
RESEARCH ARTICLE**A Database-Driven Explainable AI Framework for Android Malware Detection Using a Comparative SHAP-Based Machine Learning Analysis****Yalmaz Najmaldeen Taher ¹, Sarmad N. Mohammed ²✉, Mohammed Rashad Baker ²**¹ *College of Computer Science and Information Technology, Computer Science Department, Kirkuk University, Kirkuk, Iraq*² *College of Computer Science and Information Technology, Artificial Intelligence Department, Kirkuk University, Kirkuk, Iraq***Corresponding Author:** Sarmad N. Mohammed, **E-mail:** sarmad_mohammed@uokirkuk.edu.iq

ABSTRACT

As sophisticated Android malware continues to proliferate, detection frameworks must achieve high predictive accuracy, decision transparency, and computational efficiency. While complex ML models may provide a high classification performance, they are not always easily interpretable which can be a constraint to operational trust in security-critical mobile environments. In this work, we present an explainable static malware triage framework, which is evaluated on Drebin-215 malware benchmark dataset. In this study, five model families were tested: Logistic Regression (LR), Random Forest (RF), XGBoost, LightGBM, and a Multi-Layer Perceptron (MLP) Neural Network, all of which were tested with leakage-free 5-fold Stratified Cross-Validation. Empirical results indicate that the MLP Neural Network achieved the highest mean Recall (0.9827 ± 0.0056) and F1-Score (0.9847 ± 0.0026); however, XGBoost was able to execute more than 11× faster during the 5-fold cross validation evaluation (3.34 s vs 38.29 s), while also having a fast single sample inference latency (0.7647 ms) and small serialized model footprint (0.21 MB), which indicates competitive predictive performance. TreeExplainer was used with pooled Out-of-Fold (OOF) cross-validation predictions (N = 15,036) to obtain global and local SHAP explanations to enhance model transparency. The pooled FN subset (NOOF-FN = 112) indicated no presence of SEND_SMS across all these samples and the feature patterns for SEND_SMS and READ_PHONE_STATE indicated a negative mean SHAP contribution, which is more likely to push the XGBoost output towards the benign class. In general, the model encompasses comparative predictive assessment, OOF explainability, and computational efficiency analysis in a mobile malware triage environment supported by a database.

KEYWORDS

Android Security; Malware Detection; Machine Learning; Explainable AI (XAI); XGBoost; SHAP; Database

ARTICLE INFORMATION**ACCEPTED:** 15 September 2026**PUBLISHED:** 02 October 2026**DOI:** 10.32996/bjmss.2026.4.3.2

1. Introduction

Malware remains a major challenge to digital security because of the continuously evolving nature of cyber threats. Although traditional signature- and heuristic-based detection methods have proven effective to identify known attacks, they can be limited in identifying modern attacks with features such as obfuscation, polymorphism and zero-day exploits. These evolving attack techniques present continuing challenges for conventional malware detection approaches [10], [27]. This has led to the development of Machine Learning (ML) and Artificial Intelligence (AI) that allows for automatic learning and generalization of malicious behavior without the need for a priori rules [6].

In the last few years, ML and Deep Learning (DL) models have demonstrated their potential in enhancing the detection's performance and flexibility for various cybersecurity applications including intrusion detection, spam filtering, and malware classification [3], [13]. They can analyze vast amounts of high-dimensional data, which aids in identifying intricate attack patterns that may not be discernible by human analysts or traditional security solutions [21]. However, difficulties like adversarial evasion, imbalance in the data sets, and model scalability are yet to be resolved for their implementation on a large scale [23], [28].

One of the difficulties with complex ML models is they are not very easy to interpret. In some cases, the reasoning behind the predictions of some high-performing models is not easily interpretable and these models are called black-box (BB) models. Such

opacity can decrease user trust and restrict their use in high-stakes decision-making situations [25]. To overcome this limitation, there is a novel concept called Explainable Artificial Intelligence (XAI). The main goal of XAI methods is to increase transparency in the decision-making process of ML models, helping to make sense of why an application is considered malicious [2], [8].

The transparency of these decisions is important for security forensics, compliance and defence against adversarial evasion [12]. Driven by these operational needs, this paper compares and conducts an interpretability audit of explainable ML frameworks for Android malware detection [18]. Instead of introducing new architectures which involve static information and post-hoc explainers, our work is about how to operationalize explainable static triage under real-world deployment constraints. We consider five families of models: Logistic Regression (LR), Random Forest (RF), XGBoost, LightGBM, and Multi-Layer Perceptron (MLP) Neural Network under leakage free 5-fold Stratified Cross-Validation. Importantly, we perform post-hoc SHAP feature attributions on pooled Out-Of-Fold (OOF) predictions on the entire 15,036 dataset samples, and then perform localized error attributions on pooled misclassified malware instances (NOOF-FN = 112) to find feature-omission evasion patterns, and quantify Pareto trade-offs between classification metrics, memory footprint and sub-millisecond inference latency.

The following is the organization of the remainder of this manuscript. The review of related literature is included in Section 2, and methodology including the dataset, ML models and explanation of the SHapley Additive exPlanations (SHAP) is included in Section 3. Experimental results are reported and analyzed in Section 4 and the performance and interpretability of the models is compared. Finally, in Section 5, we summarize our results and give future directions for research.

2. Related Works

Because of the weaknesses of traditional signature-based detection techniques, the use of ML and DL in cybersecurity has been well-studied. Moamin et al. [24] provided an overview of AI approaches for malware and network intrusion detection systems and divided them into three categories: supervised, unsupervised and reinforcement learning. The research also looked at DL architectures such as CNNs, RNNs, and LSTMs, and malware analysis methods (static, dynamic, and hybrid analysis). Khan et al. [20] discussed about the XAI based intrusion detection systems for Industry 5.0, highlighting its significance in enhancing the transparency of cyber security systems powered by ML.

In recent years, the importance of XAI for enhancing transparency and trust in cybersecurity applications has also been highlighted in the literature. Kuppa and Le-Khac [22] explored black-box attacks on XAI methods in cybersecurity and assessed the effectiveness of XAI gradient-based explainers under adversarial scenarios. Alperin et al. [5] discussed the interpretability of ML-based vulnerability risk scores for cybersecurity practitioners by using XAI and visualization techniques. With this line of thought, [31] and [14] offered a thorough review of XAI applications in cybersecurity, including intrusion detection, malware detection, phishing/spam detection, botnet detection, and fraud detection, as well as research directions and challenges. Charmet et al. [15] studied the use of XAI in cybersecurity and the security of XAI methods, such as attacks on XAI pipelines and countermeasures.

There have also been studies on explainability in more specific cases of malware detection. Karn et al. [19] concentrated on cryptomining detection in Kubernetes pods using system calls and ML models, while explaining the results of the classification using SHAP, LIME and an auto-encoding-based approach for LSTM models.

