International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V74I3P113

Volume 74 Issue 3, 169-180, March 2026
© 2026 Seventh Sense Research Group®

Original Article

Smart TPOT-Based AutoML-Powered Android Malware Detection and Classification

Divya Recharla[1], M. Monisha[2]

[1,2]Department of Electronics and Communication Engineering, VISTAS, Chennai, Tamil; Nadu, India.

[1]Corresponding Author : dharani.divya14@gmail.com

Received: 23 April 2025 Revised: 27 October 2025 Accepted: 25 November 2025 Published: 28 March 2026

Abstract *- Systems and Networks can be seriously threatened by malware, often known as malicious software. The sophistication of malware assaults is increasing, making it harder to identify and stop them, for several reasons, including the protection of private data, data loss and alteration, system interruptions, monetary losses, and reputational harm. Therefore, malware detection and prevention are essential. Various machine learning models such as Random Forest, Support Vector Machine, K-NN, Extra Tree classifier, Gradient Boosting, and AdaBoost are applied for Android malware detection, as presented in this research. A Python-based machine learning tool called Python Optimised ML Pipeline (TPOT) uses genetic programming to maximize network throughput. To retrieve static information like permissions, network calls, API calls, and system traffic from the malicious apps for Android dataset, we employ TPOT to construct models. Moreover, a comparison has been made with traditional Machine learning classifiers and Automated ML for greater performance by reduction in computational time, training speed, and efficiency. Subsequently, metrics such as precision, F1-score, Recall, and accuracy are used to evaluate the overall performance of models. The analysis proved that Automated ML provides better outcomes of 99.7% accuracy with lesser computational complexity and a reduction in training time.*

Keywords *- Automated Machine Learning (AutoML), Tree-based Pipeline Optimization Tool (TPOT), Gradient Boosting, ExtraTree, and AdaBoost.*

1. Introduction

With the growing popularity of the cellular system, Android malicious apps, or malicious software, have emerged as a significant security concern [1]. For instance, an increasing number of app consumers save private information on their smartphones, including financial transactions [2], which means that cybercriminals turn their focus to mobile devices and attempt to carry out harmful actions using Android applications. It is hardly unexpected that several methods for identifying malicious applications for Android were recently put forth. Features (such as permissions, intents, and API calls) taken from the APK file are used in Android virus identification and evaluation to identify malware [3].

App analysts typically start by removing risky permissions and intents from AndroidManifest.xml. They find dangerous code segments that result in malicious behaviors by using pre-existing tools (like dex2jar) to decompile Dalvik executable (dex) files in the Android Application Package (apk) file and obtain the source code. They then read the source code from start to finish. Lastly, they are able to recognize malware by its malevolent activities, which are highly comprehensible to humans. To clarify the ML predictions, a few essentials may help better comprehend the traits that an Android application might exhibit and classify it as spyware by using permissions, APIs, intents, or code segments to match specific behaviors of the app, as shown in Figure 1.

Therefore, we must determine which factors significantly influence the classification in machine learning and whether they are indeed connected to the dangerous behaviors of the virus in order to explain why an app is labeled as malicious. From the above Figure, we are discussing the benefits and limitations of static and dynamic analysis. The primary benefit of static analysis is its ability to identify errors (or weak areas) on its own after a product is out on the marketplace. Just since it is examined and primarily because it is more beneficial on Android devices with limited memory, the absence of dangerous software favored [5].

To observe the activity of the usage, dynamic evaluation necessitates running it in a separate environment. In contrast to static investigation, dynamic analysis takes into account the source code execution procedure in order to uncover an app's inherent behaviors. Dynamic analysis is used to monitor network activity, file changes, API calls, and identify system calls [6]. Calik et al. [4] identified software codes to ascertain their features and behaviours in order to determine the possibility that they are malware, which is known as malware software analysis. Both static and dynamic evaluations are categories of Android malware detection techniques. The frequently used Android Malware Analysis Taxonomies are displayed in Figure 2.

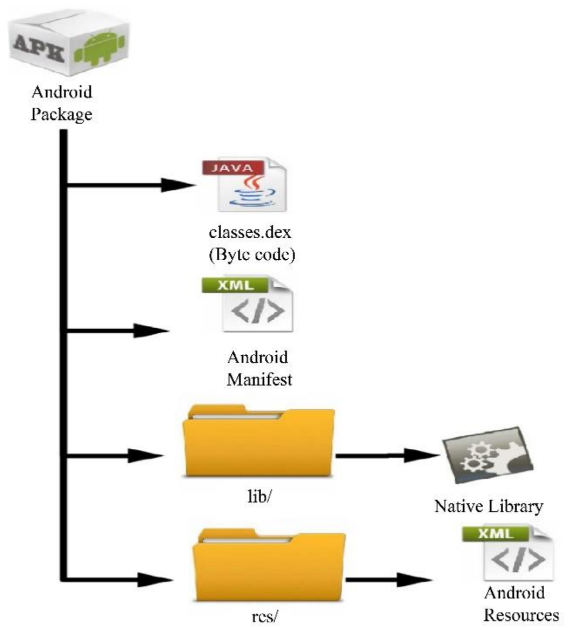

Fig. 1 Android-based application packages

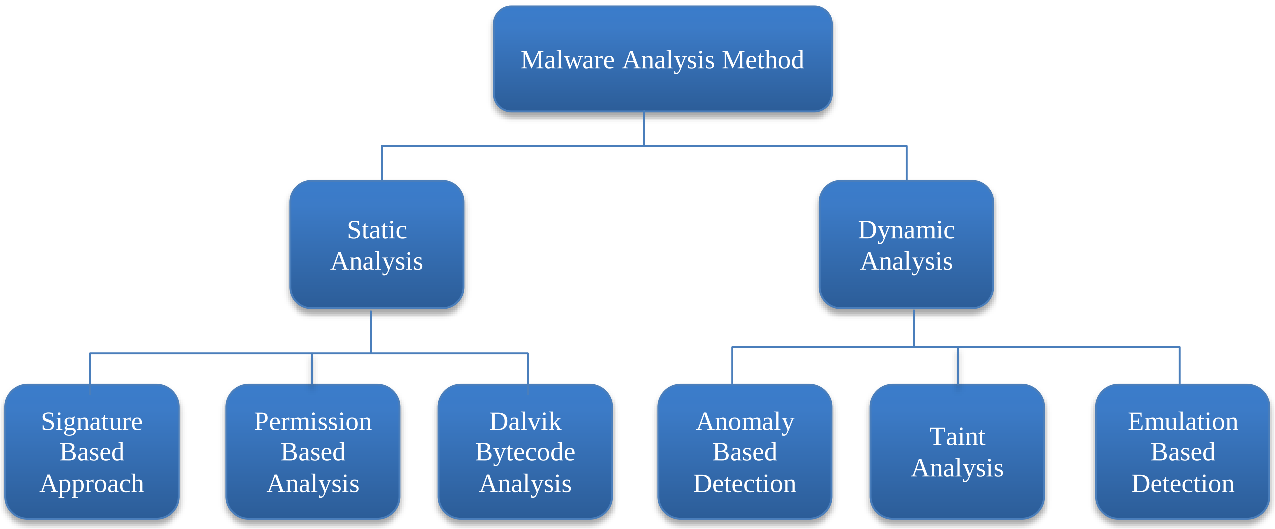

Fig. 2 Taxonomy of android malware analysis

Despite the rise of malicious code, there is currently no reliable and efficient way to identify malicious applications. We think machine learning techniques can be used to solve the problem of malware detection as machine learning becomes more and more applicable in other fields. Our research intends to produce an effective Machine Learning Model that can categorize apps into normal (0) and threat (1) based on the requested app permissions, as well as a thorough and methodical investigation of malware detection using machine learning techniques.

