best separates the data and perform strongly in high-dimensional datasets.

Histogram-Based GB (HGB) is a version of GB DT and usually provides fast and effective results in large datasets. The k-NN algorithm makes classification decisions based on the data's neighbors, making it an effective method, especially when the data distribution is clear. Each of these algorithms requires selecting the most suitable model for specific data types and problems, a process that directly impacts the model's success. The performance of classification algorithms depends on various factors such as the quality of the training dataset, feature engineering, and model optimization techniques. Different from the other classification algorithms, RF is trained with and without using PCA and they are named as RF and RF+PCA. In this way, we found it appropriate to examine the effect of the PCA method on the RF model that gave the best results.

The machine learning methods are combined within a specific logic to create ensemble learning models. Bagging, stacking, and boosting are the three primary structures that constitute ensemble learning. To begin with, in bagging, the dataset is divided into test and train groups, typically in a 70/30 ratio. Multiple bags are created by randomly and repeatedly sampling from the training data. Each bag is then trained using established models. The final decision is made by averaging the outputs or voting. Similarly, in boosting, data is also separated and randomly sampled. However, unlike bagging, each sample in boosting is trained independently and produces an output, giving each model an equal chance of success. In boosting, priority is given to data misclassified by previous models [29].

During the boosting process, three sets of classifiers are simultaneously created. The first and second classifiers are trained using different random subsets of the dataset, similar to bagging. The third classifier is trained on the data where the first and second classifiers failed. These three classifiers are then combined using the majority vote technique. In contrast, stacking makes decisions based on the percentage of the feature space where each classifier is successful. The outputs from all classifiers are combined with another classifier to make the final decision [34]. The flow of the proposed model is shown in Figure 3.

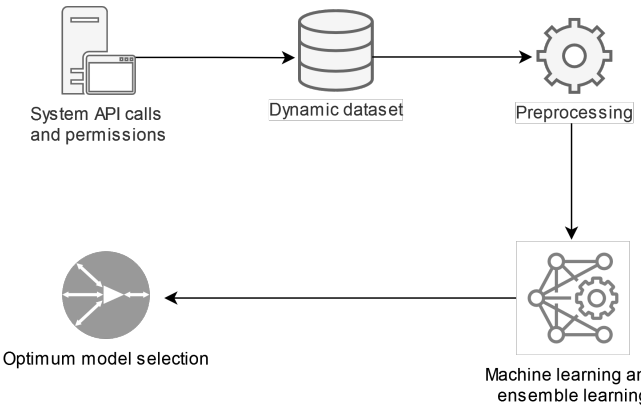

Figure 3. The flow of the proposed model.

3.4. Experimental Results

The experimental studies of the proposed method are conducted on a computer running Windows 11 64-bit operating system, equipped with a 12th Gen Intel(R) Core(TM) i7-12700H processor (20 CPUs) at 2.3GHz, 16 GB RAM, and an NVIDIA GeForce RTX 3060 GPU, using Python 3.x. Machine learning models such as SVM, RF, k-NN, and DT, as well as ensemble models like GB and AdaBoost, and stacking-based ensemble learning models where

machine learning models serve as base and meta classifiers, are being trained and tested on two different datasets. As a result of these processes, the models with the highest performance for the VirusSample and VirusShare datasets are RF as a machine learning model, RF-DT $\rightarrow$ k-NN (Stacking) contains PCA as an ensemble learning model for the VirusSample dataset, and RF as a machine learning model, RF-k-NN-GB $\rightarrow$ SVM (Stacking) as an ensemble learning model for the VirusShare dataset, respectively. Each model is compared using the parameters of accuracy, precision, recall, and F-score. Different from other ensemble models, we trained RF-DT $\rightarrow$ k-NN model which gives the best result in VirusSample and RF-k-NN-GB $\rightarrow$ SVM which gives the best results in VirusShare are trained with and without using PCA to see the effect of the feature extraction process and to improve the obtained results.