With a growing interest in adversarial robustness and feature interpretability in the area of malware detection in recent years. In [26] the researchers used recursive feature elimination with cross validation (RFECV) for ransomware detection and explored the importance of explainability for determining the influential features. In [4] the authors proposed a transfer-learning methodology for learning obfuscated malware from memory-dump data and used SHAP to provide explanations for the model predictions. Bae [9] analyzed the interpretability of malware classification models for heterogeneous data and pointed out the weaknesses of the current methods of providing explanations, suggesting a way to make feature explanations more robust. Likewise, the authors in [17] introduced a framework for explaining malware detection based on GRU and LSTM models, considering adversarial perturbations. SHAP and LIME were integrated into the framework for giving explanations at the global level and at the local level, respectively.

For Android malware detection, Arp et al. [7] proposed DREBIN, a framework for malware classification based on static features, including permissions, API calls, and hardware features, and the framework offered reasons for their classification decision. Later, Alani and Awad [1] proposed PAIRED, which used the Drebin-215 dataset to classify static Android malware with SHAP interpretability. The studies here show the importance of static features and explainability for detecting Android malware. However, the studies reviewed here do not all touch on the issues of multi-model performance and computational benchmarking, OOF attribution analysis of misclassified malware samples, and inference efficiency in a database-supported mobile security triage environment. The present work builds on these earlier studies to assess several ML models, conduct pooled OOF attribution analysis on 112 malware instances that were misclassified, and also explore runtime efficiency for real-time mobile security triage.

3. Methodology

The methodology used in this study is depicted in Fig. 1 which depicts the sequential steps which are used for Android malware detection and interpretability using XAI techniques. The proposed methodology involves five key steps: (i) Data acquisition, (ii)

Data preprocessing and feature preparation, (iii) Model training and performance comparison, (iv) Model evaluation and interpretability assessment, and (v) SHAP explainable analysis. The goal of each stage was to ensure that Android applications were properly classified as benign or malicious with transparency, giving actionable insights to the ML models' decision-making process.

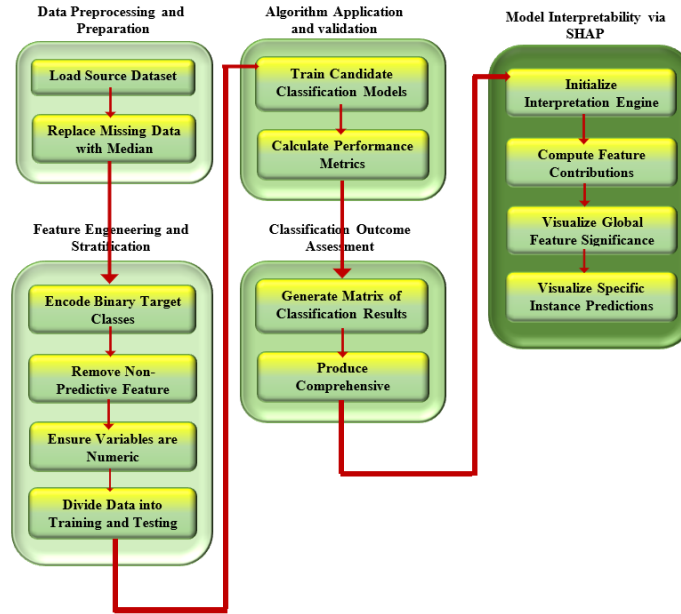


Figure 1. Methodological flowchart of the proposed approach.

3.1 Data Source and Description

This study uses the benchmark Drebin-215 dataset [7] which was collected by Arp et al. between 2010 and 2012. It contains 15,036 total Android applications, 9,476 benign samples (63.0%) and 5,560 malicious samples (37.0%) from 179 different malware families. The dataset is a manually extracted, static binary feature table of 215 boolean feature dimensions totaled across 8 broad feature categories, such as requested permissions, intent filters, restricted API calls, hardware components, suspicious URL endpoints from Android Manifest and disassembled DEX bytecode.

3.2 Data Preprocessing and Preparation

The initial step in the preprocessing process was to load the data into a Python environment using the Pandas library. The first step was to load the dataset into a Python environment with the help of the Pandas library. There were very few entries that were missing and these data were indicated with the character '?', and were identified and treated systematically as missing data. In order to preserve data integrity and prevent bias due to data omission, missing values were imputed by the median of their columns. This approach preserves the distributional characteristics of each feature and reduces the impact of outliers from the mean imputation approach.

Also, categorical labels which represented classes of applications were coded in numbers, where a benign sample had a value of 0 and a malicious sample had a value of 1. In addition, there was a non-numeric column in the data set called transact, which was dropped to ensure all input features were numeric and could be used by the selected ML algorithms. After cleaning data, the feature columns were transformed into a numeric type of data, which was uniform to further modeling.

Data preprocessing was directly integrated within scikit-learn evaluation pipelines (Pipeline) to guarantee a strict statistical evaluation and prevent data leakage between the training and evaluation subsets. Median feature value was used to impute missing values, which were represented as missing values across the 215 Drebin features. Importantly, the median imputer was fit within each cross validation fold, and then applied to transform the validation fold, so that the data used for validation was never seen in the training fold during preprocessing.

3.3 Model Training and Performance Comparison

A comprehensive and meaningful Android malware classification system was constructed by systematically testing five machine learning algorithms: Logistic Regression (LR), Random Forest (RF), XGBoost, LightGBM, and a Multi-Layer Perceptron (MLP) Neural Network. To get a wide range of learning paradigms, the algorithms were selected to represent linear modelling, bagging ensembles, gradient boosting and deep neural architectures, allowing for a thorough examination of the way in which each of

these structures performs on high-dimensional Android static data. Standard metrics (Accuracy, Precision, Recall, F1-Score, and ROC-AUC) were used to evaluate the dataset under the leakage-free 5-fold Stratified Cross-Validation. Lastly, the Pareto-optimal model balancing predictive recall, sub-millisecond inference latency, and minimal disk footprint (XGBoost) was subjected to detailed explainability analysis using pooled Out-Of-Fold (OOF) SHAP values to gain insight into the decision boundaries and evasion patterns. The role, theoretical underpinnings, and functioning of each of these models are discussed below.

3.3.1 LR

LR was used as the baseline classifier since it is simple, transparent and highly interpretable. The classification of malware can be used as a baseline benchmark [7]. It is a probability model that models the probability that an application is malicious, and the linear combination of the inputs of the features is passed into a sigmoid function that outputs the probability in the range of 0-1. The coefficients obtained in the course of training directly point to the direction and the extent of the impact that each feature has on the probability that has been predicted. LR can be used for Android malware detection to help determine which permissions, API calls, or intent attributes are highly correlated with behavior. LR is helpful because it gives us an idea of the significance of each characteristic and what the data might resemble. Despite the assumption that the data can be separated into a line and may not capture complex relationships, LR provides us with valuable insights. LR is also easy to understand making it simple to see how it makes decisions. This makes LR a good starting point to compare with a more complex model that uses many different models. LR serves as a foundation to understanding decision limits, which can then be helpful when considering more complex models that work together.