1.1. Problem Statement

As the Android market continues to grow, the number of apps with malicious activity is rising. Ten to twenty-four percent of apps available on the Play Store may be fraudulent, according to ZDNet. These applications appear to be just like any other software on the surface, but they exploit the user's system in several dangerous ways. The existing malware detection techniques are laborious and resource-intensive, but they are unable to keep up with the rapidity of emerging malware.

To overcome these challenges, this research work focused on:

- Developed a stratagem to access and analyze data of confirmed malicious applications.
- Machine learning models are employed to predict Android-based malicious applications in an efficient manner.
- Moreover, automated machine learning models are applied to train the Android-based application features, leading to a reduction in computational complexity.
- Comparison has been done among manually trained machine learning models with an auto-ML approach in terms of performance metrics, namely accuracy, F1-score, precision, and recall, in Android malware detection, relying on a publicly available dataset.

2. Related Work

Bernardo et al. [7] presented an AUTO-ML approach for identifying lung diseases by detecting COVID on chest imaging using the AutoML TPOT platform and Transfer Learning extraction of features. Nayana et al. [8], in order to identify lung tumors from CT scans, created the Tree-based Pipeline Optimization Tool with Support Vector Machine (TPOT_SVM).

Amarjyoti et al. [9] used the feature significance score as a tool to determine which characteristics in the dataset were most influential-applied both the complete feature set and the selected feature set to several permission-based datasets to guarantee their accuracy. A comparison analysis of the algorithm's performance yields insightful results, and the Random Forest classifier gets the most fantastic accuracy, around 94.4% in the entire dataset, with the fewest losses. Various researchers found that Deep Learning based recurrent Neural Network [10], hybrid model [11], and Convolutional RNN [12] for opcode sequences, Deep Learning for botnet malware [13] are appropriate in detecting Android-based malware.

To reduce manual consumption for training data, feature selection and extraction, tuning the model, and reduction in computational complexity requires custom feature engineering. Our proposed work, TPOT with a Random Forest classifier, is best suited for Android malware classification. The strength of this research work is that it proposed an AutoML-based TPOT appropriate for selecting features, extracting features, choosing a suitable model, hyperparameter optimization, and everything done automatically, hence it increases model development efficiency and minimizes the requirement for data professionals to intervene manually.

Arp, Spreitzenbarth, Hübner, Gascon, and Rieck (Drebin - 2014) introduced Drebin, a large-scale static-analysis system for Android malware detection that emphasized explainability. Moreover, extracted diverse static features from APKs (manifest metadata, permissions, API calls, network addresses, suspicious strings) and represented apps as feature vectors. Used linear classifiers (SVM-style / lightweight models) that produced interpretable feature weights for analysts. Evaluated on thousands of benign and malicious samples collected from app stores and malware repositories; reported high detection rates for known malware families. A static-only approach is vulnerable to code obfuscation, dynamic payload loading, and reflection. Also, static features can miss runtime-only malicious behavior. Drebin highlights the importance of curated, explainable static features and shows that interpretable models matter in security. Smart TPOT should include interpretable classifiers and domain-aware static feature primitives in its search space.

Olson, Bartley, Urbanowicz, and Moore (TPOT - 2016) proposed TPOT (Tree-based Pipeline Optimization Tool), an AutoML system that uses genetic programming to evolve end-to-end ML pipelines (preprocessing, feature selection, model choice, hyperparameters). TPOT encodes pipelines as trees and applies mutation/crossover guided by a population-level fitness (e.g., cross-validated accuracy). It can discover non-trivial preprocessing + model combinations. Benchmarked on standard ML datasets, TPOT often matches or outperforms manual tuning while reducing human effort. Computationally expensive for very high-dimensional datasets and can overfit if the search is not constrained. TPOT is the backbone. Smart TPOT must adapt TPOT's genetic-search idea to malware-specific primitives, constrain the search to domain-relevant operations, and add multi-objective fitness (accuracy + latency + model size).

Allix, Bissyandé, Klein, and Le Traon (AndroZoo - dataset) built and released AndroZoo, a massive repository of Android APKs collected over time from multiple sources for research use. AndroZoo provides metadata, APK files, and derived artifacts (signatures, certificates), enabling large-scale empirical studies and reproducible experiments. Hosts millions of APKs across benign and malicious apps with timestamps essential for temporal evaluations. Label quality varies depending on external labeling sources, and metadata may lack dynamic behavioral traces. AndroZoo is a primary data source for training and temporal-split evaluation of Smart TPOT; its scale allows assessment of model drift and family-wise generalization.

Mariconti, Onwuzurike, Andriotis, De Cristofaro, Ross, and Stringhini (MaMaDroid) developed MaMaDroid, a behavior-based detection approach modeling sequences of API calls as Markov chains (behavioral profiles) to detect malware. Extracted API-call sequences and abstracted them (e.g., to package/class-level) to build Markov models; used distance measures/classifiers to classify apps. Showed robustness to specific code changes and reasonable detection across time by focusing on behavior instead of raw API tokens. Building behavioral models requires access to code traces; static extraction of call sequences can still be affected by dynamic loading or reflection. Also, Markov abstractions can lose fine-grained information. Shows the value of sequence/behavioral features - Smart TPOT should include sequence-encoding primitives (n-grams, Markov features, embeddings) in its pipeline search.

Feurer, Klein, Eggensperger, Springenberg, Blum, Hutter (Auto-sklearn) Created Auto-sklearn, an AutoML system that uses Bayesian optimization and meta-learning to initialize and accelerate model search. Combines automated preprocessing, model selection, and hyperparameter tuning with warm-start from meta-learned configurations; focuses heavily on computational efficiency. Strong performance on general ML benchmarks; showed that meta-learning reduces search time. Off-the-shelf AutoML may lack domain knowledge for specialized tasks (e.g., malware features), so pure AutoML can miss domain-specific preprocessing. Suggests Smart TPOT could benefit from meta-learning (warm starts) using malware dataset characteristics, reducing expensive evolutionary evaluations.

