Pertanika J. Sci. & Technol. 34 (5): 2765 - 2790 (2026)

PERTANIKA JOURNALS

SCIENCE & TECHNOLOGY

Journal homepage: http://www.pertanika.upm.edu.my/

An Integrated Multi-Source Framework for Explainable Android Malware Detection with SOC Integration

Pallavi M. Shimpi[1]* and Nitin Namdeo Pise[1]

Department of Computer Engineering and Technology, Faculty of Engineering and Technology, Dr. Vishwanath Karad MIT World Peace University, Survey No. 124, Paud Road, Kothrud, 411038 Pune, Maharashtra, India

ABSTRACT

Neurosec Malware Scanner is an integrated malware-analysis framework that combines multi-source threat intelligence, static machine-learning classification, SHAP-based explainability, and Security Operations Centre (SOC)-oriented result presentation within a unified workflow. The system queries multiple file-hash threat-intelligence services in parallel and consolidates their outputs using a weighted aggregation mechanism. In parallel, a Python-based machine-learning pipeline extracts static malware features and evaluates four supervised classifiers: Random Forest, XGBoost, Gradient Boosting, and Logistic Regression. The framework is evaluated using five publicly available benchmark datasets: SOMLAP, TUANDROMD, Drebin, EMBER, and MH-100K. The best classification results obtained in the experiments are 99.6% on SOMLAP, 99.8% on TUANDROMD, 98.9% on Drebin, 94.3% on EMBER, and 86.7% on MH-100K. To improve interpretability, the framework incorporates SHAP-based explanations that identify the features contributing to individual model predictions. The system additionally employs cryptographic hashing, result caching, structured logging, and dashboard-based visualization to support repeatable malware-triage workflows. The proposed framework demonstrates the feasibility of integrating machine-learning detection, explainability, and threat-intelligence enrichment within an analyst-oriented malware-analysis pipeline. However, validation using recent real-world malware samples and deployment in operational SOC environments remains necessary to establish production-level effectiveness.

Keywords: Android malware detection, explainable AI, machine learning, SOC integration, static analysis, threat intelligence

ARTICLE INFO

Article history:
Received: 19 June 2026
Accepted: 02 September 2026
Published: 30 October 2026

DOI: https://doi.org/10.47836/pjst.34.5.10

mspallavi2710@gmail.com (Pallavi M. Shimpi)
nitin.pise@mitwpu.edu.in (Nitin Namdeo Pise)
* Corresponding author

INTRODUCTION

Android has become one of the most widely used mobile operating systems worldwide, making it an important target for malicious

e-ISSN: 2231-8526

© Universiti Putra Malaysia Press

applications and cyberattacks (Arp et al., 2014; Zhou & Jiang, 2012). The increasing availability of Android applications and the rapid evolution of malware techniques have created significant challenges for conventional security mechanisms. Malware authors increasingly employ obfuscation, code modification, repackaging, and other evasion techniques to bypass signature-based and rule-based detection mechanisms (Christodorescu & Jha, 2003; You & Yim, 2010). Moreover, the limitations of static analysis under certain malware evasion conditions have been reported in previous studies (Moser et al., 2007). Consequently, effective Android malware detection requires approaches that can identify malicious patterns beyond previously known signatures.

Machine-learning techniques offer greater adaptability to emerging malware patterns than traditional signature-based approaches, as they can learn discriminative characteristics from malware-related features (Hardy et al., 2016; Yuan et al., 2014). Machine-learning-based Android malware detection has therefore received considerable attention for analysing application permissions, Application Programming Interface (API) calls, intents, opcodes, source-code characteristics, and other static or behavioural features (Arp et al., 2014; Prasad et al., 2025; Santos et al., 2013; Yuan et al., 2014). Deep learning approaches have further demonstrated their ability to learn complex representations from high-dimensional malware data (Hardy et al., 2016; Yuan et al., 2014). However, many machine-learning and deep-learning models operate as black-box systems, which can limit analysts' understanding of the factors underlying individual predictions and may reduce confidence in automated malware detection results (Galli et al., 2024; Rjoub et al., 2023; Shimpi & Pise, 2026). A model that provides only a malicious or benign prediction may therefore be insufficient for security analysts who need to understand why an application has been classified as malicious.