3.3.2 RF

To address relationships between features that are not straightforward and more complex interactions, which are often found in malware datasets [30]. RF is a learning method that uses decision trees built from random samples of the data and features, then it combines their results to produce a final answer. This helps to prevent the model from being closely fit to the data makes it work better with new data and can handle datasets with a lot of noise and many features, like the Drebin-215 data. RF is very helpful for discovering relationships between components of the Android manifest, such as permissions and intents, which individually may not be enough to classify as malware but are highly suggestive when combined. Also, it indicates the most important features. Although RF is more complex than some methods to understand, it is useful in combination with another method, called SHAP, which allows us to interpret the results and make decisions that can be communicated to other researchers and practitioners who might be interested. The importance of the RF method is that it can identify patterns of the data that are significant to making smart decisions about malware. The authors of this paper, RF and SHAP, collaborate to shed light on the impact of manifest features on malware classification.

3.3.3 Extreme Gradient Boosting (XGBoost)

The XGBoost model was selected due to its good performance. It is a model based on a technique called sequential decision tree building. This is to reduce the prediction error, for each iteration [11]. The model corrects the errors made by the previous ensemble by building trees. XGBoost model is able to learn the correlation among the various features in Android malware data. It uses two types of regularisation: L1 and L2, to prevent overfitting and improve generalisation. It can also deal with missing data, which is prevalent in security data sets. The other benefit is that it is efficient as it can be calculated in parallel. The XGBoost model was the focus of this study. It is able to learn relationship between permissions, APIs and behavioral indicators. These links can all be used to identify patterns. The XGBoost predictive framework along with the SHAP analysis is helpful for Android malware classification.

3.3.4 Light Gradient Boosting Machine (LightGBM)

LightGBM was introduced as an improved version of gradient boosting models, which is based on histogram tree, to compare the performance of these models with the traditional gradient boosting models [11]. LightGBM has a leaf-wise (best-first) tree growth strategy, that grows the tree by selecting the leaf with the maximum delta loss, which is able to further reduce the training loss and enhance the classification accuracy. LightGBM discretises continuous and boolean features into discrete bins using a histogram based algorithm to optimize memory usage and execution throughput of high dimensional security data. This Histogram Binning has a significant effect on speeding up the training process and memory usage, while preserving the key non-linear feature split boundaries for the 215 Drebin features. LightGBM with 100 boosting iterations, learning rate 0.1 and 31 max leaves is able to model complex interactions between Android permissions, intent filters and API calls. Combined with the post-hoc SHAP analysis, LightGBM offers a very efficient and scalable solution for real-time threat triage on mobile devices.

3.3.5 MLP

A fully connected feedforward MLP was built to compare the performances of tree-based ensemble architectures with a baseline DL model. The network architecture consists of an input layer with 214 input feature dimensions, 2 dense hidden layers with 100 hidden neurons in each, and an output layer with a single sigmoid output unit. The hidden layers learn feature representations

that are non-linear, using Rectified Linear Unit (ReLU) activation functions ($f(x) = \max(0, x)$) to avoid gradient vanishing and to enable the network to capture high order combinations of permissions and API calls. The output layer uses a logistic sigmoid function to convert hidden representations to binary prediction probabilities for class assignment (Benign vs. Malware). The Adam (Adaptive Moment Estimation) optimizer was used to optimize the model parameters with a fixed learning rate of 0.001, and the binary cross-entropy loss function was minimized for a maximum of 300 epochs. The MLP architecture can serve as a non-tree DL reference model, and it is critical to investigate whether deep neural feature abstractions can improve the performance over tree-based gradient boosting models [11] in this case.

3.4 Model Evaluation and Confusion Matrix Analysis

A comprehensive evaluation system was used to carefully consider the Android malware detection ML classifiers. The primary characteristics we would like to know about these classifiers are the accuracy, precision, recall and F1 score. The performance of models was evaluated based on the result of their decision via Confusion Matrix. This will enable us to learn the effectiveness of the Android malware detection machine learning classifiers. All of these are necessary to ensure the selected model has high overall correctness and good performance in both classes, especially in the case of class imbalance.

3.4.1 Evaluation Metrics

These common classification metrics were used for quantitative model performance analysis:

Accuracy measures how correct the overall model is how many of the instances (benign and malicious) are correctly predicted when looking at the total number of predictions. It is defined as:

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

High accuracy means the overall performance is good, but it's not necessarily a good sign if the data set is imbalanced.

Precision, or Positive Predictive Value, is the measure of the proportion of true positive (malicious apps) predictions to the total number of positive predictions. It is given by:

$$Precision = \frac{TP}{(TP + FP)}$$

High precision means that the model makes very few false alarms (also known as False Positives (FP)).

Recall refers to how well the model can identify all true positive instances, which is to correctly identify malicious applications. It is defined as:

$$Recall = \frac{TP}{(TP + FN)}$$

The higher the recall value, the less likely the model is to fail to identify malware samples. When (FP) and False Negatives (FN) are important, the F1-Score is a harmonic mean of Precision and Recall:

$$F - 1 \text{ Score} = 2 \times \frac{Precision \times Recall}{(Precision + Recall)}$$

These metrics give a general idea of the models' performance, accounting for the trade-offs between correct detections and classification errors. They offer a well-rounded view of predictive performance for the models assessed.

3.4.2 Computational Efficiency and Methodological Specifications

To guarantee the complete academic reproducibility of the results and to evaluate the feasibility of mobile real-time triage empirically, all models were trained and tested on the same hardware configuration: Intel Core i7 workstation (6 cores, 12 threads, 3.20 GHz), 32 GB of RAM, a 64-bit Windows operating system, Python 3.11.8, scikit-learn 1.7.2, XGBoost 3.1.2, and LightGBM 4.6.0. Stochastic operations were run with the random number seed fixed (random_state = 42) and the classification probability threshold of $\tau = 0.50$.

Table 1 lists the training time, the size of the model disk in the serialization format, inference time per sample and throughput for the inference of a batch of all the candidate models. The single-sample latency for LR was also the lowest (0.0935ms) and disk footprint was very low (< 0.01MB). XGBoost was a clear winner for non-linear ensemble models with only 0.534 seconds of full training time, model file size of 0.21 MB (over 50 times smaller than 11.26 MB of the model file for RF), and a single sample prediction latency of 0.7647 ms per sample (0.0058 ms per sample in 3,000 sample batches). For real-time mobile malware detection, this sub-millisecond performance is perfect for XGBoost.

Table 1. Empirical Computational Complexity and Real-Time Efficiency Benchmark

Model	Hyperparameter Configuration	Training Time	Serialized Disk Size	Single Sample Latency	Batch (3k) Latency	Per-Sample Batch Latency
LR	max_iter=1000, solver='lbfgs', C=1.0	0.114 s	< 0.01 MB	0.0935 ms	1.00 ms	0.0003 ms
RF	n_estimators=100, max_depth=None	1.565 s	11.26 MB	4.7108 ms	61.75 ms	0.0206 ms
XGBoost	n_estimators=100, max_depth=6, lr=0.3	0.534 s	0.21 MB	0.7647 ms	17.47 ms	0.0058 ms
LightGBM	n_estimators=100, num_leaves=31, lr=0.1	1.838 s	0.35 MB	2.4802 ms	12.03 ms	0.0040 ms
MLP	hidden=(100,50), activation='relu'	7.423 s	0.82 MB	0.2466 ms	6.05 ms	0.0020 ms

3.4.3 Confusion Matrix Analysis

In addition to the numerical scores, the Confusion Matrix has been used to provide a more detailed, visualized interpretation of the classification results. It has four parts: True Positives (TP), malicious applications that are rightly considered malicious; True Negatives (TN), benign applications that are rightly considered benign; (FP), benign applications that are mistakenly considered malicious; and (FN), malicious applications that are mistakenly considered benign. This kind of error analysis can be done with a high degree of accuracy, and it can be used for deciding which model is more misclassifiable [16].