These metrics are derived from the complexity matrix of the model's output, which includes four states: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). The first metric evaluated based on these states is accuracy, which represents the proportion of correct predictions out of all predictions made. The accuracy formula is shown in Equation (9):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

Precision, defined as the ratio of correctly predicted positive results to all predicted positive results, is given by Equation (10):

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

Recall, the ratio of correctly predicted positive results to all actual positive results, is expressed by Equation (11):

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

The F-score, which is the weighted harmonic mean of precision and recall, is calculated using Equation (12):

$$Fscore = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{12}$$

The results for the VirusSample dataset are shown in Table 1, and the results for the VirusShare dataset are shown in Table 2.

4. Discussion and Future Directions

In this section, the findings obtained from the experimental results are discussed, a sensitivity analysis is conducted for the models with the highest performance, the results are compared with similar studies, and directions for future work are provided.

The results obtained from machine learning and ensemble learning models applied on the VirusSample dataset provide significant insights into the malware classification performance of different models. Notably, the RF model demonstrates superior performance compared to other models, achieving the highest values in metrics such as precision, recall, accuracy, and F-score. After applying PCA for feature extraction, the results of the RF model are improved. The high performance of the RF model can be attributed to its effectiveness in handling complex data structures and large datasets. Additionally, the RF model's ability to reduce the

Table 1.
Machine learning and ensemble learning model results with VirusSample dataset.

Model	Precision	Recall	Accuracy	F-score
SVM	0.936459	0.938755	0.938755	0.931427
RF	0.944652	0.946921	0.946921	0.940913
RF+PCA	0.947102	0.948316	0.948316	0.941883
k-NN	0.849495	0.836679	0.836679	0.820684
DT	0.934151	0.938074	0.938074	0.933382
GB	0.932013	0.933991	0.933991	0.930013
Adaboost	0.749938	0.789384	0.789384	0.761008
RF-k-NN-DT → SVM (Stacking) (RKD.S)	0.939686	0.945560	0.945560	0.938269
RF-SVM-DT → k-NN (Stacking) (RSD.K)	0.939052	0.934331	0.934331	0.934700
RF-k-NN-GB → SVM (Stacking) (RKG.S)	0.940181	0.945219	0.945219	0.938247
RF-SVM-GB → k-NN (Stacking) (RSG.K)	0.940952	0.936713	0.936713	0.936646
RF-DT → SVM (Stacking) (RD.S)	0.940426	0.945219	0.945219	0.938309
RF-DT → k-NN (Stacking) (RD.K)	0.943125	0.945219	0.945219	0.939621
RF-DT → k-NN (Stacking) (RD.K) + PCA	0.943321	0.945621	0.945621	0.938387

Table 2.
Machine learning and ensemble learning model results with VirusShare dataset.

Model	Precision	Recall	Accuracy	F-score
SVM	0.832910	0.844698	0.844698	0.819055
RF	0.846609	0.857241	0.857241	0.842304
RF+PCA	0.853505	0.862731	0.862731	0.848362
k-NN	0.762199	0.768757	0.768757	0.746775
DT	0.833990	0.843786	0.843786	0.835663
GB	0.834480	0.842417	0.842417	0.828339
Adaboost	0.580092	0.663170	0.663170	0.599148
RF-k-NN-DT → SVM (Stacking) (RKD.S)	0.844701	0.854732	0.854732	0.837243
RF-SVM-DT → k-NN (Stacking) (RSD.K)	0.820234	0.839909	0.839909	0.819988
RF-k-NN-GB → SVM (Stacking) (RKG.S)	0.845721	0.857013	0.857013	0.839285
RF-k-NN-GB → SVM (RKG.S) + PCA	0.849313	0.862070	0.862070	0.844911
RF-SVM-GB → k-NN (Stacking) (RSG.K)	0.824265	0.843101	0.843101	0.823963
RF-DT → SVM (Stacking) (RD.S)	0.843094	0.854504	0.854504	0.836565
RF-DT → k-NN (Stacking) (RD.K)	0.835409	0.843558	0.843558	0.834482

risk of overfitting and determine feature importance contributes to its high performance.

Other classical models, such as SVM and DT, also show noteworthy performance. The F-score value of the SVM model being 93.14% indicates that the model performs quite balanced in classification tasks. Similarly, the DT model is notable for its high accuracy and precision values. These results suggest that SVM and DT models can be effective in malware classification tasks. However, models like k-NN and Adaboost exhibit lower performance compared to others. Particularly, the Adaboost model's

F-score value of 76.10% indicates that the model is less effective on this dataset compared to other models.