Lundberg & Lee (SHAP model explanations) proposed SHAP, a unified framework for model-agnostic explanations built on Shapley values to attribute predictions to features. Produces consistent, additive feature attributions that can be computed for many model types (tree-aware optimizations exist). Demonstrated interpretability across domains and showed utility for debugging models. SHAP explanations can be computationally heavy for large models/datasets; careful presentation is required for analyst consumption. For security contexts, Smart TPOT must produce explainable models; integrating SHAP (or lightweight explainers) into discovered pipelines helps analysts trust and act on alerts.

3. Methodology

3.1. Dataset Description

Android based malware dataset has been gathered from the open-source Kaggle website, in which data comprises permission details of approximately 30,000 applications. Various Android malware labels, such as backdoor, banker, dropper, fileinfector, PUA, ransomware, scareware, SMS, spy, Trojan, and zero day, include 183 parameters. As a whole, the entire dataset consists of 29,999 samples, of which 20,000 are malignant-based applications, and the remaining 9,999 are standard Android applications. The dataset link is mentioned below.

3.2. Preprocessing

For convenience, details are imported as a data frame from a CSV file. The dataset is filtered to remove unnecessary features. To better comprehend and analyze the data, multiple graphs are constructed. When data is examined for zero or inaccurate principles, the column mean is used in their place. Following data analysis based on the distribution of benign and malicious programs in different environments, multiple graphs were created to illustrate the findings. Plotting and visualization are done with Matplotlib and Seaborn. All columns containing data other than permissions were eliminated.

App names were mapped to the index for convenient access to the data. Following preprocessing, the data is divided into training and testing sets in an 80:20 ratio. We performed both under- and across-sampling on the dataset, but the final results do not seem encouraging. We employed a variety of classifiers after the sample, such as AdaBoost, SVM, KNN, Random Forest, ExtraTree, and Gradient Boosting. Though it takes the longest to complete, the results are accurate and satisfying. Applying the classifiers to the given dataset is now our responsibility. We started by using Random Forest, which significantly increased the provided accuracies. Then, to improve their prediction accuracy, we applied the Boosting technique. After selecting reliable features, we employed the Boosting technique to improve the accuracy of their predictions. We applied the boosting approach immediately after picking Trustworthy attributes, while the outcomes indicate that the framework is increasing. Eventually, we utilized SVM and KNN on the final dataset and received our best results. Comparing the outcomes following feature selection reveals that we have improved and achieved greater accuracy.

3.3. Feature Selection

Choosing the optimal features from the Android malware dataset in order to achieve the most significant outcomes represents one of the primary responsibilities entailed in any supervised machine learning endeavor [14]. The significant approach for choosing such traits is the Chi-Square Test. Here, this test automatically chose 20 features from the malware database, which leads to noise reduction and performance enhancement. Several variations are statistically tested using the Chi-Square test to ascertain how variances are closely related to each other. It makes the assumption that the specified distributions are unbiased in the null Hypothesis. Thus, by identifying the characteristics that have the most significant influence on the output class label, this evaluation can be employed to identify the most outstanding qualities for the Android malware dataset. For every parameter in Android malware data, Chi-Square $\chi^2$is evaluated, subsequently arranged in decreasing order based on χ^2 value. When χ^2 increases, the outcome literally depends on the parameter; also, increasing the parameter leads to greater significance in recognizing outcomes.

Feature selection can be estimated using an Equation (1)

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{1}$$

C represents degrees of freedom, O and E represent Observed frequency and estimated frequency values, respectively. For every parameter, a table, namely a contingency table, is constructed with n rows and m columns. Each field (i, j) represents the quantity of rows that have feature as 'i' with labelling as 'j'. Hence, every field in the mentioned table is indicated as an observed frequency value. Finally, the expected frequency for every field calculation is done by taking a fraction of the feature value in malware metadata, and then multiplying by the entire quantity of class label [15]. The importance of research focused on choosing static features, such as API calls and permissions, for detecting Android malware provides less overhead with greater efficiency, and a reduction in computational overhead as well [16-17].

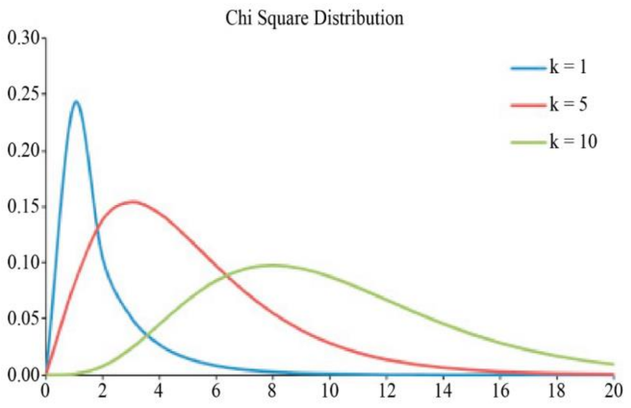

Fig. 3 Chi-Square distribution

A Chi-Square distribution may have a lengthy tail that points toward the distribution's important values, or it may skew to the right. The number of degrees of freedom in a particular task will determine the distribution's general form. The sample size is one greater than the degree of freedom. Such a distribution is a continuous distribution with degrees of freedom as shown in Figure 3.

3.4. Machine Learning Trained Models

To detect Android malware applications, we proposed machine learning classifiers.

3.4.1. Support Vector Machine (SVM)

A popular supervised learning model with related learning methods for analyzing sparse and high-dimensional data and identifying patterns is the Support Vector Machine [18]. Training and forecasting are the two processes that make up the classification challenge in general. We must split the entire data space into training and testing sets in order to categorize the dataset into two distinct subsets made up of positive and negative samples. The training set D (input space) of N pairings (x_i, y_i), where i = 1,..., N, is used throughout the training procedure, which can be stated to ensue, assuming two distinct classes are taken into consideration.

Let us concentrate on the linear classifier, which was the initial SVM model created by [19]. Suppose that a vector of dimensions w and the constant b configure each hyperplane in R_n.

w^Tx +b=0 where w represents an orthogonal vector in hyperplanes.

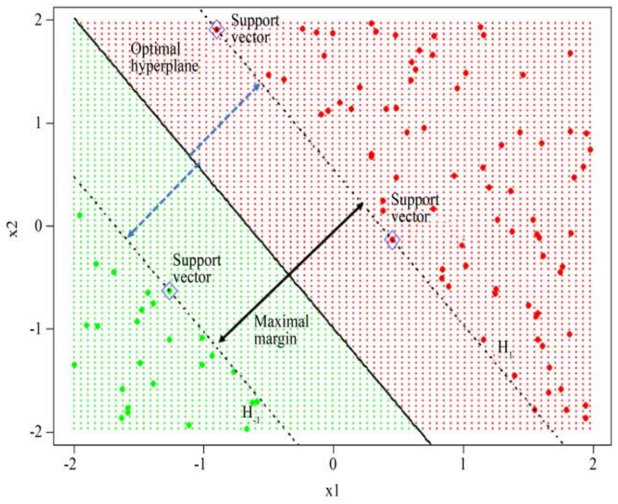

Fig. 4 Support vector machine with a hyperplane

From Figure 4 above, three spots that correspond to the dotted lines and are spaced equally apart from the greatest margin hyperplane. The most excellent margins hyperplane is "supported" by these three locations (each class must have at least one support vector).