Each model's distribution of decisions is further explained in a heatmap, which makes it much easier to identify areas for improvement. A case in point is that the high FP rate may indicate that the malware-like attributes are overly sensitive, and the high FN rate may pose a security risk, as malware may pass unnoticed. High TP and TN values and low FP and FN values indicate balanced detection and reliability, making the classifier with these values the best. We made classification reports to assess performance in each class. These were the classification reports, which provided us with the Precision, Recall and F1 Score for each class, which we could then use to compare the performance of the model on each of the classes. These metrics, along with other evaluation tools enabled us to assess the effectiveness of each algorithm and to select the best algorithm for Android malware detection.

3.5 Methodological Setup of SHAP Interpretability

We use post-hoc cooperative game theory SHAP (SHapley Additive exPlanations) to give mathematical insight into the logic behind model decisions. TreeExplainer (shap.TreeExplainer is used to calculate the feature attributions) for tree ensemble models. All SHAP feature importance rankings, summary distributions, and local waterfall explanations are made using pooled Out-Of-Fold (OOF) cross-validation predictions on the entire 15,036 samples of the dataset (N = 15,036) to remove the evaluation partition bias. The background reference distribution is defined on the out-of-fold validation sets and SHAP values are calculated in log-odds margin space before being transformed to output probabilities. The mean absolute SHAP value $\frac{1}{N} \sum_{j=1}^N |\phi_j^{(i)}|$ is used to calculate global feature ranking.

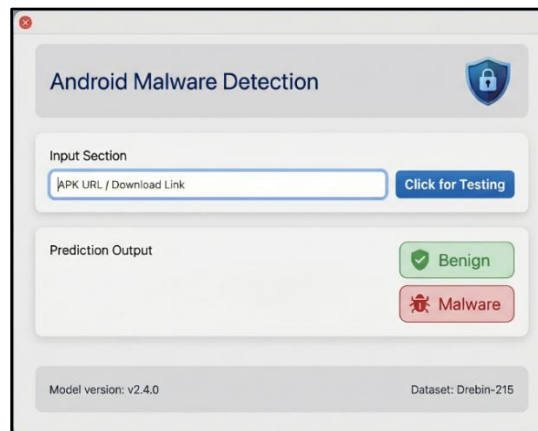


Figure 2. Proposed web-based deployment interface for real-time Android malware detection.

3.6 Conceptual of System Architecture and Database-Driven Workflow

Figure 2 illustrates the conceptual system architecture and system operational deployment process that links offline ML evaluation and automated mobile security triage. The planned deployment process has four stages:

1. The security practitioner uploads or downloads an Android Application Package (APK) file to a web backend, which then calculates the APK's cryptographic hash using the SHA-256 algorithm, and stores the raw APK file in a centralized database schema (Table 2). A relational database is used, which consists of four normalized tables (APKs, Extracted_Features, Inference_Logs, and SHAP_Attributions) for complete tracking of the artifacts, persistence of features, and auditability of predictions.
2. A background worker calls static disassembly utilities (e.g., Androguard) to generate the 215-dimensional Drebin feature representation used in our experiments. The worker reads the AndroidManifest.xml and extracts the requested permissions, intent filters and requested hardware components (113 features), and disassembles the DEX bytecode and extracts the API call invocations (102 features). The resulting binary vector is then checked against the feature schema before they are inferred.
3. The normalized feature vector is forwarded to the trained XGBoost model engine which gives the classification probability score (0.7647 ms). At the same time, a local SHAP explainer pulls out the top-5 most important feature attributions supporting the decision.
4. End-to-End System Performance Breakdown: On our evaluation benchmark workstation, the end-to-end pipeline achieves a total processing latency of ~1.40 seconds per APK (consisting of 1.20s for static disassembly, 0.05s for database persistence, 0.00076s for model prediction, and 0.15s for local SHAP attribution generation). This validates that ML inference is an add-on to the static analysis workflow without significant overhead. Future production engineering will be performed on detailed multi-tenant web server stress testing and load balancing.

Table 2. Relational Database Schema Architecture for Mobile Triage Pipeline

Table Name	Primary Key (PK)	Foreign Key (FK)	Core Attributes / Columns	Data Types & Description
APKs	apk_id	None	sha256_hash, file_name, upload_timestamp, raw_file_path	VARCHAR(64) SHA-256 hash; VARCHAR(255) filename; TIMESTAMP submission time; VARCHAR(512) storage path
Extracted_Features	feature_id	apk_id	permission_flags, api_call_flags, intent_flags, extraction_time_s	BIT(113) permissions & intents; BIT(102) disassembled DEX APIs; FLOAT disassembly duration
Inference_Logs	log_id	apk_id	model_name, prediction_class, probability_score, latency_ms	VARCHAR(50) model name; TINYINT (0 = Benign, 1 = Malware); FLOAT confidence score; FLOAT latency (0.7647 ms)
SHAP_Attributions	attribution_id	apk_id	base_value, top_features_json, shap_values_json	FLOAT SHAP expected base value; JSON top 10 feature names; JSON local SHAP attribution values

4. Results and Discussion

This section provides the experimental results obtained in the application of evaluated machine learning (ML) models in the Android malware detection task. Computational efficiency, predictive performance, classification results from confusion matrices, Receiver Operating Characteristic (ROC) curves and model interpretability via SHAP (SHapley Additive exPlanations) analysis are all taken into account.

Confusion matrices are also analyzed to delve deeper into the classification results, providing a summary of the number of (TP), (TN), (FP), and (FN) for the evaluated models. These measures help to directly observe the classification results and the prediction pattern differences among LR, RF and XGBoost can be analyzed.

The classification results of the LR model is shown in Fig. 3. In the confusion matrix, it is seen that the model correctly classified 1867 (TN) benign and 1075 (TP) malicious applications. Besides, 29 benign applications were falsely detected as malicious (FP) and 37 malicious ones were falsely detected as benign (FN). These results give a general idea about the classification results achieved by the LR model for the two classes.

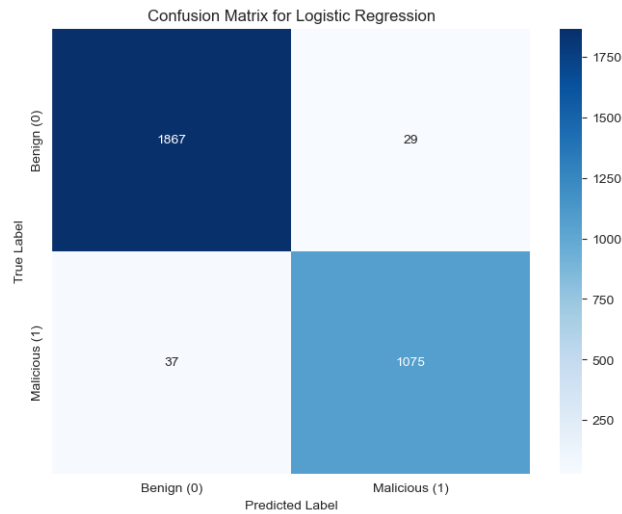


Figure 3. Confusion Matrix for LR.