The results obtained from ensemble learning methods are also quite intriguing. Particularly, stacking models created by combining RF and DT models with SVM and k-NN have shown high performance. The stacking model contains PCA as feature extraction, with RF and DT as base classifiers and k-NN as the meta classifier, achieved the highest accuracy value (94.56%) and F-score value (93.84%). This result indicates that ensemble learning methods can achieve higher performance by combining the strengths of different models. The success of stacking models stems from their ability to compensate for the shortcomings of individual models.

In this paper, the performance of various machine learning and ensemble learning models in classifying malware is evaluated using the VirusSample dataset as well as the VirusShare dataset. The results are compared based on performance metrics such as accuracy, recall, precision, and F-score.

The SVM model has demonstrated generally strong performance with an accuracy of 84.47% and an F-score of 81.91%. However, the RF+PCA model has achieved the highest performance with an accuracy of 86.27% and an F-score of 84.84%. The RF model stands out with its high precision and recall values, indicating its ability to accurately classify malware. DT and GB models are also showed similarly high performance, although they lagged behind the RF model.

The k-NN model, on the other hand, exhibited lower performance compared to other models, achieving an accuracy of 76.88% and an F-score of 74.68%. This suggests that the k-NN model may be less effective on large and diverse datasets compared to other models. The AdaBoost model showed the lowest performance with an accuracy of 66.32% and an F-score of 59.91%, indicating that this model is less successful in malware classification compared to other methods.

One of the ensemble methods, stacking, has successfully increased overall performance by combining different models. For example, the stacking contains PCA as feature extraction, with RF, k-NN, and GB as base classifiers and SVM as the meta classifier, has achieved the highest performance among the ensemble models with an accuracy of 86.21% and an F-score of 84.49%. This demonstrates that stacking methods can provide higher classification accuracy by combining the strengths of various models. Other stacking combinations are also showed similarly high performance, indicating that ensemble methods are an effective strategy for enhancing the performance of machine learning models.

We have also conducted an experimental study to see the effectiveness of the PCA feature extraction algorithm. For this purpose, we have used 2 different feature extraction algorithms: linear discriminant analysis (LDA) and kernel PCA (KPCA). LDA is a dimensionality reduction technique used to project data onto a lower-LDA is a dimensionality reduction technique used to project data onto a lower-dimensional space [35]. It maximizes the separation between multiple classes by finding a linear combination of features that best separates them. In contrast, KPCA is an extension of PCA that uses kernel functions to project data into a higher-dimensional feature space, allowing for the capture of nonlinear relationships and complex structures within the data [36]. The comparison results of feature extraction methods is presented in Table 3. As can be seen from the comparison table, LDA produced the worst results. This is because the LDA method does not take into account the within-class

variance. In contrast, the PCA method yielded much better results, while KPCA gave relatively similar results. This is because the KPCA method performs better on nonlinear and highly complex datasets; however, the dataset used does not fall into this category.

In conclusion, this paper comprehensively evaluates the effectiveness of different machine learning and ensemble learning models in malware classification using two different dynamic datasets. For both datasets, RF and stacking methods using PCA stand out with high accuracy and F-score values. These findings suggest that exploring more complex ensemble methods and model optimization techniques in future studies could be beneficial. Additionally, using different datasets and various feature engineering techniques can further enhance the generalizability and performance of the models. In this context, the results obtained on the VirusSample and VirusShare binary datasets can be considered an important reference point in the field of malware classification.

Dynamic datasets offer significant advantages in malware detection by enabling real-time monitoring and analysis of malware behaviors, which is crucial due to the evolving nature of malware. Unlike static methods, dynamic analysis captures API calls and other behavioral features, providing more up-to-date and accurate results. This approach is more resilient against evasion and anti-detection techniques as it examines activities in a real operating environment. Thus, dynamic datasets enhance the reliability of analyses and play a crucial role in detecting and classifying new and advanced malware, improving the effectiveness of security solutions.