This limitation motivates the use of Explainable Artificial Intelligence (XAI) techniques in malware analysis. XAI methods can provide information about the features that contribute to individual model predictions and can consequently improve the interpretability of machine-learning-based detection systems (Galli et al., 2024; Lundberg & Lee, 2017; Ribeiro et al., 2016; Rjoub et al., 2023). Lundberg and Lee (2017) introduced Shapley Additive exPlanations (SHAP), which provides feature-attribution information for explaining model predictions, while Ribeiro et al. (2016) proposed LIME as a model-agnostic approach for interpreting individual predictions. XAI has subsequently received increasing attention in cybersecurity because interpretable predictions can help analysts understand and validate automated security decisions (Rjoub et al., 2023). Recent research has also investigated explainability in AI-based malware detection systems (Galli et al., 2024). In Android malware analysis, feature-level explanations can help analysts identify permissions, API-related characteristics, source-code attributes, and other application features that contribute to a suspicious classification (Arp et al., 2014; Prasad et al., 2025;

Shimpi & Pise, 2026). However, explainability alone does not provide complete contextual information about the identified application, associated indicators, or potentially malicious infrastructure.

Machine-learning predictions can identify suspicious applications, but classification results alone may provide limited information for subsequent security investigation. Cyber Threat Intelligence (CTI) can complement automated detection by providing reputation information, indicators of compromise, and contextual information associated with previously observed malicious activities (Sun et al., 2023). Threat intelligence information can therefore provide additional evidence to help analysts assess the relevance and severity of a detected threat. However, threat-intelligence services are frequently used as independent sources of information or lookup data rather than being systematically integrated with machine-learning predictions and model explanations within a unified malware-analysis workflow. This creates an integration challenge between automated detection, interpretable decision-making, and contextual threat intelligence.

The practical importance of such integration is particularly evident in Security Operations Centre (SOC) environments, where analysts must process multiple alerts and correlate evidence from different security sources. Existing Android malware detection approaches have primarily concentrated on classification performance (Arp et al., 2014; Prasad et al., 2025; Santos et al., 2013; Yuan et al., 2014), while XAI-oriented studies have focused mainly on improving model transparency (Galli et al., 2024; Lundberg & Lee, 2017; Ribeiro et al., 2016; Rjoub et al., 2023; Shimpi & Pise, 2026). Similarly, threat-intelligence research has emphasised the collection, mining, and utilisation of security intelligence (Sun et al., 2023). These research directions provide important capabilities, but they are not necessarily integrated into a single Android malware-analysis workflow that connects a model prediction with its feature-level explanation and external threat-intelligence evidence.

Our previous research has investigated Android malware detection from complementary machine-learning perspectives. An adaptive swarm optimisation-based neural network was developed for Android malware detection (Shimpi & Pise, 2023b). This work was followed by an empirical investigation of machine-learning-based malware detection on Android mobile phones (Shimpi & Pise, 2023a). More recently, an optimised and explainable attentional convolutional LSTM model was proposed to improve Android malware detection while providing model interpretability (Shimpi & Pise, 2026). These studies establish a progression from conventional machine-learning-based detection toward optimised and explainable deep-learning models. The present study extends this research direction beyond an individual detection model by integrating classification, explainability, threat-intelligence correlation, and operational presentation into a unified SOC-oriented analytical framework.