Stated otherwise, the ideal hyperplane might shift if a support vector's position changed even slightly, whereas the ideal hyperplane remained unchanged if its adjacent points changed. Moreover, two boundaries of optimal hyperplanes are noted as H_1 and $H_{-1.}$

$$H_1\colon \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 = 1$$

$$H_{-1}\colon \alpha_0 + \alpha_2 x_2 + \alpha_2 x_2 = -1$$

In positive cases, data points are considered as equation.

$$H_1\colon \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 \geq 1$$

Data points available in negative cases are denoted as equation.

$$H_{-1}\colon \alpha_0 + \alpha_2 x_2 + \alpha_2 x_2 \geq -1$$

3.4.2. Gradient Boosting

One kind of supervised machine learning technique that builds an ultimate model by combining several inadequate learners is called gradient boosting. In order to develop such models, it progressively minimizes the loss of function by giving instances with incorrect predictions greater weights. The model's error rate is represented by the difference between the weak learners' predictions and actual values. The gradient, which determines the direction for model parameter adjustment in the following training cycle, is computed using this error rate. The entire architecture of the gradient boosting model is depicted in Figure 5, which shows how the models are trained, and predictions are evaluated among actual and predicted values [20].

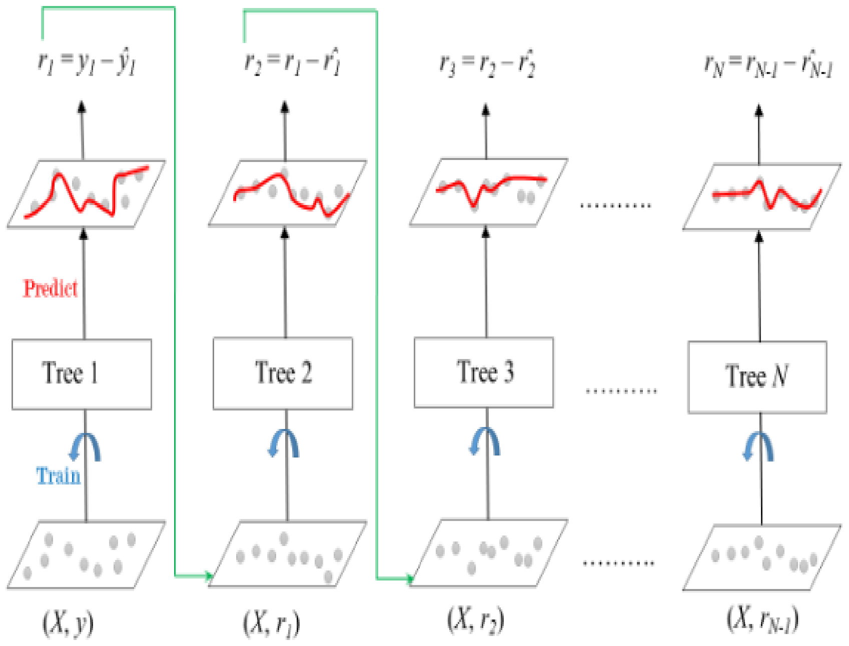

Fig. 5 Illustration of a gradient boosting classifier

3.4.3. Random Forest

Numerous individual learners are produced by an ensemble learner approach, which then aggregates the outcomes. The bagging technique is extended by the Random Forest method. In Bagging, a bootstrap sample of the input data is used to build each classifier separately. Every feature property is considered when deciding a node split in a standard decision tree classifier. However, Random Forest uses a random subset of features to determine the optimal parameter at each decision tree node.

Random Forest is less susceptible to intrinsic error in the data because of this randomized choice of features, which additionally assists in expanding effectively when there are numerous characteristics per feature vector and lessens the interdependence (correlation) among feature characteristics [21, 24].

3.4.4. K-Nearest Neighbors (KNN)

The k-NN classifies data samples to forecast Android based on the k training data points closest to the item. Using training samples and their characteristics, this method aims to classify Android malware. It is straightforward to use, much like the clustering technique, which groups new data based on its distance from current data or its nearest neighbor. Before determining the distance between the data and its closest neighbor, it must first determine the importance of the surrounding k neighbors.

The distance between two points, or the training and testing sites, is then calculated using the Euclidean formula. The formula Eucldist (x, y), which is expressed as equation (2), represents the Euclidean distance. Here, "a" stands for the initial data, "b" for the subsequent data of a feature, and "n" for the number of features selected following the feature selection phase.

$$Eucl_{dist}(x, y) = x_0 + \sum_{i=0}^{n}(a_i + b_i)^2$$

3.5. AdaBoost

Boosting is a well-liked technique for learning classifiers by turning an unsuccessful learner into an effective learner. The goal of the boosting strategy is to start with a poor classifier and build a very favorable classifier by enhancing the predictive capabilities of the weak classification algorithm. This forecast is prepared by equalizing the results of many poor classifiers. According to [22], AdaBoost, also known as Adaptive Boosting, is a popular boosting strategy that addresses classification problems by building a strong classifier from a large number of weak classifiers. In order to solve the flaws of the first model, a second model is constructed after a prototype is created, utilizing the training set of data.

Models are added until the training set or the maximum number of models is established. A popular method for enhancing decision tree performance on binary classification tasks is AdaBoost. Because it increases the effectiveness of machine learning techniques, the AdaBoost algorithm was chosen. It is typically quite good when used with less proficient students. With AdaBoost, the conventional approach is one-level decision trees. Because the trees are tiny and only hold one option for a class, they are known as decision stumps. Using the weighted training ransomware data, weak models are created and concatenated sequentially.

3.5.1. Extra Tree Classifier

The Extra Trees Classifier is a simple ensemble-based selection tree training technique. For the Extra Trees calculation, [23] generate a large number of unpruned selection trees using preprocessed characteristics from metadata. The essential properties at each split purpose of the selection tree will be randomly tested using the Extra Trees computations, which are comparable to random forest computations. Similar to Random Forest, Extra Trees generates a vast number of trees and related hubs using randomized selections of highlights. Inconsistency in Extra Trees is not caused by data upgrading, but rather by the randomized bits of everything being equal. Extra-Trees work by both lowering change and raising propensity. The intersection point is chosen at random using the Extra Trees computations.

