Cyber Security and Applications 3 (2025) 100072

Contents lists available at ScienceDirect

Cyber Security and Applications

journal homepage: http://www.keaipublishing.com/en/journals/cyber-security-and-applications/

A comprehensive investigation into robust malware detection with explainable AI

E. Baghirov*

Institute of Information Technology, 9A, B. Vahabzade Street, Baku, AZ1141, Azerbaijan

ARTICLE INFO

Keywords:
Malware detection
Malware analysis
Explainable AI
SHAP
LIME
LightGBM

ABSTRACT

In today's digital world, malware poses a serious threat to security and privacy by stealing sensitive data and disrupting computer systems. Traditional signature-based detection methods have become inefficient and time-consuming. However, data-driven AI techniques, particularly machine learning (ML) and deep learning (DL), have shown effectiveness in detecting malware by analyzing behavioral characteristics. Despite their promising performance, the black-box nature of these models requires improved explainability to facilitate their adoption in real-world applications. This can complicate the ability of cybersecurity experts to evaluate the model's reliability. In this work, Explainable Artificial Intelligence (XAI) is employed to comprehend and evaluate the decisions made by machine learning models in the detection of malware on Android devices. To evaluate malware detection, experiments were conducted using CICMalDroid dataset by applying ML models like Logistic Regression and several tree algorithms. An overall 94% F1-score was achieved, and interpretable explanations for model decisions were provided, highlighting more critical features that contributed to accurate classifications. It was found that employing XAI techniques can provide valuable insights for malware analysis researchers, enhancing their understanding of the operations of the ML model, rather than solely focusing on improving accuracy.

1. Introduction

The impact of malware on individuals, organizations, and society at large is profound and multifaceted. Cybercriminal attacks with various objectives, such as inserting advertisements and purchase offers into websites, denying service to computing systems, or stealing confidential data for financial gain, often involve the use of malicious software. This software, intended to disrupt, compromise, or gain unauthorized access to computer systems, can result in significant financial, operational, and reputational harm [1,2]. For individuals, malware poses risks ranging from identity theft to the loss of personal data and financial assets. Organizations face the daunting prospect of disrupted business operations, data breaches, and potential legal consequences. The societal impact extends beyond immediate victims, as large-scale malware attacks can cripple critical infrastructures, compromise national security, and disrupt the functioning of essential services. As malware continually evolves in sophistication and scale, the need for robust detection and prevention measures becomes imperative to safeguard the interconnected digital landscape from the far-reaching consequences of malicious activities [3].

According to the 2024 report of Malwarebytes, the number of documented ransomware attacks saw a significant increase 68% in 2023. Furthermore, in 2023, malware made up 11% of Mac detections [4]. The AV-TEST Institute records more than 450,000 new instances of malicious programs (malware) and potentially unwanted applications (PUA) every single day [5].

Malware, a prevalent cyber threat via the Internet, has raised increasing concern, prompting the deployment of various protective measures. In response to the increasing threat posed by malware, cybersecurity experts have devised various strategies to detect and mitigate its impact. Among these strategies, static and dynamic analysis methods play pivotal roles in identifying and analyzing malicious software. Static analysis involves examining the code or executable file of a program without executing it. This method typically entails inspecting characteristics such as file signatures, metadata, API calls, and patterns of code to identify potential indicators of malicious behavior [6]. Static analysis can be performed quickly, making it suitable for scanning large volumes of files in a relatively short period of time [7]. Since static analysis does not require the execution of the malware, it poses minimal risk of inadvertently triggering malicious actions [8]. Automated tools can be employed to perform static analysis efficiently, enabling rapid processing of large datasets. However, static analysis may struggle to detect polymorphic or obfuscated malware that actively seeks to evade detection by altering its code or structure [9]. Due to the reliance on heuristics and

Peer review under responsibility of KeAi Communications Co., Ltd.

* Corresponding author.
E-mail address: elsenbagirov1995@gmail.com

https://doi.org/10.1016/j.csa.2024.100072
Received 18 May 2024; Received in revised form 9 August 2024; Accepted 4 September 2024
Available online 5 September 2024

static signatures, static analysis methods can produce false positives, incorrectly flagging benign files as malicious [2,10]. Since static analysis does not involve executing the malware, it may overlook behaviors that only manifest at runtime.

Dynamic analysis involves running malware in a controlled environment, such as a virtual machine or sandbox, to observe its behavior in real time [8,11]. This method enables analysts to monitor activities such as changes to the file system, network communications, and system calls. Dynamic analysis provides valuable information on the actual behavior of malware, allowing analysts to observe its interactions with the host system and the network. Unlike static analysis, dynamic analysis can potentially detect polymorphic malware and previously unseen threats by observing their behavior. Dynamic analysis can detect evasion techniques employed by malware to avoid static detection methods. However, running malware in a dynamic analysis environment requires significant computational resources and can impact the performance of the host system. Dynamic analysis may introduce a delay in detection as analysts observe the malware's behavior, potentially allowing it to cause damage before being identified. Advanced malware may employ evasion techniques to detect the presence of a virtual environment or sandbox, limiting the effectiveness of dynamic analysis [12].

Despite the strengths of static and dynamic analysis methods, malware authors continue to evolve their tactics to avoid detection, presenting ongoing challenges to cybersecurity professionals. Future research in the field of malware detection is likely to focus on developing hybrid approaches that combine the strengths of static and dynamic analysis while addressing their respective limitations. Furthermore, advances in explainable AI promise to improve the interpretability of detection models, allowing analysts to better understand and trust the decisions made by automated malware detection systems [2,6]. By comprehensively understanding the landscape of current malware detection methods, including their advantages, drawbacks, and challenges, researchers and practitioners can inform the development of more robust and effective approaches to combatting the ever-evolving threat posed by malicious software.

Although improving accuracy has been the central objective of most research in machine learning-based Android malware detection, there is a significant lack of insight into the underlying reasons for the models' performance. Recent studies have highlighted that the impressive accuracy of these models may stem from various experimental biases rather than genuine detection of malware characteristics. Despite this, much of the existing research continues to focus solely on prediction accuracy without considering whether the models are identifying malware based on relevant features. Transparent models, such as tree-based models, do not require complex steps for interpretation, as their structure can be easily visualized by displaying the decision tree. In contrast, boosting algorithms and other advanced machine learning models need additional methods to interpret their decisions and understand their inner workings. These models are often more complex and less intuitive, necessitating the use of explainable AI techniques to provide insights into their decision-making processes.

The following are the main contributions of this work.

- *Comprehensive Investigation*: The work conducts an in-depth exploration of robust malware detection, focusing on the integration of XAI techniques.
- *Insights from XAI and Understanding Critical Variables:* Our research highlights the invaluable insights offered by Explainable AI, shedding light on the key variables that ML models use to differentiate between benign and malicious apps. This contributes to a deeper understanding of the functioning of malware detection models and enhances both their knowledge and interpretability.
- *High Detection Rate:* The proposed LightGBM model achieved the best results, demonstrating a high detection rate in malware identification.

The rest of this systematic study is structured as follows. Section II discusses related works on malware detection. Section III provides the background on basic malware analysis and detection concepts, explainable AI techniques. Section IV outlines the experiments and results.

2. Related works

Few works have investigated XAI within malware detection methods. Some studies have been analyzed, and their approaches and limitations are briefly discussed in this section.