Based on the above limitations, the key research gap this study addresses is the limited integration of automated malware classification, feature-level explainability, multi-source threat-intelligence enrichment, and SOC-oriented operational visualisation within a single Android malware-analysis workflow. The proposed work does not claim novelty from independently introducing machine learning, SHAP, or threat intelligence, since these technologies have been extensively investigated (Galli et al., 2024; Lundberg & Lee, 2017; Ribeiro et al., 2016; Rjoub et al., 2023; Sun et al., 2023). Instead, the contribution lies in integrating these established capabilities so that an analyst can move from automated malware prediction to an explanation of the prediction, and subsequently to contextual threat-intelligence evidence, within the same analytical process.

In this study, a unified SOC-oriented malware analysis framework is developed by integrating static application analysis, machine-learning-based malware classification, SHAP-based explainability, multi-source threat intelligence correlation, and operational visualisation. The framework first extracts relevant application characteristics and applies machine-learning models to classify potentially malicious applications. The prediction results are subsequently complemented by feature-level SHAP explanations and information from external threat intelligence sources. The resulting information is presented through an operational workflow intended to support analyst-oriented malware assessment and investigation.

The framework is evaluated on multiple benchmark datasets that represent heterogeneous malware feature spaces. The primary evaluation focuses on Android malware datasets, consistent with the research's main objective. The EMBER Windows Portable Executable (PE) dataset is used only as an auxiliary cross-platform evaluation environment (Anderson & Roth, 2018). The inclusion of EMBER is intended to examine whether the analytical architecture can operate across different malware representations rather than to claim that Windows malware and Android malware constitute equivalent detection tasks. Because the feature spaces and operating-system characteristics differ, Android and Windows results are analysed and reported separately. This design allows the Android experiments to remain the primary evidence for the research objective while providing additional insight into the architectural adaptability of the proposed framework.

The main contributions of this research are a unified SOC-oriented malware-analysis framework that integrates static analysis, machine-learning classification, SHAP-based explainability, external threat-intelligence correlation, and operational visualisation into a common analytical workflow.

An explainability-driven analysis mechanism is incorporated to provide feature-level information underlying malware predictions, enabling analysts to examine the factors contributing to individual classification decisions.

A multi-source threat-intelligence aggregation mechanism is integrated to supplement machine-learning predictions with external reputation information and security intelligence, thereby providing additional contextual evidence for malware investigation.

A cross-dataset evaluation strategy is employed to examine the proposed analytical pipeline across heterogeneous malware datasets, with Android datasets representing the primary evaluation environment and the EMBER Windows PE dataset providing auxiliary cross-platform validation (Anderson & Roth, 2018).

The framework is designed as an analyst-oriented decision-support pipeline rather than solely as a classification model, connecting automated detection results with interpretable evidence and contextual threat intelligence. Therefore, the novelty of the proposed work does not primarily arise from introducing a new standalone machine-learning classifier, SHAP method, or threat-intelligence service. Instead, it lies in integrating these capabilities into a unified malware-analysis workflow that connects automated detection with feature-level explanation and actionable security intelligence. This integration provides a practical approach to malware triage by enabling analysts to progress from an automated prediction to an explanation of that prediction, and subsequently to supporting threat intelligence evidence, within the same analytical process.

RELATED WORK

Signature-Based and Static/Dynamic Malware Analysis

Traditional malware detection systems have predominantly relied on signatures and predefined patterns to identify known malicious software. Signature-based techniques are effective for previously identified malware because they can provide accurate detection with relatively low computational requirements (Christodorescu & Jha, 2003). However, such approaches are less effective against malware that employs code modification, obfuscation, polymorphism, or other evasion techniques. Previous studies have demonstrated that even relatively small modifications to malicious code can reduce the effectiveness of purely signature-based detection approaches (Moser et al., 2007; You & Yim, 2010).

Malware analysis techniques can broadly be categorised into static and dynamic approaches. Static analysis examines the characteristics of an application or executable without executing the potentially malicious program. Such characteristics may include permissions, API calls, strings, structural properties, and opcode-related information. Static analysis can provide relatively efficient and scalable malware screening because the suspicious application need not be executed (Arp et al., 2014; Christodorescu & Jha, 2003). However, its effectiveness may be affected when malware deliberately modifies or hides its static characteristics (Moser et al., 2007).