3.5.2. Tree-Based Pipeline Optimization Tool

Tpot is a Language mechanized machine learning module that optimizes the machine learning workflow by applying the ideas of genetic programming. By carefully examining millions of potential parameters to determine which one best fits your data, it automates the most laborious aspect of machine learning. A free and open-source AutoML program called TPOT (Tree-based Pipeline Optimization program) [25] streamlines the process of pipeline optimization for machine learning. The optimal machine learning pipelines, encompassing data pretreatment, choosing features, selecting models, and hyperparameter tweaking, are automatically found by TPOT using genetic programming.

A Python-based machine learning tool called Python Optimised ML Pipeline (TPOT) uses genetic programming to maximize network throughput [26-27]. To retrieve static information like permissions, network calls, API calls, and system traffic from the malicious apps for Android dataset, we employ TPOT to construct models. Our proposed framework is depicted in Figure 6.

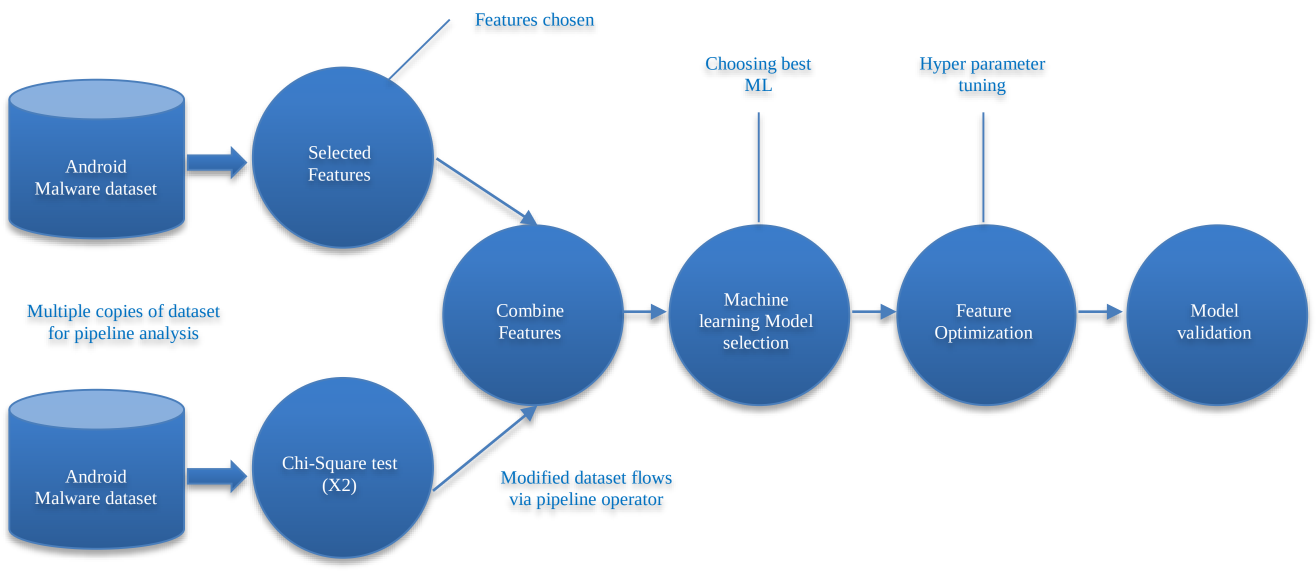

Fig. 6 Proposed framework of AutoML-based TPOT

Pseudo Code for TPOT

```
FUNCTION AutoML(data)
 Step 1: Data Preprocessing
data_cleaned = clean_data(data)
features, target = extract_features_and_target(data_cleaned)
train_data, test_data = split_data(features, target, ratio=0.8)
Step 2: Model Selection
model_list = get_available_models()
best_model = NULL
best_score = $-\infty$
Step 3: Hyperparameter Tuning
   FOR model IN model_list:
hyperparameters = get_hyperparameter_space(model)
```

```
optimized_model = tune_hyperparameters(model, train_data, hyperparameters)
 Step 4: Model Training
trained_model = train_model(optimized_model, train_data)
 Step 5: Model Evaluation
score = evaluate_model(trained_model, test_data)
    # Step 6: Select Best Model
    IF score >best_score:
best_score = score
best_model = trained_model
Step 7: Deploy Best Model
deploy_model(best_model)
  RETURN best_model, best_score
```

3.5.3. Process of the Proposed Workflow

The entire workflow of the proposed work in Android malware detection is shown in the Figure 7.

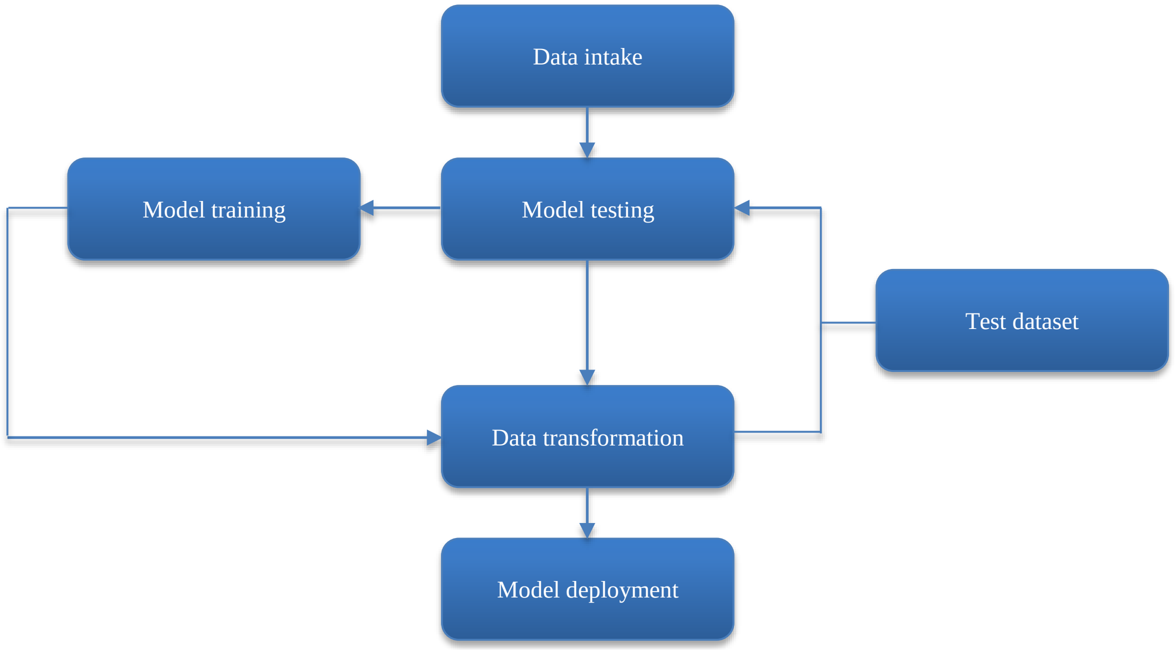

Fig. 7 Workflow for Proposed Model

4. Experimental analysis

4.1. Performance Evaluation

Precision, recall, f-score, and accuracy are the metrics used to assess the overall performance of machine learning models during both the training and testing phases.