All results in Table 1 and Table 2 have obtained by randomly splitting the datasets into 70% for training and 30% for testing. The paper provides a sensitivity analysis for the machine learning and ensemble learning models with the highest F-score values, based on random splits of the datasets into 60-40, 80-20, and 90-10 for training and testing, respectively. The sensitivity analyses for the VirusSample and VirusShare datasets are shown in Table 4 and Table 5, respectively.

The sensitivity analyses conducted in the paper evaluate the impact of different train-test split ratios on model performance. For this purpose, we used models that gave the best results. Table 4 provides valuable insights into the performance stability of machine learning and ensemble learning models under various train-test splits. The analysis results show that the RF model consistently performs well across different splits and exhibits a notable improvement in performance metrics as the proportion of training data increases. For instance, the RF model achieves an F-score of 93.95% with a 60-40 split, which rises to 94.65% with a 90-10 split. This indicates that the RF model benefits from more training data, allowing for better generalization and higher accuracy in malware classification.

Similarly, the stacking model with RF and DT as base classifiers and k-NN as the meta-classifier also demonstrates robust performance across different splits. The F-score for this model increases from 90.49% with a 60-40 split to 94.54% with a 90-10 split. The slight variations in precision, recall, and accuracy metrics across the splits suggest that the stacking model effectively leverages the strengths of its component models, even with varying amounts of training data. The performance stability of the stacking model highlights its potential, particularly in dynamic malware detection tasks, where the ability to adapt to different data volumes is crucial.

Table 5 presents the results of experiments on the RF model and the stacking model with RF, k-NN, and GB as base classifiers and SVM as the meta classifier, showing the impact of different train-

Table 3.
Comparision of different feature extraction algorithms.

Feature Extraction	Dataset	Model	Precision	Recall	Accuracy	F-score
PCA	VirusSample	RF	0.9471	0.9483	0.9483	0.9419
		RD.K	0.9433	0.9456	0.9456	0.9384
	VirusShare	RF	0.8535	0.8627	0.8627	0.8484
		RKG.S	0.8493	0.8621	0.8621	0.8449
LDA	VirusSample	RF	0.8955	0.8874	0.8874	0.8870
		RD.K	0.8941	0.8881	0.8881	0.8851
	VirusShare	RF	0.7400	0.7432	0.7432	0.7310
		RKG.S	0.7245	0.7314	0.7314	0.7167
KPCA	VirusSample	RF	0.9443	0.9459	0.9459	0.9390
		RD.K	0.9393	0.9415	0.9415	0.9353
	VirusShare	RF	0.8539	0.8620	0.8620	0.8458
		RKG.S	0.8552	0.8607	0.8607	0.8424

Table 4.
Sensitivity analysis data for VirusSample dataset based on train/test split.

Model	Split	Precision	Recall	Accuracy	F-score
RF	60-40	0.944046	0.945636	0.945636	0.939537
	70-30	0.944652	0.946921	0.946921	0.940913
	80-20	0.947824	0.949974	0.949974	0.944718
	90-10	0.949576	0.951020	0.951020	0.946516
RD.K	60-40	0.921750	0.900970	0.900970	0.904920
	70-30	0.943125	0.945219	0.945219	0.939621
	80-20	0.946120	0.940786	0.940786	0.941256
	90-10	0.947843	0.944898	0.944898	0.945413

Table 5.
Sensitivity analysis data for VirusShare dataset based on train/test split.

Model	Split	Precision	Recall	Accuracy	F-score
RF	60-40	0.842483	0.854797	0.854797	0.839651
	70-30	0.846609	0.857241	0.857241	0.842304
	80-20	0.848422	0.858413	0.858413	0.843927
	90-10	0.852309	0.864569	0.864569	0.848250
RKG.S	60-40	0.848933	0.859415	0.859415	0.842180
	70-30	0.845721	0.857013	0.857013	0.839285
	80-20	0.854491	0.859439	0.859439	0.842180
	90-10	0.848450	0.862517	0.862517	0.842764

test split ratios on metrics such as accuracy, recall, precision, and F-score.

The analyses for the RF model indicate a general improvement in performance as the size of the training dataset increases. Particularly, with a 70-30 split, the RF model has achieved the highest performance with an accuracy of 84.53% and an F-score of 84.83%. These results demonstrate that providing more training data allows the model to classify malware more accurately and consistently.