Dynamic analysis, in contrast, observes malware behaviour during execution, including system calls, API interactions, network communication, and other runtime activities.

Dynamic analysis can provide behavioural information that may not be available through static inspection alone (Egele et al., 2012). However, execution-based analysis generally requires additional computational resources and controlled environments such as sandboxes. Consequently, static analysis remains attractive for large-scale malware screening, while hybrid approaches can combine static and dynamic characteristics when computational resources permit.

For Android malware detection, static analysis has been particularly important because application permissions, API calls, intents, and other application-level characteristics can be extracted without executing the application. DREBIN demonstrated the effectiveness of combining multiple lightweight static features for Android malware detection (Arp et al., 2014). Zhou and Jiang (2012) also provided an extensive characterisation of Android malware and its evolution, establishing the importance of application-level characteristics in identifying malicious Android applications.

Machine-Learning-Based Android Malware Detection

Machine-learning approaches have been increasingly adopted to overcome some limitations of signature-based malware detection. Rather than relying exclusively on predefined signatures, machine-learning models can learn discriminative patterns from extracted malware features and subsequently classify previously unseen samples. DREBIN demonstrated that supervised machine learning could effectively detect Android malware using static features derived from applications (Arp et al., 2014). Yuan et al. (2014) proposed DroidSec and demonstrated the application of deep learning to Android malware detection. Deep learning approaches can automatically learn complex feature representations and may reduce dependence on manually designed detection rules. Opcode-based representations have also been investigated for machine-learning-based malware detection. Santos et al. (2013) demonstrated that opcode sequences can provide useful features for identifying unknown malware. Such research indicates that different representations of executable or application characteristics can contribute to automated malware classification. The authors' previous research has also examined several machine-learning approaches for Android malware detection. Shimpi and Pise (2023b) proposed an adaptive swarm optimisation-based neural network for Android malware detection. An empirical investigation of machine learning techniques for Android mobile malware detection was subsequently presented in Shimpi & Pise (2023a). More recently, an optimised and explainable attentional convolutional LSTM model was proposed for Android malware detection (Shimpi & Pise, 2026). These studies demonstrate a progression from conventional machine-learning-based classification toward optimised and explainable deep-learning approaches. Prasad et al. (2025) proposed AndroMD, an Android malware detection framework based on source-code analysis and permission scanning. This work further demonstrates the value of combining multiple application-level characteristics for Android malware detection. Importantly,

AndroMD should be cited as Prasad et al. (2025), rather than You and Yim (2010). You and Yim (2010) address malware obfuscation techniques and should not be cited as the source of the AndroMD framework. Although these machine-learning approaches demonstrate strong potential for malware classification, classification performance alone does not necessarily provide sufficient information for security analysts. In operational environments, analysts may also require an explanation of why a particular application was classified as malicious, as well as additional contextual information to support the investigation.

Explainable Artificial Intelligence for Malware Detection

The increasing use of machine-learning and deep-learning models in cybersecurity has created a need for interpretable decision-making. Complex models may achieve high classification performance while offering limited information about the factors underlying individual predictions. This black-box characteristic can make it difficult for analysts to validate automated decisions and investigate suspicious applications.

SHapley Additive exPlanations (SHAP) provide a feature-attribution mechanism for explaining individual model predictions (Lundberg & Lee, 2017). SHAP assigns contribution values to input features, allowing analysts to examine whether individual features increase or decrease the predicted probability of a particular class. LIME provides another model-agnostic approach for generating local explanations of machine-learning predictions (Ribeiro et al., 2016).