Ideally classified Android malware samples fit the actual class as Truly Positive (TP), the number of samples that were incorrectly classified as False Positive (FP) fit the actual value, the predicted value of data that were incorrectly classified as unfit to the class Truly Negative (TN), and the output value of Android malware that was incorrectly classified as unfit to actual values as False Negative (FN).

Precision- Precision indicates the ratio of truly identified positive samples from predicted positive samples, described as equation (3).

$$Precision = \frac{Truely\ predicted\ as\ positive}{True\ Positive + False\ Positive} \quad (3)$$

Recall- calculated as the ratio of truly predicted samples as positive to the combination of valid positive and false negatives using equation (4).

$$Recall = \frac{Truely\ predicted\ as\ positive}{True\ Positive + False\ Negative} \quad (4)$$

Accuracy- Accuracy represents the prediction of Android malware correctly for the entire input sample. It is defined as the total no of correctly predicted samples divided by the total no of samples.

$$Accuracy = \frac{Number\ of\ samples\ truely\ predited\ as\ android}{Overall\ input\ data\ samples} \quad (5)$$

F1 Score- F1 score is defined as the harmonic mean of precision and recall values.

$$F - Score = 2 * \frac{Precision \times Recall}{Precision + Recall} \quad (6)$$

Various metrics, such as accuracy in Figure 8, precision in Figure 9, Recall in Figure 10, and F1 score in Figure 11, are compared with six machine learning classifiers.

To enhance overall efficiency, the authors used TPOT, a tool in Python that automatically chooses a suitable classifier to reduce manual errors and computational complexity.