[2] presents a novel framework integrating XAI methodologies into AI-based malware detection systems to enhance result interpretability. It evaluates four XAI methods - SHAP, LIME, LRP, and Attention Mechanism - across datasets with varying sizes and model architectures like LSTM and GRU. The research aims to identify the strengths and weaknesses of these approaches and assess their applicability. However, there is no clear result showing the impact of features on detection across different explainable methods, leaving an area of uncertainty regarding the consistency and reliability of feature importance as interpreted by various XAI techniques. Additionally, although there are numerous classes in the dataset, only the backdoor, trojan, and downloader classes are interpreted, leaving other malware classes unexplored in terms of feature importance and interpretability.

[13] employs Explainable AI approaches to scrutinize the efficacy of ML based malware classifiers under idealized experimental conditions, revealing over-optimistic performance due to temporal sample inconsistencies in the training dataset. The findings shed light on the ML models' reliance on temporal differences rather than actual malicious behaviors for classification, highlighting the need for realistic experimental designs in malware detection research. The authors investigated both SHAP and LIME; however, the results of SHAP and LIME figures have not been shown, leaving a gap in the visual demonstration of feature impacts across these explainable methods.

[14] employs XAI to delve into the decision-making processes of ML models used in Android malware detection. Through experiments conducted on seven datasets, it demonstrates the potential for accurate malware identification, albeit with variations in available features across datasets. The study explores the implications of integrating expert-dependent features, highlighting both their potential to enhance accuracy and the challenges they pose in terms of resource and time requirements, human bias, and scalability issues in the Android application landscape. One potential drawback highlighted in the paper is the resource-intensive nature of incorporating expert-dependent features into the malware detection process. This approach requires a detailed manual analysis, which can increase both the time and resource requirements of the detection procedure. Additionally, the investigated datasets are binary, and it would be beneficial to explore multiclass datasets.

[15] presents a method to explain CNN predictions in Android malware detection by identifying key opcode sequence locations, comparing results with LIME, and demonstrating alignment across eight malware families. This validation bolsters confidence in CNN efficacy and offers benefits for identifying critical opcode sequences in malware analysis. However, the lack of detailed result explanations and the investigation using only binary datasets are limitations, indicating a need for further research.

[16] proposed an explainable malware detection system leveraging transfer learning and malware visual features. They utilized a pre-trained Bidirectional Encoder Representations from Transformers (BERT) model to extract textual features and a malware-to-image conversion algorithm to transform network byte streams into visual representations. Key features were identified using the FAST extractor and BRIEF descriptor. These features were balanced with the Synthetic Minority Over-Sampling Technique (SMOTE) and processed through a Convolutional Neural Network (CNN) to mine deep features. Finally, the

balanced features were fed into an ensemble model for efficient malware classification, with extensive analysis conducted on CICMalDroid 2020 and CIC-InvesAndMal2019 datasets, and an interpretable AI experiment to validate the methodology. However, the results of LIME are not presented anywhere in the paper. Additionally, in the SHAP results, features are labeled as F1, F2, etc., with the actual feature names missing. Also, insights derived from a single sample might not be representative of the model's performance across different types of data. There is a need for validation using a broader range of samples to ensure the explanation is applicable in various scenarios.

EvadeDroid, a problem-space adversarial attack designed to evade black-box Android malware detectors, achieves high evasion rates (80%-95%) against various detectors with minimal queries, as presented in [17]. The technique leverages opcode-level transformations from benign apps, preserving stealthiness with a 79% evasion rate against commercial antiviruses, thus demonstrating its practicality in real-world scenarios. EvadeDroid's primary limitation lies in the increased size of adversarial examples (AEs) for small malware apps, which could trigger suspicion in malware detectors. Additionally, while it is effective against static feature-based classifiers, its ability to bypass behavior-based detectors and applicability to other platforms remain open areas for further research.

In [18], the authors explore metric learning for embedding Windows PE files into a low-dimensional vector space, enhancing malware detection and classification. They derive various metric embeddings using a neural network trained with contrastive loss and evaluate performance on the EMBER and SOREL datasets, showing that these embeddings maintain effectiveness with reduced storage needs. The study also addresses practical considerations, including robustness to adversarial attacks and the introduction of task-specific objectives to boost performance. The paper's proposed metric embeddings, while generally effective, suffer from limitations such as reduced robustness to adversarial evasion when task-specific objectives are added. Additionally, the embeddings did not consistently outperform raw feature classifiers across all tasks, suggesting challenges related to overfitting, hyperparameter tuning, and the integration of semantic information.

In [19], the authors suggest a deep learning model for Android malware detection and classification, employing Stacked Autoencoders (SAE) for dimensionality reduction and Convolutional Neural Networks (CNN) for binary visualization of malware. While the model achieves a high accuracy of approximately 98.50% on the Drebin-215 dataset and demonstrates strong performance through precision, recall, and F1-score metrics, it has notable limitations. The reliance on dimensionality reduction techniques may omit crucial features, affecting the models robustness against diverse malware types. Additionally, the evaluation is restricted to a single dataset, which might not fully capture the variety of real-world malware. The model also faces challenges in explainability, as deep learning methods often lack transparency in feature impact and decision-making processes, complicating the interpretation of classification results.

Common problems in the analyzed works include the reliance on over-sampling or under-sampling techniques, which artificially boost accuracy but may not translate to real-world effectiveness. Additionally, the explanation of XAI results is often weak, with insufficient detail on feature impact across different methods, and in some cases, results from SHAP and LIME are not presented. Furthermore, most studies focus on binary datasets, with few investigating multiclass datasets, which limits the comprehensiveness of the malware detection models. Lastly, inconsistencies in feature interpretation and a lack of realistic experimental designs undermine the reliability and applicability of these models in real-world scenarios.

3. Theoretical background and basic concepts

This section introduces some fundamental concepts that underpin this work. It starts by describing the structure of Windows PE files that form the core of the data set. Next, it outlines the machine learning methods employed in the analysis. Finally, it focuses on the explanation techniques utilized throughout the paper to interpret the model's decisions.

3.1. Windows portable executable file structure

In the realm of cybersecurity and malware detection, understanding the structure of Portable Executable (PE) files is paramount. PE files serve as the standard executable format for the Windows operating system, encompassing a broad range of applications, libraries, and dynamically linked modules.

The Portable Executable file structure, as illustrated in Fig. 1, comprises headers and sections, each playing a crucial role in the execution and functionality of executable files within the Windows operating system environment [3,20]. At the forefront lies the headers, beginning with the DOS header, a legacy artifact containing metadata such as the DOS executable file signature and the offset to the PE header. The following is the PE header, denoting the start of the PE-specific structure. This header, adorned with the ”PE/0/” signature, houses essential metadata including the optional header and the section table. The optional header provides additional information about the file, such as its architecture type and the location of the entry point, while the section table enumerates the characteristics and locations of each section within the file.

In parallel, sections within the PE file encapsulate distinct components vital to program execution and functionality. The code section, also known as the text section, houses the executable instructions comprising the program's logic and functionality. Complementing this is the

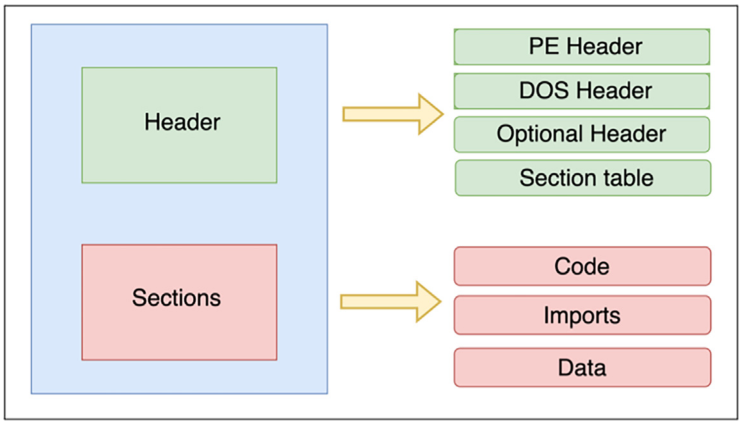


Fig. 1. PE file structure.