XAI has increasingly been investigated in cybersecurity because interpretable model outputs can help analysts understand automated security decisions. Rjoub et al. (2023) reviewed XAI approaches for cybersecurity and highlighted the importance of interpretability for security applications. Galli et al. (2024) further investigated explainability in AI-based behavioural malware detection systems and demonstrated the relevance of feature-level explanations in malware analysis. For Android malware detection, XAI can provide additional information about the application characteristics contributing to a malicious prediction. Such explanations may include contributions from permissions, API-related features, code characteristics, or other static attributes (Arp et al., 2014; Prasad et al., 2025; Shimpi & Pise, 2026). However, an explanation of a model prediction does not, by itself, provide information about the application's reputation, associated indicators of compromise, or related malicious infrastructure. Therefore, XAI addresses the interpretability problem but does not completely address the contextual requirements of malware investigation.

Cyber Threat Intelligence for Malware Investigation

Cyber Threat Intelligence (CTI) provides contextual information that can complement automated malware detection. CTI may include indicators of compromise, reputation

information, relationships between malicious entities, previously observed attack information, and other security intelligence. Sun et al. (2023) surveyed cyber threat intelligence mining and highlighted the importance of intelligence-driven approaches for proactive cybersecurity defence.

In a malware analysis workflow, external threat intelligence can provide additional evidence once an application or file has been identified as suspicious. For example, reputation information or previously observed indicators can help analysts determine whether a detected sample is associated with known malicious activity. Such contextual information can complement the output of machine-learning classifiers. However, threat intelligence and machine-learning detection are frequently treated as separate analytical functions. A machine-learning model may produce a probability or class label, while threat-intelligence platforms may separately provide reputation or indicator information. Similarly, XAI systems may explain the model decision without incorporating external intelligence. Consequently, the outputs of these components may remain disconnected throughout the investigation. This distinction is important for the present study. The contribution is not the introduction of a new threat-intelligence service or a new XAI algorithm. Instead, the proposed framework integrates external threat intelligence with machine-learning predictions and SHAP-based explanations, allowing the three sources of evidence to be considered within a common malware-analysis workflow.

SOC-Oriented Malware Analysis and Research Gap

SOC requires analysts to process and correlate information from multiple security sources. From this operational perspective, malware detection accuracy alone is not sufficient to support a complete investigation. Analysts may need to determine whether an application is suspicious, understand the features responsible for the prediction, obtain contextual intelligence about the sample, and review the combined information through an operational interface. Existing research provides important components of this process. Android malware studies have investigated static analysis and machine-learning-based classification (Arp et al., 2014; Prasad et al., 2025; Santos et al., 2013; Shimpi & Pise, 2023a; Shimpi & Pise, 2023b; Shimpi & Pise, 2026; Yuan et al., 2014; Zhou & Jiang, 2012). XAI research has introduced methods for interpreting machine-learning decisions (Galli et al., 2024; Lundberg & Lee, 2017; Ribeiro et al., 2016; Rjoub et al., 2023). CTI research has established mechanisms for collecting and analysing external security intelligence (Sun et al., 2023). However, these capabilities generally address different stages or objectives of malware analysis. The specific research gap addressed by this study is, therefore, the limited integration of these capabilities into a unified Android malware analysis workflow. The suggested architecture consolidates five essential elements into a cohesive framework: static application analysis and feature extraction, machine-learning-driven

malware classification, feature-level elucidation of specific predictions, multi-source threat intelligence augmentation, and Security Operations Center (SOC)-focused visualisation of the aggregated evidence. Collectively, these elements facilitate malware identification, enhance the clarity of classification outcomes, augment the analysis with pertinent threat intelligence, and deliver the consolidated evidence in a format appropriate for security analysts.

The proposed framework addresses this gap by connecting these functions within a common analytical pipeline. The objective is not to claim that machine learning, SHAP, static analysis, or threat intelligence are individually novel. Instead, the proposed contribution lies in their integration and operational sequencing for analyst-oriented Android malware investigation.

Comparison with Existing Approaches

Table 1 summarises the functional differences between representative studies and the proposed framework.