The classification results of the RF model are shown in Fig. 4. From the confusion matrix, it can be seen that the RF correctly classified 1,886 benign applications (TN) and 1,085 malicious applications (TP). The model generated 10 FP and 27 malicious applications were classified as benign (FN). RF had a lower number of FP and FN classifications than LR.

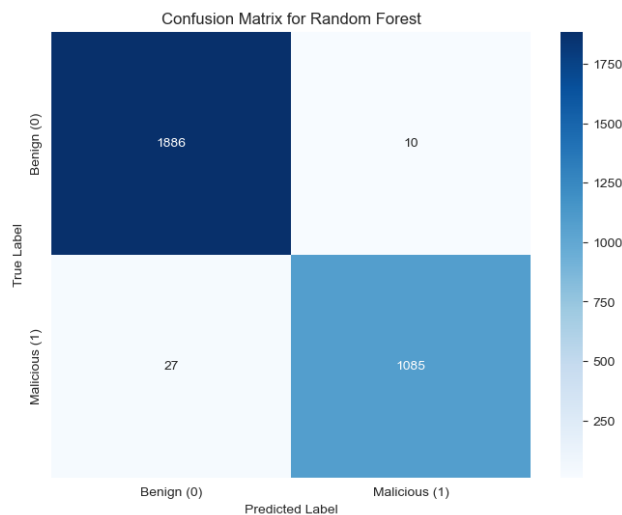


Figure 4. Confusion Matrix for RF.

The XGBoost confusion matrix is shown in Fig. 5. The model was able to correctly classify 1,885 benign applications (TN) and 1,089 malicious applications (TP). Compared with RF, it had one more FP and four fewer FN, which revealed that the distribution of classification errors of the two models differed.

Table 3 shows the results with 5-fold stratified cross validation for the 15,036 samples (9476 benign and 5560 malicious) in the dataset. This evaluation estimates performance on several stratified data partitions instead of a single train-test partition. All reported values are the mean and standard deviation of the mean over the five folds for each evaluation metric. The variation in folds varied by model and evaluation measure.

The Accuracy and F1-score of RF and XGBoost were almost the same. RF obtained an Accuracy of 0.9880 ± 0.0024 and an F1-score of 0.9837 ± 0.0033 , while XGBoost achieved 0.9882 ± 0.0024 and 0.9839 ± 0.0033 , respectively. However, there were more significant differences in Precision and Recall, with RF achieving a higher Precision of 0.9929 ± 0.0021 than XGBoost (0.9880 ± 0.0025) and a lower Recall of 0.9746 ± 0.0068 than XGBoost (0.9799 ± 0.0058). Their ROC-AUC values were very similar, at 0.9984 ± 0.0006 for RF and 0.9985 ± 0.0005 for XGBoost.

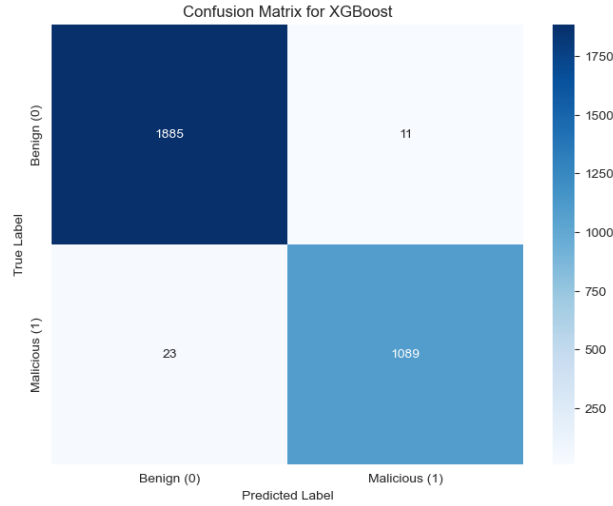


Figure 5. Confusion Matrix for XGBoost.

Among the evaluated models, MLP achieved the highest mean Recall (0.9827 ± 0.0056) and F1-score (0.9847 ± 0.0026), followed by LightGBM with an F1-score of 0.9843 ± 0.0044 and XGBoost with 0.9839 ± 0.0033 . MLP also obtained the highest Accuracy at 0.9887 ± 0.0019 , while LightGBM recorded 0.9884 ± 0.0032 . The model differences were rather small and the overall performance measures were in a rather narrow range.

There were also differences in the computational characteristics of the models. The time needed for the 5-fold evaluation was 3.34 s for XGBoost, 7.87 s for RF and 38.29 s for MLP. Moreover, Table 1 shows that XGBoost's single-sample inference latency is 0.7647 ms and the serialized disk size of XGBoost is 0.21 MB. Among the models evaluated, XGBoost demonstrated good prediction accuracy and moderate complexity.

Table 3. Performance comparison of ML models using 5-Fold Stratified Cross-Validation

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC	Execution Time
LR	0.9763 ± 0.0033	0.9745 ± 0.0027	0.9610 ± 0.0080	0.9677 ± 0.0045	0.9961 ± 0.0005	1.80 s
RF	0.9880 ± 0.0024	0.9929 ± 0.0021	0.9746 ± 0.0068	0.9837 ± 0.0033	0.9984 ± 0.0006	7.87 s
XGBoost	0.9882 ± 0.0024	0.9880 ± 0.0025	0.9799 ± 0.0058	0.9839 ± 0.0033	0.9985 ± 0.0005	3.34 s
LightGBM	0.9884 ± 0.0032	0.9891 ± 0.0027	0.9795 ± 0.0072	0.9843 ± 0.0044	0.9985 ± 0.0005	4.78 s
MLP	0.9887 ± 0.0019	0.9867 ± 0.0026	0.9827 ± 0.0056	0.9847 ± 0.0026	0.9980 ± 0.0006	38.29 s

The ROC curves of LR, RF and XGBoost are shown in Fig. 6. The curves are used to show the relationship between the true positive rate and false positive rate over various classification thresholds and to offer a threshold-independent look at how effectively the models can separate malicious from benign applications. The diagonal line is the reference line indicating the performance of a classifier having an AUC of 0.50.

The ROC curves of all three models are well above the diagonal reference line and are close to the upper left corner of the plot as shown in Fig. 6. The curves are also tightly clustered, with RF and XGBoost showing the most similar performance of discrimination across the tested thresholds. This is a visual indication of the high and closely grouped ROC-AUC values reported in Table 3.

SHAP analysis was also performed to investigate the predictions generated by XGBoost beyond the overall performance metrics, to assess the contribution of each feature to the predictions. The next analysis takes into account the feature contributions both globally and at the level of individual prediction.