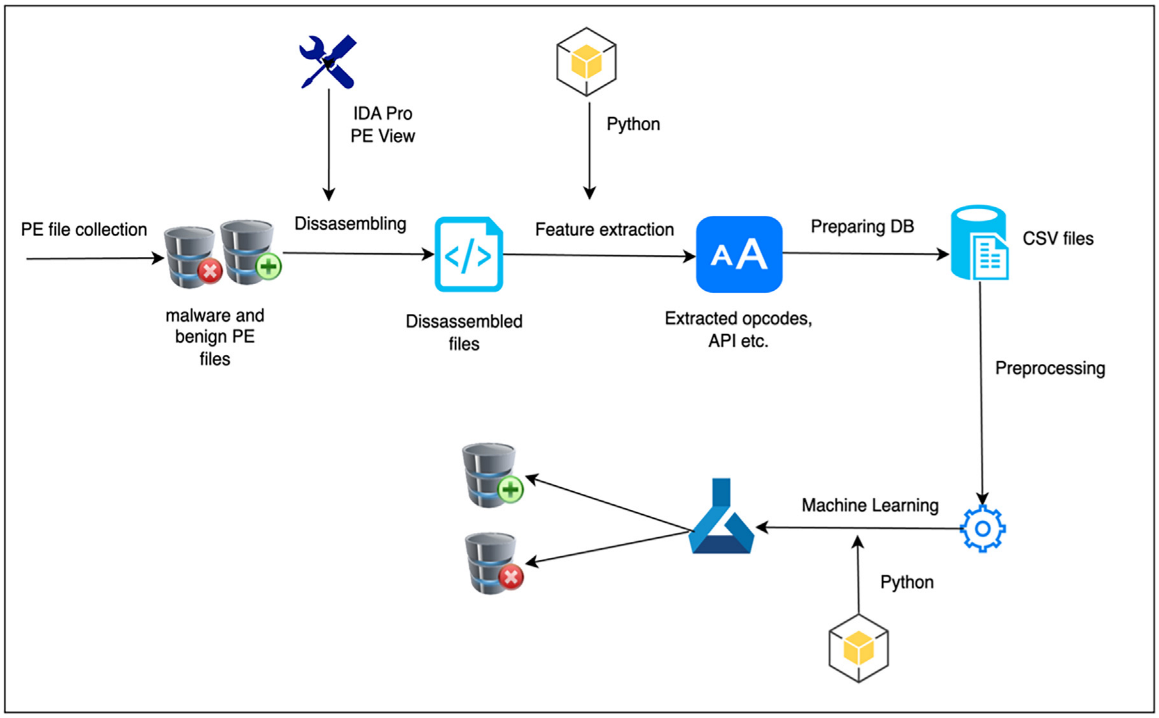


Fig. 2. Common flow of malware detetion with machine learning.

data section that stores initialized data utilized during program execution. Meanwhile, the resource section accommodates non-executable resources like icons, images, and strings, accessible by the program at runtime. Lastly, the import section delineates external functions and libraries upon which the program relies, facilitating dynamic linking and interoperability. By scrutinizing and understanding the nuances of these headers and sections, analysts glean insights into the inner workings of executable files, empowering them to detect and mitigate potential security threats effectively [3,20].

As technology evolves and cyber threats become increasingly sophisticated, a robust comprehension of the PE file format remains essential in safeguarding digital ecosystems against malicious activities and ensuring the integrity and security of software applications.

3.2. Machine learning for malware detection

To address the challenges posed by this dynamic threat environment, researchers and cybersecurity practitioners have turned to machine learning as a promising approach to enhance malware detection capabilities. Using the power of artificial intelligence and data-driven algorithms, machine learning techniques offer the potential to detect and mitigate both known and previously unseen malware with greater accuracy and efficiency [8,11]. In this context, the application of machine learning to malware detection has garnered significant attention and investment, with researchers exploring various methodologies, algorithms, and data sources to develop robust and adaptive detection systems. By analyzing characteristics and patterns inherent in malware samples, such as code behavior, file structure, and execution traces, machine learning models can learn to differentiate between benign software and malicious code, thereby enabling proactive identification and mitigation of potential threats.

The process flow for malware detection with machine learning begins with the collection of both malware and benign PE files. This collection process involves tools such as IDA Pro and PE View. Next, the PE files undergo disassembly. Subsequently, various features are extracted from the disassembled files using the Python programming language. These extracted features are then stored in a CSV or text file for preprocessing and as input for machine learning algorithms. Finally, machine learning methods are applied to classify the files as either malware or benign. The entire process is illustrated in Fig. 2.

In this work, Random Forest, Gradient Boosting Trees, Logistic Regression, LightGBM, XGBoost, and Decision Tree algorithms are used. LightGBM algorithm has been shown in Algorithm 1.

3.3. Explainable AI techniques

While AI models, including those used in malware detection, often exhibit impressive performance in tasks such as classification and prediction, their internal workings are often opaque and difficult to interpret. This lack of transparency raises concerns about the reliability and fairness of AI-powered applications, particularly in high-stakes domains such as healthcare, finance, and cybersecurity. It's important to maintain high learning performance in models, but equally essential, and sometimes required, is ensuring their explainability - that is, their capacity to be understood and trusted by humans [21]. XAI seeks to bridge this gap by offering methods, techniques, and tools for elucidating the rationale behind AI predictions and recommendations. By providing users with explanations that are comprehensible, intuitive, and actionable, XAI enables stakeholders to assess the trustworthiness of AI systems, identify potential biases or errors, and make informed decisions based on AI-generated insights.

3.3.1. Local Interpretable Model-Agnostic Explanations

LIME is a model-agnostic method that approximates the decision boundaries of a complex model by creating a locally interpretable surrogate model. It generates perturbations to input data and observes the corresponding changes in predictions, providing insights into how the model behaves in the vicinity of a specific instance [22].

Formally, given an instance S, and its neighborhood π_S, generated by perturbing elements of S, along with the black-box model f, LIME discovers an explainable model g by minimizing the following objective function (1):

$$\varphi(S) = \underset{g\in G}{\arg\min}\, L(f, g, \pi_S) + \Omega(g) \tag{1}$$

Algorithm 1 LightGBM Algorithm

```
• Input: Training data $(X, y)$, number of boosting rounds $T$, learning rate $\eta$, maximum depth $d$, other hyperparameters.
• Output: Trained model $f_T(x)$
1. Initialize the model with a constant value:
   $$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$
2. for $t = 1$ to $T$
   (a) Compute the negative gradients (pseudo-residuals) for the current model:
       $$r_i^{(t)} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{t-1}(x)}$$
   (b) Construct a regression tree to fit the pseudo-residuals:
       $$h_t = \arg\min_{h\in H} \sum_{i=1}^{n} \left(r_i^{(t)} - h(x_i)\right)^2$$
   (c) Compute the leaf weights of the tree by solving:
       $$w_j = \arg\min_{w} \sum_{i\in I_j} L(y_i, f_{t-1}(x_i) + w)$$
       where $I_j$ is the set of indices of the data points assigned to leaf $j$.
   (d) Update the model:
       $$f_t(x) = f_{t-1}(x) + \eta h_t(x)$$
3. end for
4. Return $f_T(x)$
```

Here, $L(f, g, \pi_S)$ represents the unfaithfulness of g concerning f, while $\Omega(g)$ denotes gs complexity. It is important to highlight that LIME, being a model-agnostic technique, operates independently of f [2].