Table 1
Functional comparison of existing malware-analysis approaches.

Approach / Study	Malware Analysis	Machine Learning	Explainability / XAI	Threat Intelligence	SOC-Oriented Integration	Primary Focus
Christodorescu and Jha (2003)	Static analysis	No	No	No	No	Malicious code pattern detection
Moser et al. (2007)	Static analysis	No	No	No	No	Limitations of static malware analysis
Bayer et al. (2009)	Behavioural analysis	Yes	No	No	No	Malware clustering
Zhou and Jiang (2012)	Static and behavioural analysis	Limited	No	No	No	Android malware characterisation
Arp et al. (2014)	Static analysis	Yes	Limited	No	No	Android malware detection
Yuan et al. (2014)	Android application analysis	Deep learning	No	No	No	Android malware classification
Hardy et al. (2016)	Malware analysis	Deep learning	No	No	No	Intelligent malware detection
Ribeiro et al. (2016)	General classification	Yes	Yes	No	No	Model-level explanation

Table 1 (*continued*)

Approach / Study	Malware Analysis	Machine Learning	Explainability / XAI	Threat Intelligence	SOC-Oriented Integration	Primary Focus
Lundberg and Lee (2017)	General ML analysis	Yes	Yes (SHAP)	No	No	Feature-level model explanation
Ronen et al. (2018)	Static PE analysis	Yes	No	No	No	Windows malware classification
Prasad et al. (2025)	Android static analysis	Yes	No	No	No	Source-code and permission-based Android malware detection
Galli et al. (2024)	Behavioural malware analysis	Yes	Yes	No	No	Explainable malware detection
Sun et al. (2023)	Cyber threat intelligence	No	No	Yes	Partially	Threat-intelligence mining and cybersecurity
Proposed framework	Static Android analysis	Yes	Yes (SHAP)	Yes, multi-source	Yes	Integrated SOC-oriented Android malware analysis

Previous studies focus on different features of malware detection, machine learning, explainability, and threat intelligence, as summarised in Table 1. This comparison was produced by the authors based on the relevant literature to highlight the functional and architectural differences of the existing approaches and the proposed framework. We present a framework that combines a machine-learning classification, feature-level SHAP explanation, and multi-source threat-intelligence enrichment in an analyst-centric workflow. This comparison shows the combination of various capabilities, rather than claiming superiority to any single malware detector. Benchmark datasets and standard assessment metrics are used to evaluate the quantitative classification performance.

Quantitative Positioning Against Existing Android Malware Studies

To further situate the proposed framework within the Android malware detection literature, quantitative results should be compared with studies that use comparable datasets and evaluation metrics. Table 2 should therefore be populated using performance values reported in the original publications.

Table 2
Quantitative comparison with representative Android malware detection studies.

Study	Dataset	Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	AUC (%)	XAI	SOC Integration
Arp et al. (2014) – DREBIN	DREBIN	Linear SVM	—	—	94.0	—	—	No	No
Alkahtani and Aldhyani (2022)	DREBIN	SVM	80.71	—	—	—	—	No	No
Alkahtani and Aldhyani (2022)	DREBIN	KNN	81.57	—	—	—	—	No	No
Aamir et al. (2024) – AMDDL model	DREBIN	CNN	99.92	98.61	99.16	98.88	—	No	No
Palma et al. (2024)	DREBIN	Best ML model	97.47	—	—	—	—	Yes	No
Zaidi et al. (2025)	DREBIN	CNN + XGBoost	98.76	—	—	—	—	No	No
Proposed framework	DREBIN	Best classifier: XGBoost	98.9	98.6	98.4	98.5	99.6	Yes (SHAP)	Yes
Proposed framework	TUANDROMD	Random Forest	99.8	99.7	99.7	99.8	99.8	Yes (SHAP)	Yes

Because the studies may differ in datasets, feature representations, preprocessing procedures, train-test splits, and evaluation protocols, direct numerical comparisons should be interpreted with caution. The primary purpose of Table 2 is to position the proposed framework against comparable Android malware detection approaches, rather than to claim superiority solely based on differences in reported accuracy. Where the same dataset and evaluation protocol are available, direct comparison is more meaningful. For studies using different datasets, the results should be presented as contextual evidence rather than as a strict ranking of algorithms.