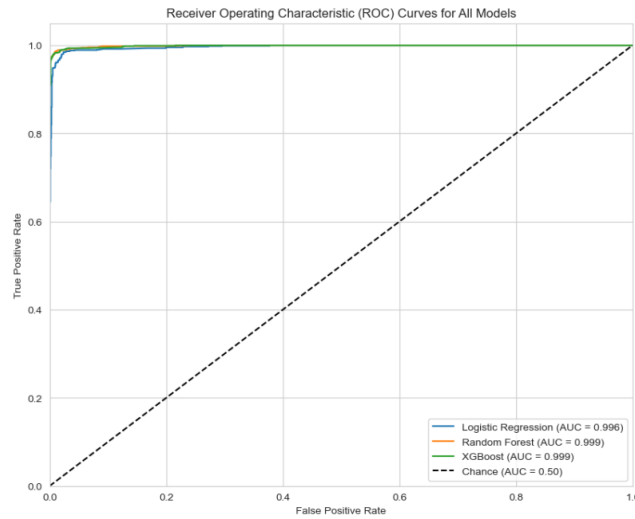


Figure 6. ROC curves of LR, RF, and XGBoost models for malicious application detection.



Figure 7. Global Feature Importance.

Fig. 7 also shows the global feature importance results, where the 20 features with the highest importance are ranked by their mean absolute values of SHAP values across the pooled out-of-fold samples ($N = 15,036$). The SHAP values were calculated in the log-odds margin space of the model. It is an average of the size of the individual features contribution to the model output for the samples evaluated, regardless of whether the contribution is positive or negative.

As can be seen in Fig. 7, the mean absolute SHAP values of the ranked features are mostly dominated by SEND_SMS followed by TelephonyManager.getDeviceId and Ljava.net.URLDecoder. Ljava.lang., Lio.netty., and Lorg. are also among the top features. Class.getName, onServiceConnected, attachInterface, ClassLoader, TelephonyManager.getLine1Number, and READ_PHONE_STATE. The results show that feature contributions vary with the features evaluated, with the greatest average contribution of the evaluated features being the SEND_SMS feature.

The SHAP summary plot (Fig. 8) shows the distribution and direction of feature contribution across the samples evaluated, while the overall magnitude of the feature contribution is summarized in Fig. 7.

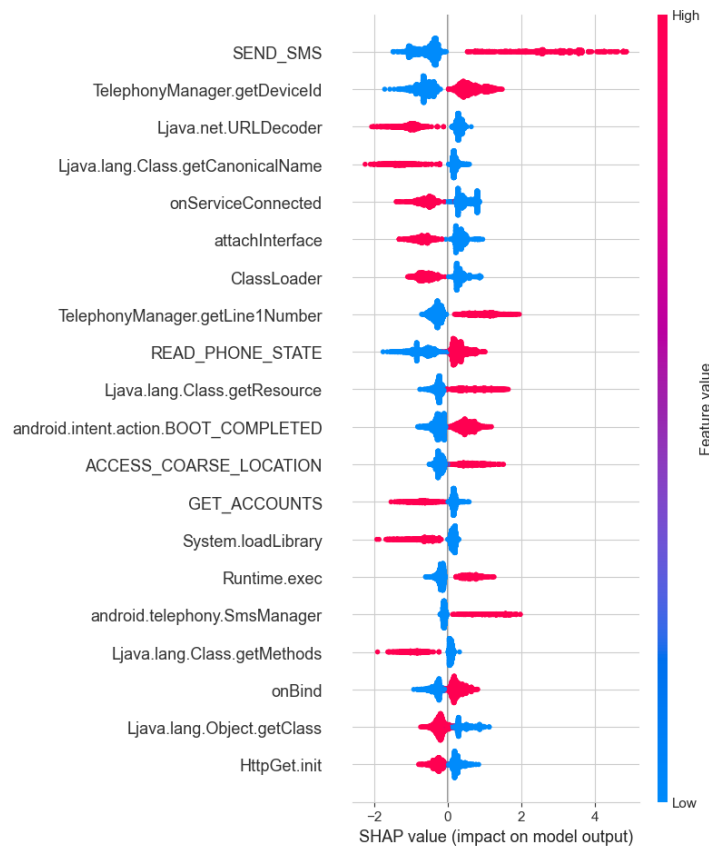


Figure 8. SHAP Summary Plot Illustrating Feature Impact and Value Correlation.

A more detailed picture of the distribution and direction of the feature contributions to the XGBoost output can be gained from the SHAP summary plot in Fig. 8. In the case of SEND_SMS, higher feature values tend to have higher SHAP values and lower values are mostly found on the negative side. The trend is that the higher the value of SEND_SMS, the more towards the malicious classification, and the lower the value of SEND_SMS, the more towards the opposite classification.

The same thing applies for TelephonyManager.getDeviceId, READ_PHONE_STATE, and android.intent.action.BOOT_COMPLETED, in which the higher feature values tend to be linked with positive SHAP values. The amount and distribution of these contributions, however, varies from feature to feature, and from sample to sample. Fig. 8 shows overall the direction and variation of the individual feature contributions to the model output, complementing the global importance ranking in Fig. 7.

In order to further investigate the FN predictions, local SHAP analysis was conducted on the pooled out-of-fold FN subset (112 of the 5,560 malicious samples evaluated over the five folds, or 2.01%). In this subset, the contribution value of SHAP for SEND_SMS was -0.0491 and for READ_PHONE_STATE was -0.0382 , while the value for SEND_SMS was 0 (absent) for all FN samples. These negative contributions suggest a tendency for the respective feature patterns to pull the XGBoost output in the direction of the benign class in the assessed FN subset.

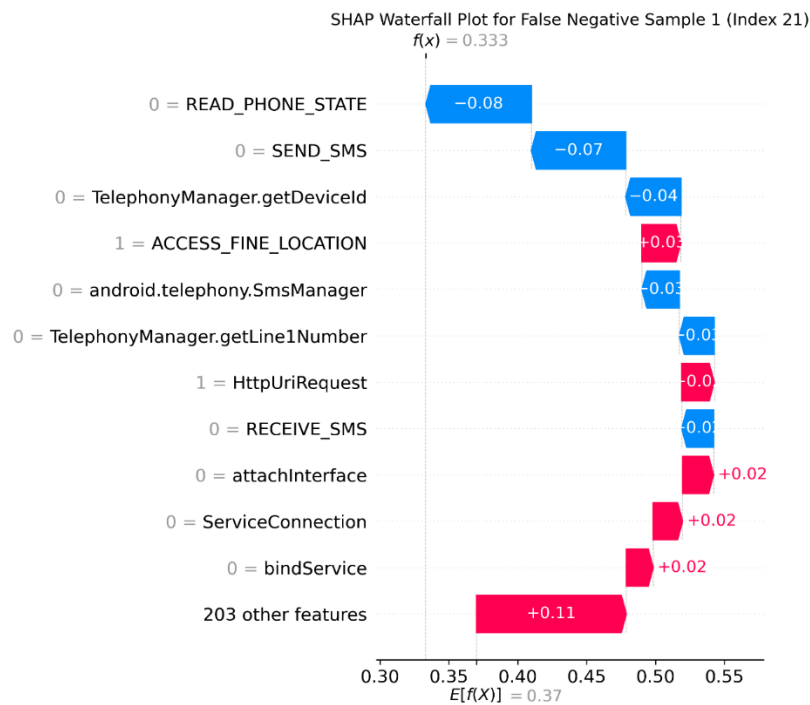


Figure 9. Local SHAP Waterfall Plot for a representative False Negative malware sample misclassified as benign

Fig. 9 shows a local SHAP waterfall analysis for a representative FN sample, showing which features contributed to the sample being classified as FN. In this sample, the following permission(s) are not present: READ_PHONE_STATE, SEND_SMS, TelephonyManager.getDeviceId yielded negative SHAP values of about -0.08 , -0.07 and -0.04 respectively, which pushed the model's predictions towards benign. Other features were added in both directions and their effects added up to a model output of about 0.333 for this sample. The local attribution is thus an example of the ability to relate the combined contribution of several features to a particular FN prediction.