3.3.2. SHapley Additive exPlanations

SHAP values are based on cooperative game theory and provide a way to fairly distribute the contribution of each feature to the prediction. In the context of XAI, SHAP values offer a global understanding of feature importance across the entire dataset, aiding in the interpretation of model decisions [2,8,11,23].

Shapley values are instrumental in making machine learning models more interpretable. They achieve this by providing local explanations through the assignment of a numerical measure, called credit, to each input feature within the model, particularly in models based on decision trees. These local explanations are then aggregated to form a comprehensive global structure, shedding light on the overall behavior of the model, whether it's a single decision tree or an ensemble of decision trees. Such ensembles may utilize algorithms like Random Forest, employing bagging, or LightGBM, utilizing boosting. In essence, the Shapley values contribute to understanding the significance of features in the decision-making process of the model, thus enhancing interpretability [8]. Shapley values are computed by sequentially integrating one feature at a time into the output function of the model, subject to certain conditions outlined in Eq. (2).

Algorithm 2 Compute SHAP Values

```
• Input: Model $f$, dataset $\mathcal{D}$
• Output: SHAP values $\phi$ for each feature
1. Initialize $N \leftarrow$ set of all features
2. Compute the expected value of the model prediction:
   $$\mathbb{E}[f(X)] = \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i)\in\mathcal{D}} f(x_i)$$
3. Initialize an empty dictionary to store SHAP values: $\phi \leftarrow \{i : 0 \text{ for } i \in N\}$
4. for each feature $i \in N$
   (a) Initialize an empty list to store marginal contributions: $MC \leftarrow []$
   (b) for each sample $x \in \mathcal{D}$
       i. Randomly sample subsets $S$ from $N \setminus \{i\}$
       ii. for each subset $S$
           A. Compute $f_x(S \cup \{i\})$
           B. Compute $f_x(S)$
           C. Compute the marginal contribution: $\Delta f = f_x(S \cup \{i\}) - f_x(S)$
           D. Append $\Delta f$ to $MC$
   (c) Compute the average marginal contribution for feature $i$: $\phi_i \leftarrow \frac{1}{|MC|} \sum MC$
5. end for
Output SHAP values $\phi$
```

$$f_x(S) = E[f(X) \mid \mathrm{do}(X_s = x_s)] \tag{2}$$

Here, S represents the set of characteristics on which to condition, while X denotes a random variable drawn from the input features M of the model, with x representing the input vector for the current prediction [8]. The algorithm for computing SHAP values is shown in Algorithm 2.

Shap and LIME are post-hoc, meaning they are implemented after the model is constructed and trained, and they are model-agnostic, treating f as a black box and solely analyzing its inputs and outputs [2].

In addition to Shapley and LIME methods, there are alternative approaches for interpreting machine learning models. These methods, such as Permutation Importance [24] and Global Model Explanation [25], offer distinct perspectives and techniques for understanding model behavior and feature importance.

In our study, SHAP and LIME were applied to interpret the predictions made by our machine learning models in the context of Android malware detection. For SHAP, we utilized the TreeSHAP algorithm specifically designed for tree-based models like LightGBM, which calculates the contribution of each feature to the final prediction. We generated SHAP values for each instance in our dataset, which allowed us to visualize and understand the impact of individual features on the model's output. These visualizations helped us identify key features that significantly influenced the classification decisions. SHAP values were

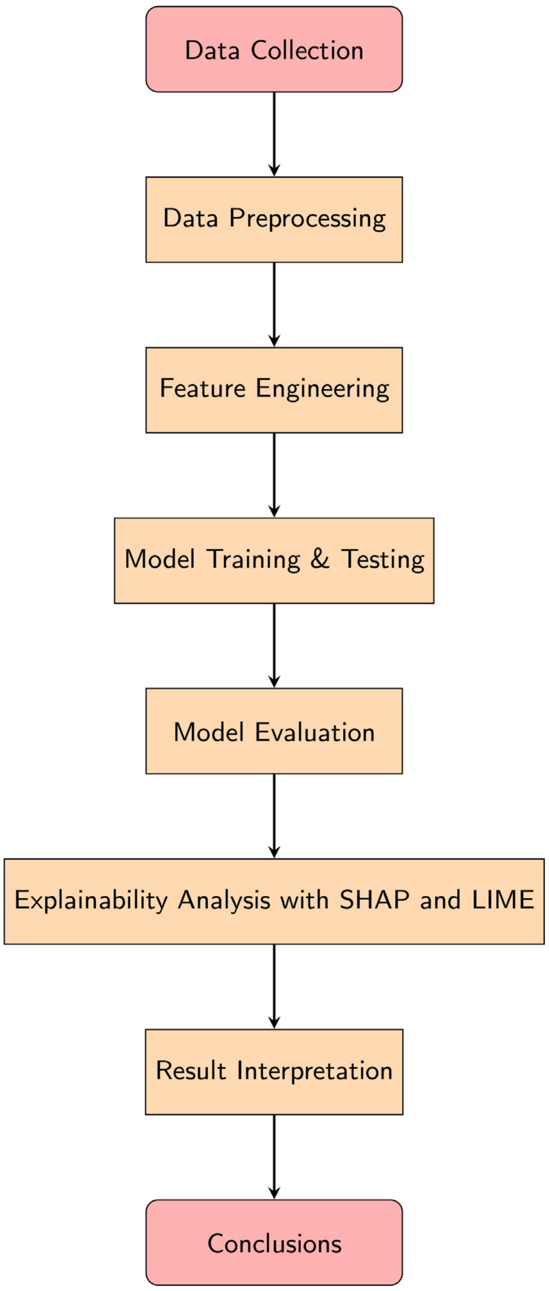


Fig. 3. General Workflow of the Study.

generated for each instance in our dataset using the Dataiku platform. For LIME, we selected a subset of instances from our test set to generate local explanations. LIME approximates the complex model with an interpretable model, such as a linear regression, around the prediction of interest. By perturbing the input features and observing the changes in the output, LIME provides insights into which features are most important for that particular prediction. This approach helped us understand how the model behaves for specific instances, offering a different perspective compared to SHAP's global interpretation. Python code was written to obtain LIME results.

In this study, a structured workflow is followed to ensure comprehensive analysis and robust results. The workflow includes data collection, preprocessing, feature engineering, model training, model evaluation, explainability analysis, result interpretation, and concluding insights. Each step is crucial for the overall success of the study. The detailed workflow is illustrated in Fig. 3.

Table 1
Distribution of file categories in CICMalDroid dataset.

Category	Count
Adware	1253
Banking	2100
SMS Malware	3904
Riskware	2546
Benign	1795

Table 2
Experimental environment parameters.

Parameter	Details
Platform	Dataiku
Version	12.6.2 (licensed)
Notebook Server Version	5.4.0-dku-10.0-0
Python Version	Python 3.6.8

4. Experimental results and analysis

4.1. Dataset description

The experimental setup utilized the CICMalDroid 2020 dataset [26,27], which consisted of more than 17,341 Android samples collected between December 2017 and December 2018. This dataset encompassed diverse categories such as Adware, Banking malware, SMS malware, Riskware, and Benign apps, providing comprehensive coverage of static and dynamic features. The dataset included a total of 470 features, making it a valuable resource for evaluating the effectiveness of XAI methods in malware detection on a wide range of sophisticated Android samples. Samples were collected from various sources, including the VirusTotal service, Contagio security blog, AMD, MalDozer, and other datasets utilized by recent research contributions, with the sources cited in the paper. The distribution of files within the dataset is illustrated in Table 1. To address the class imbalance in our dataset, we utilized the class_weight parameter during model training. This parameter allows us to assign different weights to each class, thereby compensating for the imbalanced distribution of samples across the classes.