Positioning of the Proposed Study

The literature reviewed above demonstrates three major research directions. The first focuses on improving malware detection using static analysis, machine learning, and deep learning (Arp et al., 2014; Hardy et al., 2016; Prasad et al., 2025; Santos et al., 2013; Shimpi & Pise, 2023a; Shimpi & Pise, 2023b; Shimpi & Pise, 2026; Yuan et al., 2014; Zhou & Jiang, 2012). The second focuses on improving model transparency through XAI techniques such as SHAP and LIME (Galli et al., 2024; Lundberg & Lee, 2017; Ribeiro et al., 2016; Rjoub et al., 2023). The third focuses on the collection and utilisation of cyber threat intelligence to provide contextual security information (Sun et al., 2023).

The proposed framework builds upon these established research directions by integrating them into a common analytical architecture. Its primary objective is Android malware detection, and therefore, the Android datasets constitute the principal evaluation environment. The EMBER Windows PE dataset is used only as an auxiliary cross-platform validation environment (Anderson & Roth, 2018). The Windows dataset is not treated as an Android malware dataset, and results from different operating systems are not combined into a single performance measure. This distinction is necessary because Android and Windows malware datasets differ in operating-system architecture, feature representation, application structure, and malware characteristics. Android datasets may contain permissions, API usage, intents, and application-level characteristics, whereas EMBER represents Windows Portable Executable characteristics. Therefore, the cross-platform experiments are interpreted as evidence of architectural adaptability rather than direct evidence that Android and Windows detection tasks are equivalent. Overall, the existing literature demonstrates that machine learning can provide automated malware classification, XAI can improve interpretability, and CTI can provide contextual security intelligence. The specific contribution of the proposed study is to integrate these capabilities into a single SOC-oriented Android malware analysis workflow. The framework, therefore, seeks to bridge the gap between automated classification and the practical evidence required for malware triage and investigation.

PROPOSED FRAMEWORK

The proposed framework is a modular system structure that contains five major layers of operational functionality, with each layer serving a specific function:

Frontend Layer	:	A dashboard for the storage of SOC datasets with operational capabilities for uploading files from external sources and visual representation of the dataset (Web-based SOC dashboard).
Backend Layer	:	Serves as the interface for file processing, coordination of APIs, and aggregated results.
Machine Learning Engine	:	Implements a Python-based machine learning classification engine pipeline.
Threat Intelligence Aggregation	:	Provides an interface to integrate results from external API sources.
Storage Database Layer	:	Provides an internal file hashing database system for caching and hashing the processing results of uploaded files.