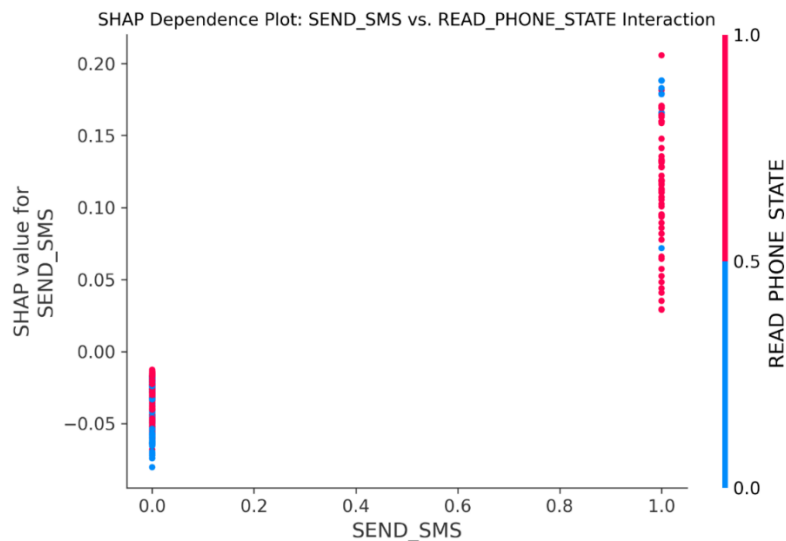


Figure 10. SHAP dependence plot demonstrating feature interaction between SEND_SMS permission and READ_PHONE_STATE permission.

SHAP dependence analysis was also performed to analyze the relationship between top-ranked features. As seen in Fig. 10, SEND_SMS is plotted on the x-axis, the SHAP value on the y-axis, and READ_PHONE_STATE is plotted by the color scale. The SHAP values for SEND_SMS are mostly negative when the value is 0 (which is when the variable is absent), while the values are mostly positive when the variable is present (value = 1). For the samples where SEND_SMS is present, the difference in SHAP

values across various levels of READ_PHONE_STATE suggests that there is interaction between the two features in the model. Specifically, the larger values of READ_PHONE_STATE tend to have larger positive SHAP contributions for SEND_SMS. This pattern shows that the value of SEND_SMS will affect the output of the model, depending on the value of READ_PHONE_STATE. SHAP results should be viewed as feature attributions that indicate how a feature value influences the output of the fitted model, not as a direct causal relationship between the feature value and malicious behaviour. Thus, the observed attribution patterns describe the behavior of that model when deciding on the malware they evaluated and do not prove to be resistant to unforeseen malware behavior or adversarial manipulation.

In sum, these SHAP analyses offer additional global and local insights into the XGBoost predictions. The global analyses in Figs. 7 and 8 determine the relative importance, direction and distribution of the contribution of features, and the local analysis in Fig. 9 shows how the contribution of features is combined in an individual FN prediction. The dependence analysis in Fig. 10 also suggests that the effect of a highly-ranked feature may differ depending on the values of another feature. These analyses, combined, offer an extra layer of interpretive context for the model's predictions, beyond simple measures of predictive performance.

4.1 Threats to Validity and Limitations

The reported performance is based on 5-fold stratified cross validation and may not be directly representative of performance in the presence of temporal shifts in feature distribution. As pointed out by Pendlebury et al. in the TESSERACT framework [29] the characteristics of Android malware are changing, which can impact static malware detection using historical data sets. Further, the explicit consideration of temporal drift in the evolution of malware is not taken into account when using pre-extracted tabular features. In future studies, the authors will thus take into account the modern dynamic-analysis datasets and temporal split evaluations to further examine the generalizability of models over time.

5. Conclusion and Future Work

In this study, the performance of ML models for Android malware detection was assessed based on static application features, focusing on predictive accuracy, computational efficiency, and interpretability. In all models analyzed, XGBoost showed promising prediction accuracy with moderate computational needs. Global and local feature contribution information was added to the predictive evaluation using the SHAP analysis. Functionalities like SEND_SMS, TelephonyManager. The features getDeviceId and READ_PHONE_STATE demonstrated significant impact on the model output and the local analysis of FN predictions also indicated that different feature patterns may affect misclassification.

These findings prove the usefulness of not only predictive evaluation but also interpretability at the level of the model in the context of looking at ML-based Android malware detection. The SHAP analysis did not just report the overall and individual model predictions, but also gave information about the features associated with the overall and individual predictions. However, all results are only for this particular data set, the fixed features representation, and the evaluation context of this paper and cannot be used as proof of causality between single features and malicious behavior.

Future research will include expanding the evaluation to dynamic behavioral data and exploring deep learning architectures such as CNNs and LSTMs to evaluate their performance under complementary feature representations. Temporal evaluation strategies will also be examined to assess the performance of a model in the face of temporal changes of malware characteristics. Further, regular updates to training datasets and adversarial robustness testing against feature manipulation are important avenues for measuring the reliability of malware detection models in a dynamically changing threat landscape.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

- [1] Alani, M. M., & Awad, A. I. (2022). PAIRED: An explainable lightweight Android malware detection system. *IEEE Access*, 10, 73214–73228. <https://doi.org/10.1109/ACCESS.2022.3189645>
- [2] Alani, M. M., Mashatan, A., & Miri, A. (2023). Adversarial explainability: Utilizing explainable machine learning in bypassing IoT botnet detection systems. *arXiv*. <https://doi.org/10.48550/arXiv.2310.00070>
- [3] Alhogail, A., & Alharbi, R. A. (2025). Effective ML-based Android malware detection and categorization. *Electronics*, 14(8), 1486. <https://doi.org/10.3390/electronics14081486>
- [4] Almadhor, A., Alsubai, S., Kryvinska, N., Al Hejaili, A., Bouallegue, B., Ayari, M., & Abbas, S. (2025). Transfer learning with XAI for robust malware and IoT network security. *Scientific Reports*, 15, 26971. <https://doi.org/10.1038/s41598-025-12404-w>
- [5] Alperin, K. B., Wollaber, A. B., & Gomez, S. R. (2020). Improving interpretability for cyber vulnerability assessment using focus and context visualizations. In *Proceedings of the 2020 IEEE Symposium on Visualization for Cyber Security (VizSec)* (pp. 30–39). IEEE. <https://doi.org/10.1109/VizSec51108.2020.00011>