A similar trend is observed for the ensemble model with the highest performance, which is the stacking model consisting of RF, k-NN, and GB as base classifiers and SVM as the meta classifier. This model achieved high accuracy and F-score values with 80-20 and 90-10 splits. Specifically, with a 90-10 split, the model has achieved an accuracy of 86.25% and an F-score of 84.28%. This indicates that ensemble methods are more robust against variations in dataset size and can produce more consistent results by combining the strengths of different models.

Overall, the sensitivity analysis results show that increasing the size of the training dataset improves the performance of both individual and ensemble models. These findings highlight the importance of large and diverse datasets in developing more effective malware detection and classification models. They emphasize the need to carefully consider data split strategies to ensure robust and reliable model evaluation in malware detection research.

The proposed models are compared with similar studies using the same datasets. However, in this paper, these two datasets have underwent a preprocessing process, converting the data into a binary dataset and also PCA is used for feature extraction. Therefore, the datasets used are not exactly the same. However, this comparison is made because they are still similar in terms of content. The results

Table 6.
Comparison result with similar study using same datasets.

Machine Learning Results					
Dataset: VirusSample			**Dataset: VirusShare**		
Paper	Model	F-score	Paper	Model	F-score
[32]	SVM	0.8975	[32]	SVM	0.7581
	RF	0.8391		RF	0.6609
Our	SVM	0.9314	Our	SVM	0.8191
	RF+PCA	0.9419		RF+PCA	0.8484
Ensemble Learning Results (Highest)					
Dataset: VirusSample			**Dataset: VirusShare**		
Paper	Model	F-score	Paper	Model	F-score
[32]	XGB	0.9031	[32]	XGB	0.7525
Our	RD.K+PCA	0.9384	Our	RKG.S+PCA	0.8449

shown in Table 6 compare the performance of various machine learning and ensemble learning models on two different dynamic malware datasets named VirusSample and VirusShare. When comparing our results with [32], it is possible to see the success rates of different methods on both datasets.

According to the data presented in Table 6, the results obtained from two different studies are compared. It is observed that our results, indicated as ”Our”, achieve higher F-scores on both the VirusSample and VirusShare datasets compared to [32]. This indicates that our results demonstrate superior performance in both machine learning and ensemble learning methods.

When examining the machine learning results, for the VirusSample dataset, the F-score values for the SVM and RF models in [32] are 0.8975 and 0.8391, respectively, whereas our results show these values as 0.9314 and 0.9419. Similarly, for the VirusShare dataset, our results are also higher. In the compared study, the F-score values for the SVM and RF models on the VirusShare dataset are 0.7581 and 0.6609, respectively, while in our results, these values are 0.8191 and 0.8484.

Table 7.
Comparison result with similar studies using different datasets.

Paper	Classifier	Datasets	Accuracy
[3]	RF, J48, SVM	Malgenome	91.90%
[4]	RF, SVM, k-NN, LR	Drebin, CICInvesAndMal2019	93.77%
[5]	Two-class SVM	Google Play (Malware sample sources not disclosed)	86.50%
[7]	Deep Learning	Alibaba Cloud Security Malware	94.49%
Our	RF, RD.K, RKG.S	VirusSample, VirusShare	94.83%

Ensemble learning results also show our high results. For the VirusSample dataset, the F-score for the XGB model in the compared paper is 0.9031, while our results show an F-score of 0.9384 with the RD.K+PCA model. For the VirusShare dataset, the F-score for the XGB model in the same study is 0.7525, whereas our results achieve 0.8449 with the RKG.S+PCA model. In ensemble models, we tried different combinations of stacking various machine learning models, and in Table 6, we share the stacking combinations that achieved the highest results. The impact of additional preprocessing performed and PCA feature extraction on the datasets is evident in the differences observed.

In addition to the comparison results that use the same dataset presented in Table 6, we have compared our results with similar studies that use dynamic features and machine-learning techniques on different datasets for malware detection. For this purpose, we have chosen four similar studies which are [3], [4], [5], [7]. The used datasets, machine learning methods, and best accuracy results are presented in Table 7. As can be seen, the proposed model gives the best accuracy result.