4.2. Experimental setup

The experimental environment utilized for analysis and modeling involved the Dataiku platform, which provided a robust ecosystem for conducting advanced data analytics and machine learning tasks. Dataiku's integrated environment offered a comprehensive suite of tools and functionalities tailored for data preparation, feature engineering, model building, and evaluation. The Greenplum database was used to store and handle the CICMalDroid dataset. Parameters of experimental environment has been show in Table 2. The entire dataset is split into two portions for training and testing, with 80% allocated for training and testing, and 20% for validation.

4.3. Evaluation metrics

To evaluate the performance of the machine learning algorithms, several evaluation metrics were employed. Given that the dataset used is a multiclass dataset, these metrics were calculated separately for each class. An overall metric was calculated by aggregating the individual class metrics to provide a comprehensive evaluation of the model's performance.

Accuracy. Accuracy, as referenced in Eq. (3), measures the proportion of correctly classified instances out of the total instances in the dataset. It provides an overall indication of how well the algorithms are able to classify Android malware accurately.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

Precision. Precision, as referenced in Eq. (4), calculates the proportion of true positive predictions out of all positive predictions. It evaluates the ability of the algorithms to correctly identify malware instances without falsely classifying benign applications as malware.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

Recall (Sensitivity). Recall, as referenced in Eq. (5), measures the proportion of true positive predictions out of all actual positive instances. It assesses the algorithms' ability to detect all malware instances correctly without missing any.

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

F1-score. The F1-score, as referenced in Eq. (6), is the harmonic mean of precision and recall. It provides a balanced measure that considers both precision and recall, which is particularly useful in situations where there is an imbalanced dataset with differing numbers of malware and benign samples.

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{6}$$

ROC-AUC score. MROC_{AUC} as an equivalent of ROC_{AUC} for multi-class classification [28]. Let C be the number of classes,

$$\text{MROC}_{AUC} = \frac{1}{C \times (C-1)} \sum_{i=0}^{C-1} \sum_{\substack{j=0 \\ j \neq i}}^{C-1} A(i,j) \tag{7}$$

Detail of how $A(i,j)$ is computed.

Input arrays: Let y_{truth} be the array of ground truth class values (in $\{0, \dots, C-1\}$). It is of shape (M, 1), i.e. M rows and 1 column:

$$y_{\text{truth}} = \begin{pmatrix} 3 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \text{ M rows, values} \in \{0, \dots, C-1\} \tag{8}$$

Let y_{probas} be the array of predicted probabilities. Each column i corresponds to a class i, where values are probability estimates for the class i. It is of shape (M, C), i.e. M rows and C columns:

$$y_{\text{probas}} = \begin{pmatrix} 0.1 & 0.9 & \dots & 0 \\ 0.8 & 0 & \dots & 0.2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0.7 \\ 0 & 0.1 & \dots & 0.9 \end{pmatrix} \text{ M rows} \tag{9}$$

$A(i,j)$ computation: For every pair of classes $i \neq j$, let $L_{i,j}$ be the subset of rows of y_{truth} where the ground truth is either i or j.

$$L_{i,j} = \{k \in \{0, \dots, M-1\} \mid y_{\text{truth},k} = i \text{ or } y_{\text{truth},k} = j\} \tag{10}$$

With $y_{\text{truth},L_{i,j}}$ and $y_{\text{proba},L_{i,j}}$ the corresponding arrays with only these rows

- Let $y_{\text{truth},L_{i,j},\text{binarized}}$ be a copy of $y_{\text{truth},L_{i,j}}$ with all values at j set to 0 and all at i set to 1
- Note $y_{\text{probas},L_{i,j},\text{col } i}$ the column i of $y_{\text{probas},L_{i,j}}$

$$A(i,j) = \text{ROC}_{AUC}(y_{\text{truth},L_{i,j},\text{binarized}}, y_{\text{probas},L_{i,j},\text{col } i}) \tag{11}$$

with ROC_{AUC} the usual binary classification ROC AUC metric.

In the formulas, TP (True Positives) represents the count of correctly predicted malware instances. TN (True Negatives) indicates the number of instances accurately predicted as benign. FP (False Positives) denotes the count of benign instances incorrectly classified as malware. FN (False Negatives) signifies the number of malware instances mistakenly classified as benign.

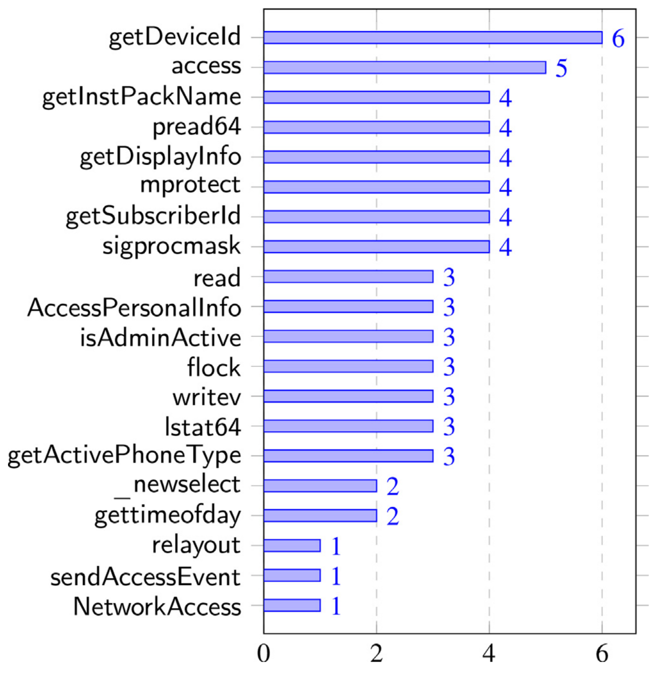


Fig. 4. Feature importance for LightGBM model.

4.4. Metric results for different machine learning models

The testing results, as depicted in Table 3, showcase the performance of various algorithms in terms of train time, accuracy, precision, recall, F1 score, and ROC-AUC. Notably, LightGBM exhibit the highest accuracy at 0.93 F1-score. Table 4 describes the confusion matrix, providing a clear understanding of how each class performed in terms of true positives and misclassifications based-on LightGBM model predictions.

4.5. SHAP values based on LightGBM classifier

In the Fig. 4, feature importance plot reveals that certain features like getDeviceId and access are critical in the detection of malware in Android applications. These features should be given special attention in both the development and analysis phases of the model. On the other hand, features with lower importance scores, while still useful, have less impact on the model's overall performance and might be considered for removal or further analysis to determine their necessity. This insight helps in understanding which features are driving the model's decisions and can aid in refining and optimizing the model for better performance.

Summary plot of different class labels has been shown in Figs. 5–9 for class 'SMS malware', 'Riskware', 'Banking', 'Benign', 'Adware' respectively. The vertical axis depicts the features sorted by importance, with the most significant ones at the top. Meanwhile, the horizontal axis shows the SHAP value, denoting the extent of change in log odds. Each dot on the plot corresponds to a data row, with its color representing the feature value red for high values and blue for low ones.

For the SMS Malware Class, features like getDeviceId, access, pread64, getInstallerPackageName, and getDisplayInfo significantly increase the likelihood of an application being classified as SMS malware when they have high values. This indicates that the presence or high numerical values of these features strongly correlate with SMS malware. Similarly, isAdminActive and flock also show a substantial impact, particularly when they have high values, further emphasizing their importance in identifying SMS malware.