- [6] Alshomrani, M., Albeshri, A., Alsulami, A. A., & Alturki, B. (2025). An explainable hybrid CNN–Transformer architecture for visual malware classification. *Sensors*, 25(15), 4581. <https://doi.org/10.3390/s25154581>
- [7] Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., & Rieck, K. (2014). DREBIN: Effective and explainable detection of Android malware in your pocket. In *Proceedings of the 2014 Network and Distributed System Security Symposium*. Internet Society. <https://doi.org/10.14722/ndss.2014.23247>
- [8] Bacevicius, M., & Paulauskaite-Taraseviciene, A. (2023). Machine learning algorithms for raw and unbalanced intrusion detection data in a multi-class classification problem. *Applied Sciences*, 13(12), 7328. <https://doi.org/10.3390/app13127328>
- [9] Bae, H. (2024). Evaluation of malware classification models for heterogeneous data. *Sensors*, 24(1), 288. <https://doi.org/10.3390/s24010288>
- [10] Baghirov, E. (2024). Advanced machine learning and interpretability for Windows malware detection. *Journal of Modern Technology and Engineering*, 9(3), 165–177. <https://doi.org/10.62476/jmte93165>
- [11] Baker, M. R., Etem, T., Jihad, K. H., & Buyrukoglu, S. (2025). The role of hyperparameter tuning in phishing website classification: A comparative analysis of ML models. In *Hybrid Intelligent Systems* (pp. 68–77). Springer. https://doi.org/10.1007/978-3-031-78928-1_8
- [12] Cannarile, A., Dentamaro, V., Galantucci, S., Iannacone, A., Impedovo, D., & Pirlo, G. (2022). Comparing deep learning and shallow learning techniques for API calls malware prediction: A study. *Applied Sciences*, 12(3), 1645. <https://doi.org/10.3390/app12031645>
- [13] Cao, S., Sun, X., Widyasari, R., Lo, D., Wu, X., Bo, L., Zhang, J., Li, B., Liu, W., Wu, D., & Chen, Y. (2026). A systematic literature review on explainability for ML/DL-based software engineering. *ACM Computing Surveys*, 58(4), Article 95. <https://doi.org/10.1145/3763230>
- [14] Capuano, N., Fenza, G., Loia, V., & Stanzione, C. (2022). Explainable artificial intelligence in CyberSecurity: A survey. *IEEE Access*, 10, 93575–93600. <https://doi.org/10.1109/ACCESS.2022.3204171>
- [15] Charmet, F., Tanuwidjaja, H. C., Ayoubi, S., Gimenez, P.-F., Han, Y., Jmila, H., Blanc, G., Takahashi, T., & Zhang, Z. (2022). Explainable artificial intelligence for cybersecurity: A literature survey. *Annals of Telecommunications*, 77(11–12), 789–812. <https://doi.org/10.1007/s12243-022-00926-7>
- [16] Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21, 6. <https://doi.org/10.1186/s12864-019-6413-7>
- [17] Chowdhury, N. H. I., Islam, M. E., Hasan, M. S., & Mehedi, A. A. (2025). AI-powered behavior-based malware detection using advanced temporal and process-state features: A robust explainable framework. *Cuestiones de Fisioterapia*, 54(3), 3883–3902. <https://doi.org/10.48047/z6pyk694>
- [18] Jagan, S., Ashish, A., Mahdal, M., Isabels, K. R., Dhanke, J., Jain, P., & Elangovan, M. (2023). A meta-classification model for optimized ZBot malware prediction using learning algorithms. *Mathematics*, 11(13), 2840. <https://doi.org/10.3390/math11132840>
- [19] Karn, R. R., Kudva, P., Huang, H., Suneja, S., & Elfadel, I. M. (2021). Cryptomining detection in container clouds using system calls and explainable machine learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(3), 674–691. <https://doi.org/10.1109/TPDS.2020.3029088>
- [20] Khan, N., Ahmad, K., Tamimi, A. A., Alani, M. M., Bermak, A., & Khalil, I. (2024). Explainable AI-based intrusion detection system for Industry 5.0: An overview of the literature, associated challenges, the existing solutions, and potential research directions. *arXiv*. <https://doi.org/10.48550/arXiv.2408.03335>
- [21] Kumar, R., & Subbiah, G. (2022). Zero-day malware detection and effective malware analysis using Shapley ensemble boosting and bagging approach. *Sensors*, 22(7), 2798. <https://doi.org/10.3390/s22072798>
- [22] Kuppa, A., & Le-Khac, N.-A. (2020). Black box attacks on explainable artificial intelligence (XAI) methods in cyber security. In *Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). IEEE. <https://doi.org/10.1109/IJCNN48605.2020.9206780>
- [23] Lin, Y., & Chang, X. (2021). Towards interpreting ML-based automated malware detection models: A survey. *arXiv*. <https://arxiv.org/abs/2101.06232>
- [24] Moamin, S. A. H., Abdulhameed, M. K., Al-Amri, R. M., Radhi, A. D., Naser, R. K., & Pheng, L. G. (2025). Artificial intelligence in malware and network intrusion detection: A comprehensive survey of techniques, datasets, challenges, and future directions. *Babylonian Journal of Artificial Intelligence*, 2025, 77–98. <https://doi.org/10.58496/BJAI/2025/008>
- [25] Mowri, R. A., Siddula, M., & Roy, K. (2022). Application of explainable machine learning in detecting and classifying ransomware families based on API call analysis. *arXiv*. <https://doi.org/10.48550/arXiv.2210.11235>
- [26] Mowri, R. A., Siddula, M., & Roy, K. (2023). Is iterative feature selection technique efficient enough? A comparative performance analysis of RFECV feature selection technique in ransomware classification using SHAP. *Discover Internet of Things*, 3, 21. <https://doi.org/10.1007/s43926-023-00053-2>
- [27] Muhammad, A. E., Yow, K.-C., Bačaniin-Džakula, N., & Khan, M. A. (2025). L-xaids: A LIME-based eXplainable AI framework for intrusion detection systems. *Cluster Computing*, 28(10), 654. <https://doi.org/10.1007/s10586-025-05326-9>
- [28] Nguyen, M. D., La, V. H., Mallouli, W., Cavalli, A. R., & D. Oca, E. M. (2024). Toward anomaly detection using explainable AI. In *CyberSecurity in a DevOps Environment: From Requirements to Monitoring* (pp. 293–324). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-42212-6_10
- [29] Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., & Cavallaro, L. (2019). TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security '19)* (pp. 729–746). USENIX Association.
- [30] Uysal, E. I., Demircioğlu, G., Kale, G., Bostanci, E., Güzel, M. S., & Mohammed, S. N. (2019). Network anomaly detection system using genetic algorithm, feature selection and classification. In *Proceedings of the 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)* (pp. 1–5). IEEE. <https://doi.org/10.1109/ISMSIT.2019.8932807>
- [31] Zhang, Z., Hamadi, H. A., Damiani, E., Yeun, C. Y., & Taher, F. (2022). Explainable artificial intelligence applications in cyber security: State-of-the-art in research. *IEEE Access*, 10, 93104–93139. <https://doi.org/10.1109/ACCESS.2022.3204051>