In future studies, several potential directions can be explored to enhance the effectiveness of malware detection using dynamic datasets. The integration of advanced feature selection techniques that can dynamically adapt to evolving malware behaviors can be implemented. Deep learning architectures, particularly recurrent neural networks (RNNs), which can better capture the sequential nature of API calls, can be explored. Finally, expanding the scope of dynamic datasets to include more diverse and representative samples can further enhance the generalizability of the models, ensuring they remain effective against a broader range of malware types.

5. Conclusion

This paper demonstrates the effectiveness of feature extraction, machine learning and ensemble learning models in detecting malware using dynamic analysis datasets. For both the VirusSample and VirusShare datasets, the RF+PCA model achieved the highest performance with accuracies of 94.83% and 86.27%, respectively. Among ensemble learning methods, the RD.K+PCA model has achieved the highest performance on the VirusSample dataset with an accuracy of 94.56%, while the RKG.S+PCA model on the VirusShare dataset with an accuracy of 86.21%. These findings highlight the importance of dynamic analysis for real-time malware detection and the potential of ensemble methods to improve classification accuracy. Future research should explore advanced feature selection techniques and deep learning architectures to further enhance model performance. The integration of

diverse dynamic datasets will be crucial in developing robust and comprehensive malware detection solutions, contributing significantly to the field of cybersecurity.

References

[1] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, "Execute this! analyzing unsafe and malicious dynamic code loading in android applications," in *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS)*, vol. 14, 2014, pp. 23–26.

[2] M. Ahmad, V. Costamagna, B. Crispo, F. Bergadano, and Y. Zhauniarovich, "Stadart: Addressing the problem of dynamic code updates in the security analysis of android applications," *Journal of Systems and Software*, vol. 159, p. 110386, 2020.

[3] Y. Rosmansyah, B. Dabarsyah *et al.*, "Malware detection on android smartphones using api class and machine learning," in *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, 2015, pp. 294–297.

[4] A. Roy, D. S. Jas, G. Jaggi, and K. Sharma, "Android malware detection based on vulnerable feature aggregation," *Procedia Computer Science*, vol. 173, pp. 345–353, 2020.

[5] F. Shen, J. Del Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek, "Android malware detection using complex-flows," *IEEE Transactions on Mobile Computing*, vol. 18, no. 6, pp. 1231–1245, 2018.

[6] T. Chen, H. Zeng, M. Lv, and T. Zhu, "Ctimd: Cyber threat intelligence enhanced malware detection using api call sequences with parameters," *Computers & Security*, vol. 136, p. 103518, 2024.

[7] S. Zhang, J. Wu, M. Zhang, and W. Yang, "Dynamic malware analysis based on api sequence semantic fusion," *Applied Sciences*, vol. 13, no. 11, 2023.

[8] A. A. Alhashmi, A. A. Darem, A. M. Alashjaee, S. M. Alanazi, T. M. Alkhaldi, S. A. Ebad, F. A. Ghaleb, and A. M. Almadani, "Similarity-based hybrid malware detection model using api calls," *Mathematics*, vol. 11, no. 13, 2023.

[9] C. Li, Q. Lv, N. Li, Y. Wang, D. Sun, and Y. Qiao, "A novel deep framework for dynamic malware detection based on api sequence intrinsic features," *Computers & Security*, vol. 116, p. 102686, 2022.

[10] C. Li, Z. Cheng, H. Zhu, L. Wang, Q. Lv, Y. Wang, N. Li, and D. Sun, "Dmalnet: Dynamic malware analysis based on api feature engineering and graph learning," *Computers & Security*, vol. 122, p. 102872, 2022.

[11] J. Singh and J. Singh, "Assessment of supervised machine learning algorithms using dynamic api calls for malware detection," *International Journal of Computers and Applications*, vol. 44, no. 3, pp. 270–277, 2022.

[12] J. Yang, J. Tang, R. Yan, and T. Xiang, "Android malware detection method based on permission complement and api calls," *Chinese Journal of Electronics*, vol. 31, no. 4, pp. 773–785, 2022.