In the Riskware Class, getDeviceId, access, getInstallerPackageName, and related features exhibit a similar pattern where high values increase the prediction probability for Riskware. This consistent pattern across multiple features suggests that these permissions and identifiers are crit-

Table 3
Performance metrics of various models.

Name	Train time	Accuracy	Precision	Recall	F1-score	ROC AUC
XGBoost	6m 22s	0.92	0.90	0.90	0.90	0.99
LightGBM	***7m 15s***	***0.94***	***0.92***	***0.93***	***0.93***	***0.99***
Decision Tree	3m 36s	0.75	0.78	0.71	0.72	0.89
Gradient Boosted Trees	4m 15s	0.92	0.91	0.90	0.90	0.99
SVM	7m 34s	0.78	0.76	0.76	0.76	0.93
Logistic Regression	35m 59s	0.84	0.81	0.81	0.81	0.94
Random forest	3m 41s	0.91	0.90	0.90	0.90	0.97

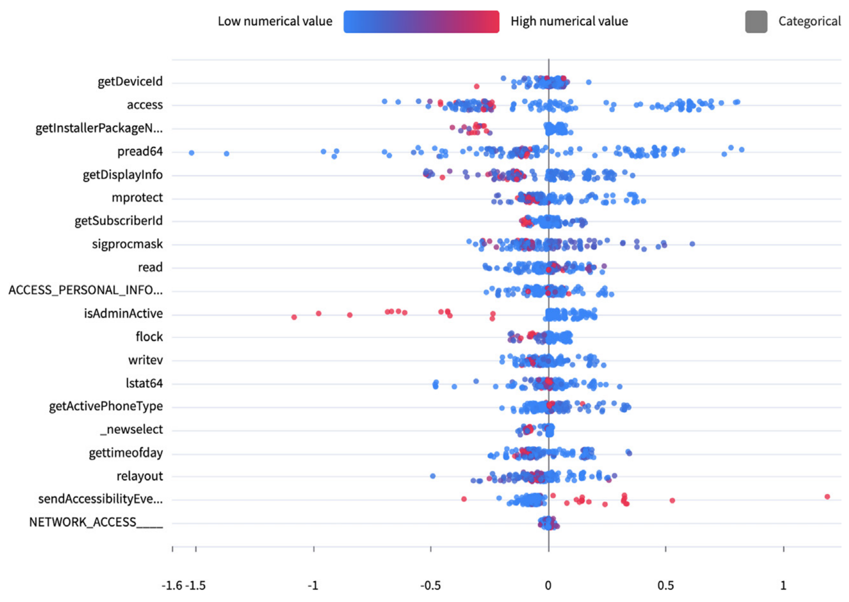


Fig. 5. SMS malware class.

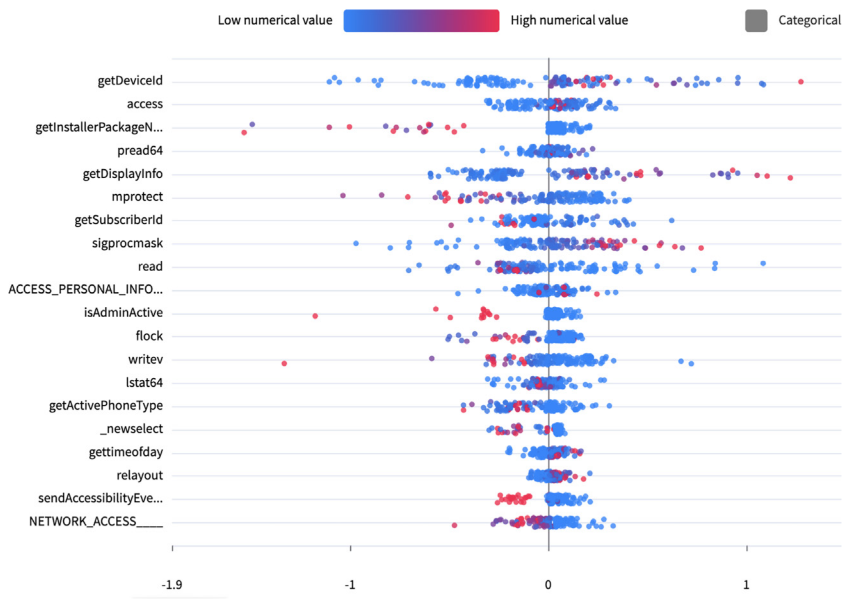


Fig. 6. Riskware class.

ical for distinguishing Riskware. Additionally, features like mprotect and getSubscriberId also play significant roles, reinforcing the model's reliance on these attributes for accurate classification.

For the Adware Class, the impact of getDeviceId, access remains prominent, with high values of these features leading to an increased likelihood of adware detection. Features such as sigprocmask and getDisplayInfo also contribute notably to the predictions, indicating their relevance in identifying adware behaviors.

In contrast, for the Benign Class, the model's behavior is somewhat different. While getDeviceId and access are still crucial, high values of these features actually decrease the likelihood of an application being classified as benign. This inverse relationship underscores the importance of these features in distinguishing benign applications from various types of malware. Other features like flock and isAdminActive show low impact, suggesting that these features are less relevant for identifying benign applications.

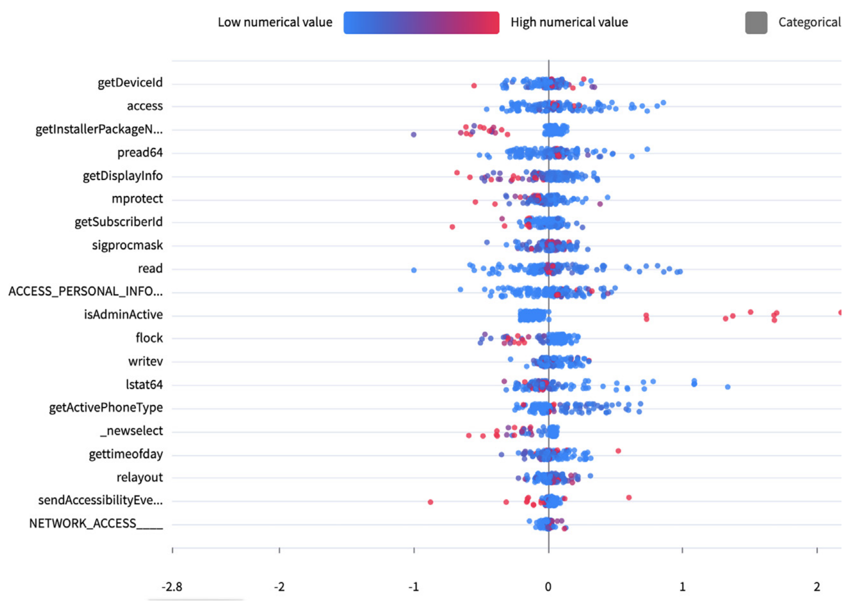


Fig. 7. Banking class.