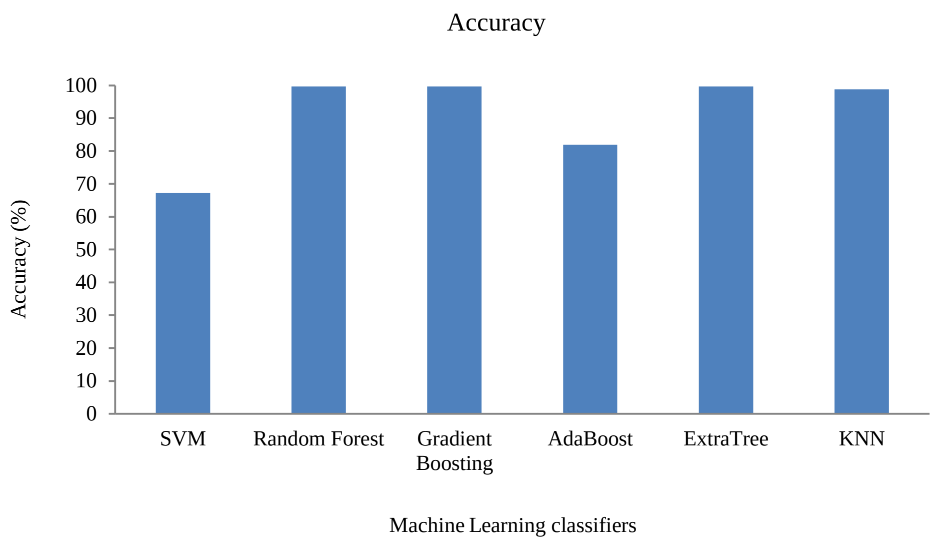

Fig. 8 Performance evaluation based on accuracy

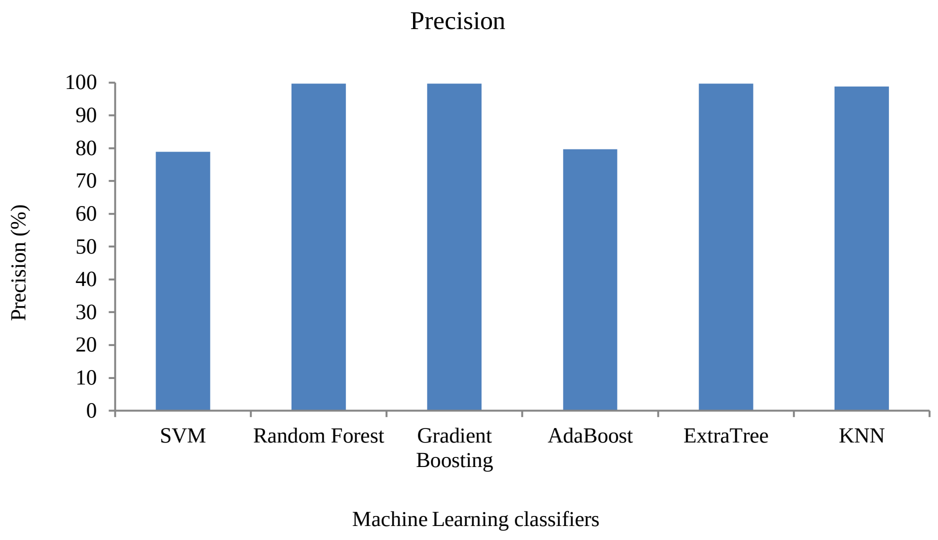

Fig. 9 Performance evaluation based on precision

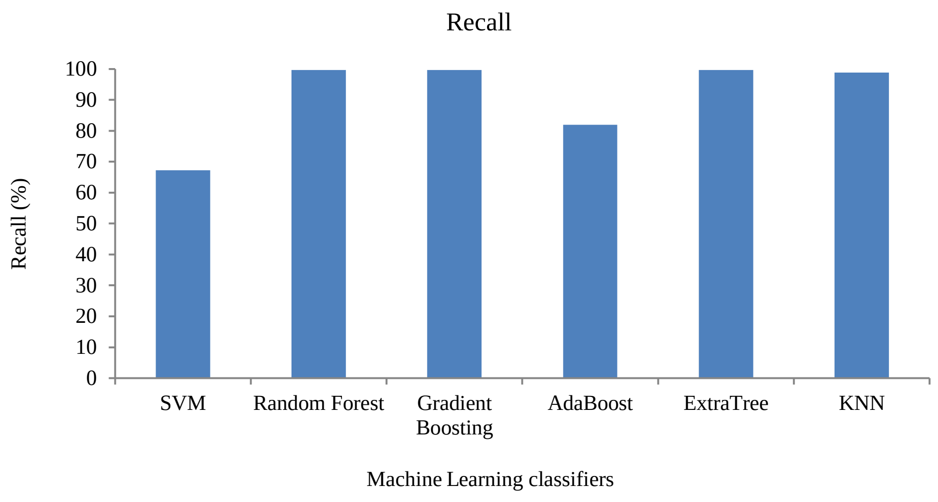

Fig. 10 Performance evaluation based on recall

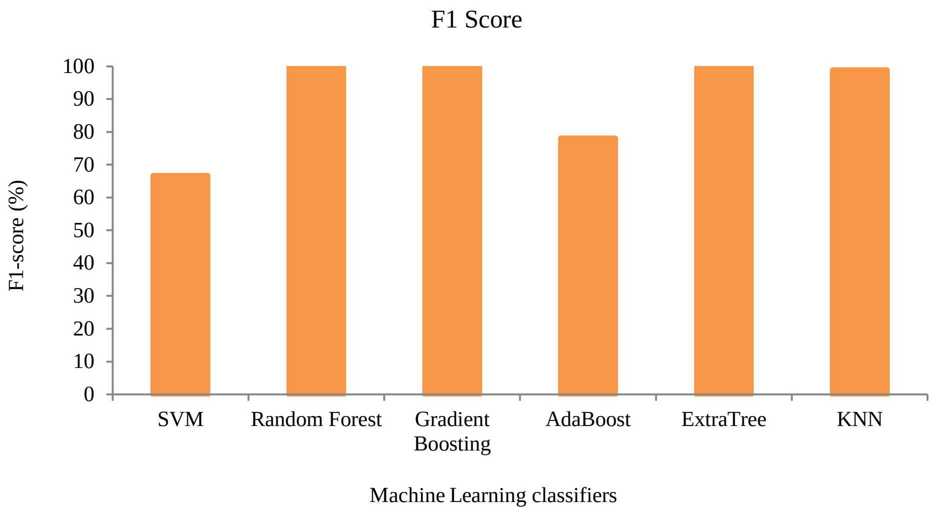

Fig. 11 Performance evaluation based on F1-score

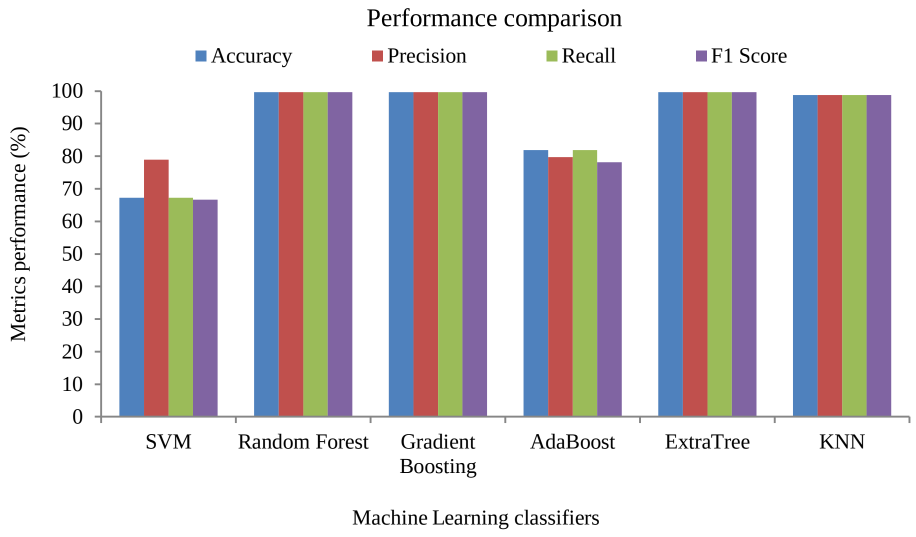

Fig. 12 Overall performance comparison

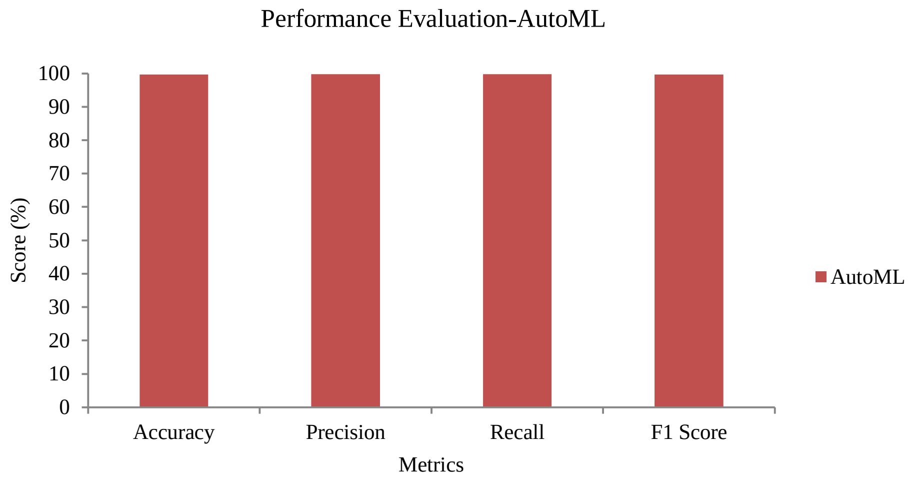

Fig. 13 AutoML classifier for android malware

4.2. AutoML Classification for Android Malware Detection

Here, we implemented AutoML-based TPOT for predicting Android malware with static as well as dynamic features. The comparison bar chart is depicted in Figure 13.

Table 1. Assessment of various machine learning models

Machine learning models	**Accuracy**	**Precision**	**Recall**	**F1-score**
KNN	98.8	98.8	98.8	98.8
Extra Tree	99.7	99.7	99.7	99.7
Gradient Boosting	99.7	99.7	99.7	99.7
AdaBoost	81.9	79.7	81.9	78.1
SVM	67.2	78.93	67.2	66.7
Random Forest	99.7	99.7	99.7	99.7
TPOT-based AutoML	99.7	99.78	99.78	99.7

4.3. Discussion

Machine learning model selection, development, and tweaking are all automated using Automated Machine Learning (AutoML). It seeks to minimize human error in processes like preliminary data processing, selecting features and engineering, selecting a model, tuning hyperparameters, and assessment of models, while also making Machine Learning (ML) approachable to those who are not experts. Data analysts manually pick techniques, adjust parameters, and maximize feature optimization when using conventional machine learning.

In this research work, we used six machine learning models in which data collection, preprocessing, feature selection, and a suitable model were chosen, and finally, validation was done manually. In this case, the maximum possibility of human errors occurred, training time for Android-based samples takes the maximum time, and selecting a suitable model is not appropriate in a few cases. To avoid such errors, we applied TPOT-based AutoML classifiers, which performed everything automatically with minimum computational time and a lack of errors. Hence, our proposed TPOT-based AutoML provides better outcomes of 99.7% accuracy, 99.7% precision, 99.6% recall, and 99.7% F1-score in the prediction of Android-based malware in smart devices.

Benefits of our proposed TPOT classification tool:

- Android-based malware might be detected prior to implementation so that we can avoid scratches. Moreover, helpful for verifying app downloads prior to installation.

Millions of applications may be effectively examined as persistent characteristics (such as permissions, API calls, and opcodes) could be swiftly retrieved. Also, this makes it easy to integrate with extensive malware systems.