[13] J. Tang, W. Xu, T. Peng, S. Zhou, Q. Pi, R. He, and X. Hu, "Android malware detection based on a novel mixed bytecode image combined with attention mechanism," *Journal of Information Security and Applications*, vol. 82, p. 103721, 2024.

[14] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[15] H.-j. Zhu, W. Gu, L.-m. Wang, Z.-c. Xu, and V. S. Sheng, "Android malware detection based on multi-head squeeze-and-excitation residual network," *Expert Systems with Applications*, vol. 212, p. 118705, 2023.

[16] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[17] A. Ksibi, M. Zakariah, L. Almuqren, and A. S. Alluhaidan, "Efficient android malware identification with limited training data utilizing multiple convolution neural network techniques," *Engineering Applications of Artificial Intelligence*, vol. 127, p. 107390, 2024.

[18] M. Tan, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[19] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, pp. 1–12, 2017.

[20] A. T. W. Almais, A. Susilo, A. Naba, M. Sarosa, C. Crysdian, I. Tazi, M. A. Hariyadi, M. A. Muslim, P. M. N. S. A. Basid, Y. M. Arif, M. S. Purwanto, D. Parwatiningtyas, Supriyono, and H. Wicaksono, "Principal component analysis-based data clustering for labeling of level damage sector in post-natural disasters," *IEEE Access*, vol. 11, pp. 74 590–74 601, 2023.

[21] L.-C. Chang, J.-Y. Liou, and F.-J. Chang, "Spatial-temporal flood inundation nowcasts by fusing machine learning methods and principal component analysis," *Journal of Hydrology*, vol. 612, p. 128086, 2022.

[22] S. A. Abdul-Wahab, C. S. Bakheit, and S. M. Al-Alawi, "Principal component and multiple regression analysis in modelling of ground-level ozone and factors affecting its concentrations," *Environmental Modelling & Software*, vol. 20, no. 10, pp. 1263–1271, 2005.

[23] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[24] M. Schonlau and R. Y. Zou, "The random forest algorithm for statistical learning," *The Stata Journal*, vol. 20, no. 1, pp. 3–29, 2020.

[25] J. R. Quinlan, "Learning decision tree classifiers," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 71–72, 1996.

[26] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and

B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.

[27] A. Almomany, W. R. Ayyad, and A. Jarrah, "Optimized implementation of an improved knn classification algorithm using intel fpga platform: Covid-19 case study," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 3815–3827, 2022.

[28] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, p. 21, 2013.

[29] A. Karakaya, A. Ulu, and S. Akleylek, "Goalalert: A novel real-time technical team alert approach using machine learning on an iot-based system in sports," *Microprocessors and Microsystems*, vol. 93, p. 104606, 2022.

[30] T. Hastie, R. Tibshirani, and J. Friedman, "Boosting and additive trees," *The elements of statistical learning: data mining, inference, and prediction*, pp. 337–387, 2009.

[31] A. Karakaya and S. Akleylek, "A novel iot-based health and tactical analysis model with fog computing," *PeerJ Computer Science*, vol. 7, p. e342, 2021.

[32] B. Gençaydin, C. N. Kahya, F. Demirkiran, B. Düzgün, A. Çayir, and H. Dağ, "Benchmark static api call datasets for malware family classification," in *2022 7th International Conference on Computer Science and Engineering (UBMK)*, 2022, pp. 1–5.

[33] A. Karakaya, I. Karakaya, and T. Temizceri, "An online shoppers purchasing intention model based on ensemble learning," in *2023 4th International Informatics and Software Engineering Conference (IISEC)*, 2023, pp. 1–4.

[34] A. Karakaya and A. Ulu, "A novel mobile malware detection model based on ensemble learning," in *2023 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2023, pp. 1–6.

[35] P. Xanthopoulos, P. M. Pardalos, T. B. Trafalis, P. Xanthopoulos, P. M. Pardalos, and T. B. Trafalis, "Linear discriminant analysis," *Robust data mining*, pp. 27–33, 2013.

[36] H. Hoffmann, "Kernel pca for novelty detection," *Pattern recognition*, vol. 40, no. 3, pp. 863–874, 2007.