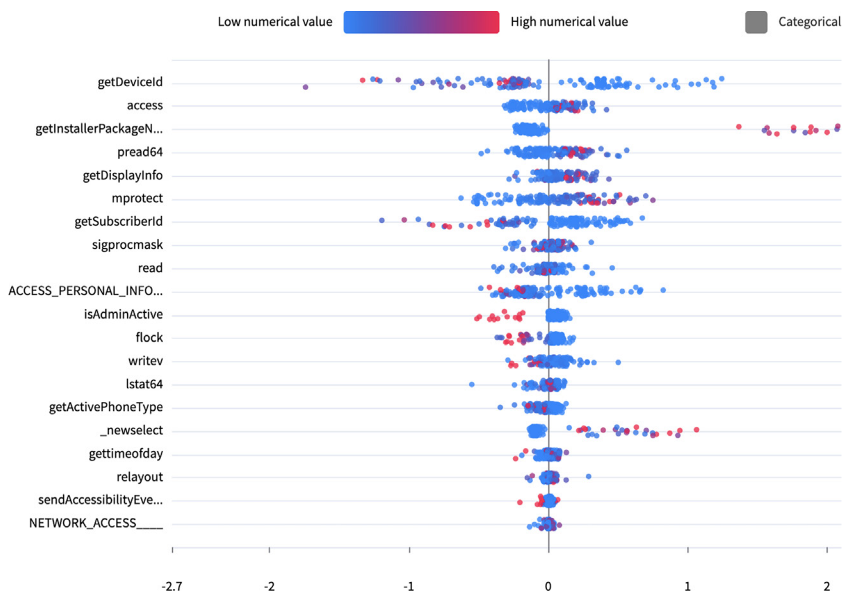


Fig. 8. Benign class.

Table 4
Confusion Matrix for CICMalDroid Dataset. Class 1: Benign, Class 2: Banking, Class 3: SMS malware, Class 4: Riskware, Class 5: Adware.

Actual\Predicted	Class 1	Class 2	Class 3	Class 4	Class 5	Total
Class 1	179	2	0	7	4	192
Class 2	5	316	5	8	10	344
Class 3	1	4	604	2	1	612
Class 4	20	6	1	377	15	419
Class 5	6	3	0	13	252	274
Total	211	331	610	407	282	1841

For the Banking Class, features such as getDeviceId, access, and sigprocmask again show significant influence, with high values increasing the probability of classifying an application as banking-related malware. Features like getSubscriberId and mprotect also play important roles, highlighting the critical permissions and identifiers needed to accurately detect banking malware.

4.6. *LIME results based on LightGBM classifier*

Local Interpretable Model-agnostic Explanations, diverges from offering a holistic comprehension of the model across the entire dataset. Instead, it zooms in on elucidating the model's predictions for individual instances.

Setting up a LIME explainer involves two primary steps: (1) importing the lime module, and (2) fitting the explainer using the training data and the corresponding target labels. At this stage, the mode is configured for classification, aligning with the specific task at hand. From the Fig. 10, it is obvious that, outcome comprises three primary elements arranged from left to right: (1) the model's predictions, (2) the contributions of features, and (3) the actual value corresponding to each feature. Fig. 10 displays a random sample result from LIME.Looking through this random sample LIME result, we observe that the sample is predicted as adware with 98% confidence. The decision is driven by features such as ”getdisplayinfo”, ”futex”, ”munmap”, ”writev”, ”sigprocmask”, ”pread64”, and ”mprotect”. These features demonstrate higher

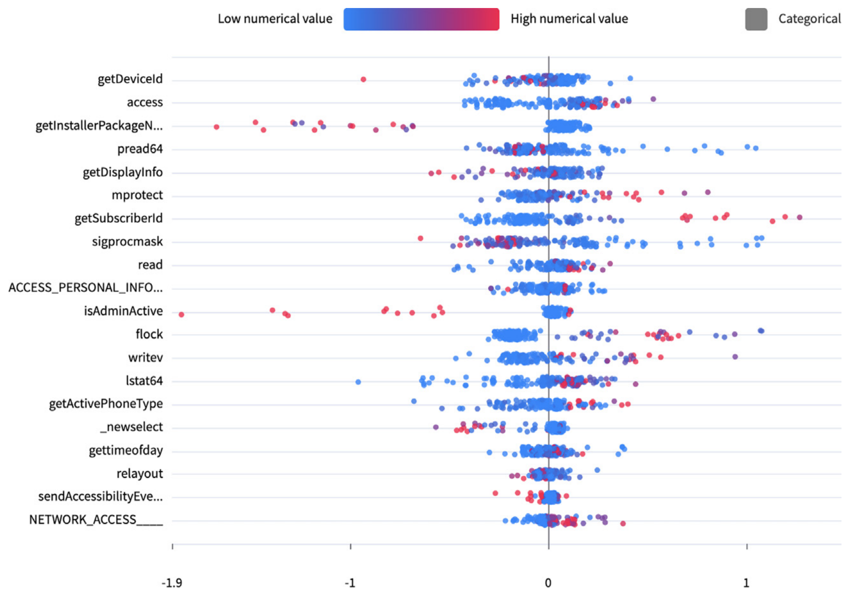
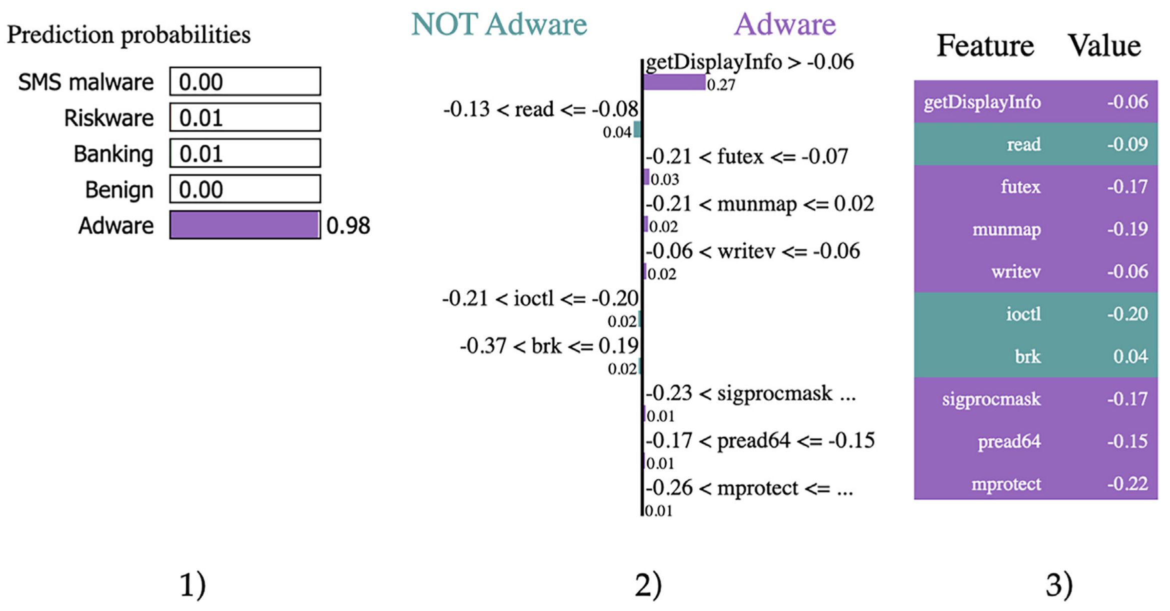


Fig. 9. Adware class.



Feature	Value
getDisplayInfo	-0.06
read	-0.09
futex	-0.17
munmap	-0.19
writev	-0.06
ioctl	-0.20
brk	0.04
sigprocmask	-0.17
pread64	-0.15
mprotect	-0.22

Fig. 10. LIME results of random sample.

importance compared to others, influencing the model's decision to classify the sample as not malware.

5. Limitations

Implementing XAI in real-time environments for malware detection on Android devices presents several challenges. One primary issue is the performance overhead, as XAI techniques often require additional computational resources to generate explanations for machine learning model decisions. This can introduce latency and affect the performance of real-time malware detection systems that need to operate swiftly to prevent infections. Additionally, the complexity of explanations generated by XAI techniques must be easily understandable by security analysts. Creating clear and concise explanations for complex models in real-time is challenging, especially when dealing with high-dimensional data typical of malware detection. Ongoing research and development are required to optimize XAI methods for real-time application in resource-constrained and dynamically changing environments.