- It functions despite the need for a software emulator, digital gadgets, or an actual Android device. It remains more affordable because it may be done remotely.

5. Conclusion

The detection and monitoring of Android malware is the main topic of this investigation. The TPOT framework, an automated hybrid analysis framework for Android malware detection, was proposed in this research. 30,000 Android apps were used to test this framework, which includes both static and dynamic features. In summary, the authors used various machine learning based algorithms to achieve a maximum accuracy of 99.7% with a high training rate, more manual efforts during the entire phase till validation, with a maximum time period.

Hence, we introduced the TPOT classifier tool, which is an Auto ML tool that helps in optimal machine learning pipelines, encompassing data pretreatment, choosing features, selecting models, and hyperparameter tweaking. is automatically found by TPOT using genetic programming, which attained a similar accuracy of 99.7% leading to progress in this research work for Android-based malware detection.

The comparative report provided by the researchers helped them advance this field. The findings clearly demonstrate that the framework outperformed manually trained machine learning algorithms in terms of accuracy when using the TPOT automated machine learning model.

References

[1] Min Yang et al., “An Android Malware Detection Model based on DT-SVM,” *Security and Communication Networks*, vol. 2020, no. 1, pp. 1-11, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[2] Kaijun Liu et al., “A Review of Android Malware Detection Approaches based on Machine Learning,” *IEEE Access*, vol. 8, pp. 124579-124607, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[3] Mohammed N. AlJarrah et al., “A Context-Aware Android Malware Detection Approach using Machine Learning,” *Information*, vol. 13, no. 12, pp. 1-25, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[4] Esra Calik Bayazit, Ozgur Koray Sahingoz, and Buket Dogan, “Malware Detection in Android Systems with Traditional Machine Learning Models: A Survey,” *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, Ankara, Turkey, pp. 1-8, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[5] A. Ananya et al., “SysDroid: A Dynamic ML-based Android Malware Analyzer using System Call Traces,” *Cluster Computing*, vol. 23, no. 4, pp. 2789-2808, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[6] Modupe Odusami et al., “Android Malware Detection: A Survey,” *Applied Informatics: First International Conference*, Bogotá, Colombia, pp. 255-266, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[7] Francisco Nauber Bernardo Gois, Joao Alexandre Lobo Marques, and Simon James Fong, *TPOT Automated Machine Learning Approach for Multiple Diagnostic Classification of Lung Radiography and Feature Extraction*, Computerized Systems for Diagnosis and Treatment of COVID-19, pp. 117-135, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[8] Nayana N. Murthy, and K. Thippeswamy, “TPOT with SVM Hybrid Machine Learning Model for Lung Cancer Classification using CT Image,” *Biomedical Signal Processing and Control*, vol. 104, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[9] Amarjyoti Pathak, Utpal Barman, and Th. Shanta Kumar, “Machine Learning Approach to Detect Android Malware using Feature-Selection based on Feature Importance Score,” *Journal of Engineering Research*, vol. 13, no. 2, pp. 712-720, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[10] Matilda Rhode, Pete Burnap, and Kevin Jones, “Early-Stage Malware Prediction using Recurrent Neural Networks,” *Computers& Security*, vol. 77, pp. 578-594, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[11] Abdulsamad E. Yahya et al., “A Novel Hybrid Deep Learning Model for Detecting and Classifying Non-Functional Requirements of Mobile Apps Issues,” *Electronics*, vol. 12, no. 5, pp. 1-22, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[12] Seungho Jeon, and Jongsub Moon, “Malware-Detection Method with a Convolutional Recurrent Neural Network using Opcode Sequences,” *Information Sciences*, vol. 535, pp. 1-15, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[13] Abdulghani Ali Ahmed et al., “Deep Learning-based Classification Model for Botnet Attack Detection,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 13, no. 7, pp. 3457-3466, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[14] Jaehyeong Lee et al., “Android Malware Detection using Machine Learning with Feature Selection based on the Genetic Algorithm,” *Mathematics*, vol. 9, no. 21, pp. 1-20, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[15] Hee-Yong Kwon, Taesic Kim, and Mun-Kyu Lee, “Advanced Intrusion Detection Combining Signature based and Behavior-based Detection Methods,” *Electronics*, vol. 11, no. 6, pp. 1-19, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[16] Durmuş Özkan Şahin et al., “A Novel Permission-based Android Malware Detection System using Feature Selection based on Linear Regression,” *Neural Computing and Applications*, vol. 35, pp. 7, pp. 4903-4918, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[17] K. Santosh Jhansi, Sujata Chakravarty, and P. Ravi Kiran Varma, “Feature Selection and Evaluation of Permission-based Android Malware Detection,” *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, Tirunelveli, India, pp. 795-799, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[18] Yuxin Ding et al., “Android Malware Detection Method based on Bytecode Image,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 5, pp. 6401-6410, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[19] Ya Pan et al., “A Systematic Literature Review of Android Malware Detection using Static Analysis,” *IEEE Access*, vol. 8, pp. 116363-116379, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[20] C.W. Colaco et al., “Defensedroid: A Modern Approach to Android Malware Detection,” *Strad Research*, vol. 8, no. 5, pp. 271-282, 2021. [Google Scholar]

[21] Abdelouahab Amira et al., “TriDroid: A Triage and Classification Framework for Fast Detection of Mobile Threats in Android Markets,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 2, pp. 1731-1755, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[22] A. Jyothish, Ashik Mathew, and P. Vinod, “Effectiveness of Machine Learning based Android Malware Detectors against Adversarial Attacks,” *Cluster Computing*, vol. 27, no. 3, pp. 2549-2569, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[23] Rinanza Zulmy Alhamri et al., “Supervised Learning Methods Comparison for Android Malware Detection based on System Calls Referring to ARM (32-bit/EABI) Table,” *Journal of Information Technology and Cyber Security*, vol. 2, no. 1, pp. 15-24, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[24] Fadare Oluwaseun Gbenga, Adetunmbi Adebayo Olusola, and Oyinloye Oghenerukevwe Elohor, “Towards Optimization of Malware Detection using Extra-Tree and Random Forest Feature Selections on Ensemble Classifiers,” *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 9, no. 6, pp. 223-232, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[25] Zhenshuo Chen, Eoin Brophy, and Tomas Ward, “Malware Classification using Static Disassembly and Machine Learning,” *arXiv preprint*, pp. 1-9, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[26] G. Revathy et al., “CancerAI: A Deep Learning Framework for Ovarian Cancer Prediction,” *2024 5th International Conference on Electronics and Sustainable Communication Systems (ICESC)*, Coimbatore, India, pp. 1318-1323, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[27] Jianting Yuan et al., “Malicious URL Detection based on a Parallel Neural Joint Model,” *IEEE Access*, vol. 9, pp. 9464-9472, 2021. [CrossRef] [Google Scholar] [Publisher Link]