6. Conclusions and future works

By integrating XAI techniques, the interpretability and transparency of malware detection models were enhanced, fostering trust and understanding within the cybersecurity landscape. The CICMalDroid dataset, along with SHAP and LIME analyses, enriched the exploration of decision-making processes, revealing actionable insights into feature importance. The LightGBM algorithm excelled with high precision, recall, F1 scores, and ROC-AUC, demonstrating the effectiveness of gradient boosting techniques. Shapley values enhanced model interpretability by highlighting the importance of individual features, while post-hoc techniques like LIME provided effective, model-agnostic explanations for predictions.

Future work will focus on exploring additional XAI methods to provide deeper insights into the decision-making process of the models. Testing the robustness of the architecture against adversarial attacks is planned to understand how perturbations can affect the quality of the explanations generated. Additionally, it is planned to extend the scope to more complex models, including deep learning architectures, to understand and explain their decision-making processes. Furthermore, there are plans to take trend datasets and utilize the most recent period of time datasets to monitor and check the PSI and model drift. This will also involve checking changes in feature importance over time to ensure the robustness and reliability of the suggested model. Future work will also evaluate XAI methods to ensure a deeper understanding and reliability of the model explanations.

Declaration of Generative AI in the writing process

During the preparation of this work the author used ”ChatGPT 4o” in order tolanguage editing and refinement. After using this tool/service, the authorreviewed and edited the content as needed and takes full responsibility for the content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

E. Baghirov: Writing – review & editing, Writing – original draft, Visualization, Supervision, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

References

[1] K. Santosh, P. Govind, S. Kumar, Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: a comprehensive review, Telemat. Inform. Rep. 14 (2024) 1–18, doi:10.1016/j.teler.2024.100130.
[2] G. Antonio, V. Gatta, V. Moscato, Explainability in AI-based behavioral malware detection systems, Comput. Secur. 141 (2024) 1–17, doi:10.1016/j.cose.2024.103842.
[3] P. Maniriho, A.N. Mahmood, M. Chowdhury, A systematic literature review on windows malware detection: Techniques, research issues, and future directions, J. Syst. Softw. 209 (2024) 1–31, doi:10.1016/j.jss.2023.111921.
[4] 2024. ThreatDown by Malwarebytes, State of Malware report. 1–31
[5] Avtest Independent IT-Security Institute, Malware Statistics, 2024, https://www.av-test.org/en/statistics/malware.
[6] B. Ahmed, J. Kalita, M. Bensaoud, A survey of malware detection using deep learning, Mach. Learn. Appl. 16 (2024), doi:10.1016/j.mlwa.2024.100546.
[7] E. Baghirov, A. Baku, Evaluating the performance of different machine learning algorithms for android malware detection, 2023 5th International Conference on Problems of Cybernetics and Informatics (PCI) (2023) 1–4, doi:10.1109/PCI60110.2023.10326006.
[8] R. Kumar, S. Geetha, Effective malware detection using shapely boosting algorithm, Int. J. Adv. Comput. Sci. Appl. 13 (1) (2022) 101–111, doi:10.14569/IJACSA.2022.0130113.
[9] B. Cheng, J. Ming, E.A. Leal, H. Zhang, J. Fu, G. Peng, J.Y. Marion, Obfuscation-resilient executable payload extraction from packed malware, 2021. 30th USENIX Security Symposium. 3451-3468
[10] Z. Bazrafshan, H. Hashemi, S. Fard, A survey on heuristic malware detection techniques, in: 5th Conf. Inf. Knowl. Technol., IEEE, 2013, p. 113-120, doi:10.1109/IKT.2013.6620049.
[11] R. Kumar, S. Geetha, Explainable machine learning for malware detection using ensemble bagging algorithms, in: Fourteenth Int. Conf. Contemp. Comput., ACM, New York, USA, 2022, p. 453-460, doi:10.1145/3549206.3549284.
[12] J. Song, S. Choi, J. Kim, A study of the relationship of malware detection mechanisms using Artificial Intelligence, ICT Express, 2024, pp. 1–18, doi:10.1016/j.icte.2024.03.005.
[13] Y. Liu, C. Tantithamthavorn, L. Li, Y. Liu, Explainable AI for android malware detection: towards understanding why the models perform so well?, 2022. ArXiv:2209.00812. 1–12
[14] H. Bragancha, V. Rocha, E. Souto, Explaining the effectiveness of machine learning in malware detection: insights from explainable AI, in: Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, 2024, p. 181-194, doi:10.5753/sbseg.2023.233595.
[15] M. Kinkead, M. Stuart, N. McLaughlin, Towards explainable CNNs for android malware detection, Comput. Sci. 184 (2021) 959-965, doi:10.1016/j.procs.2021.03.118.
[16] F. Ullah, A. Alsirhani, M.M. Alshahrani, A. Alomari, H. Naeem, S.A. Shah, Explainable malware detection system using transformers-based transfer learning and multi-model visual representation, Sensors 22 (18) (2022) 6766, doi:10.3390/s22186766.
[17] H. Bostani, V. Moonsamy, Evadedroid: a practical evasion attack on machine learning for black-box android malware detection, Comput. Secur. 139 (2024) 103676, doi:10.1016/j.cose.2023.103676.
[18] E.M. Rudd, D. Krisiloff, S. Coull, D. Olszewski, E. Raff, J. Holt, Efficient malware analysis using metric embeddings, Digit. Threats 5 (1) (2024) 4, doi:10.1145/3615669.
[19] B. Menaouer, I.A. E, N. M, Android malware detection approach using stacked autoencoder and convolutional neural networks, Int. J. Intell. Inf. Technol. 19 (1) (2023) 1–22, doi:10.4018/IJIIT.329956.
[20] L.X. Ling, L. Wu, J. Zhang, Adversarial attacks against windows PE malware detection: a survey of the state-of-the-art, Comput. Secur. 128(2023). 10.1016/j.cose.2023.103134
[21] S. Ali, T. Abuhmed, S. El-Sappagh, Explainable artificial intelligence (XAI): what we know and what is left to attain trustworthy artificial intelligence, Inf. Fusion 99(2023) 1–52. 10.1016/j.inffus.2023.101805
[22] M.T. Ribeiro, S. Singh, C. Guestrin, Why should i trust you?: explaining the predictions of any classifier, 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (2016) 1–10. Https://arxiv.org/abs/1602.04938v3
[23] L.S. Shapley, H. Kuhn, A. Tucker, A value for n-person games, in: Contributions to the Theory of Games II, Princeton University Press, Princeton, 1953, pp. 307–317.
[24] L. Breiman, R. Forests, Mach. Learn. 45 (2001) 5–32.
[25] C. Yang, R. Anand, R. Sanjay, Global model interpretation via recursive partitioning, 2018. ArXiv:1802.04253v2. 1–8
[26] S. Mahdavifar, A.F.A. Kadir, R. Fatemi, D. Alhadidi, A.A. Ghorbani, Dynamic android malware category classification using semi-supervised deep learning, 18th IEEE Int. Conf. Dependable, Autonomic, Secure Comput. (DASC) (2020), doi:10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.
[27] S. Mahdavifar, D. Alhadidi, A.A. Ghorbani, Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder, J. Netw. Syst. Manag. 30 (1) (2022) 1–34, doi:10.1007/s10922-021-09634-4.
[28] Multiclass Receiver Operating Characteristic (ROC), https://scikit-learn.org.