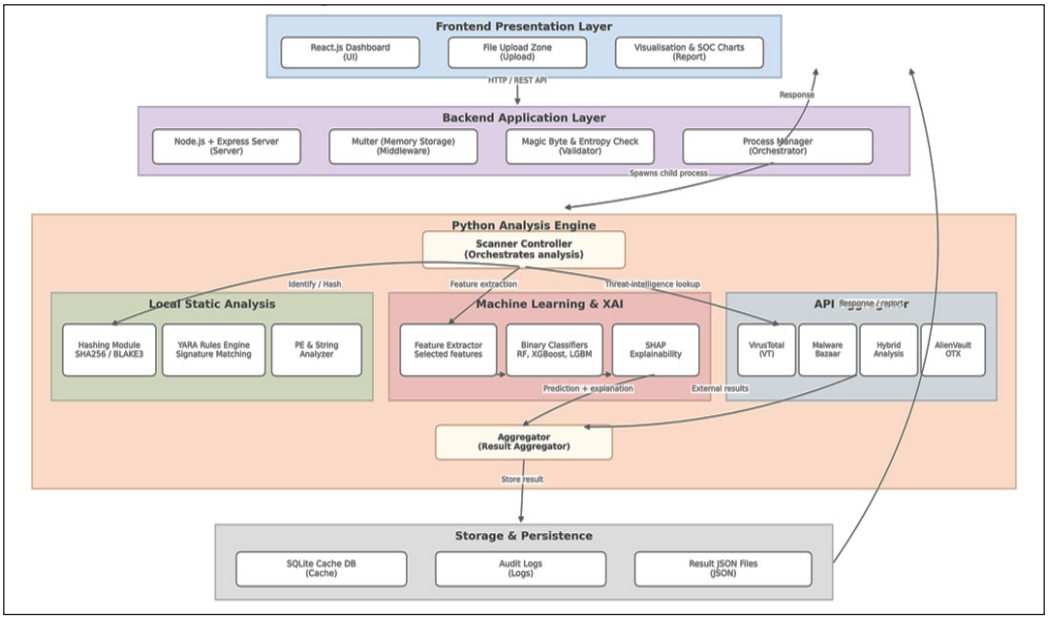

Figure 1. Proposed SOC-oriented malware analysis framework architecture

All uploaded files undergo hashing, static analysis, machine-learning inference, and threat-intelligence correlation before being aggregated into final processed results. Figure 1

depicts the operational interaction among the frontend dashboard, the backend orchestration layer, and the machine-learning Python analysis engine. Each of the three components performs either a static analysis, machine learning-based probabilistic inference, or threat intelligence correlation as part of the overall analysis pipeline before the aggregated results are output for visual representation in a SOC environment.

The novelty of the proposed framework lies primarily in its integrated operational architecture rather than in introducing a new standalone machine-learning algorithm. Existing Android malware detection studies have predominantly focused on improving classification performance, while other studies have investigated explainability or external threat intelligence as separate capabilities. In contrast, the proposed framework establishes a sequential analytical workflow that connects static malware analysis, machine-learning-based classification, feature-level SHAP explanations, multi-source threat-intelligence enrichment, and SOC-oriented visualization. This design enables an analyst to progress from automated malware prediction to an explanation of the contributing features and subsequently to contextual threat-intelligence evidence within the same workflow. Therefore, the framework is intended as an analyst-oriented decision-support architecture that complements automated classification with interpretable and contextual security information.

METHODOLOGY

System Architecture

The overall framework is designed with a modular architecture comprising a backend built in Python and a frontend built in React. The backend components include static analysis, feature preprocessing, model training, explainability, and multi-source threat intelligence enhancement. The modules are coordinated using a centralised Multiengine Scanner controller. The frontend communicates with the backend via a Node.js API (analyze.js). The API returns results only after local data processing is complete; streaming results back to the frontend will not be implemented at this time. In the early stages of integration, external API calls were embedded directly in the enrichment pipeline. If any one of these external APIs were to fail, the entire aggregation process would fail. By isolating the responses from the external APIs, we can contribute to a final threat score if at least one engine successfully processes the data. To further reduce the likelihood of receiving slow responses from any of the external APIs, server session durations have been increased, and redundant calls are minimized by introducing a caching mechanism.

Static Analysis Layer

The system is designed to perform static analysis, ensuring that files are never executed and thereby reducing operational risks during large-scale screening. The analysis includes

generating SHA-256 cryptographic hashes for file identification, applying YARA-based pattern matching to detect known suspicious patterns, and examining file structures and internal strings to identify potentially malicious characteristics. Additionally, file entropy and metadata are analysed to identify unusual patterns that may indicate malicious activity. The Shannon entropy of a given byte sequence is calculated using Equation 1, which measures the uncertainty in the byte distribution.

$$H = -\sum_{x} p\,(x) \log_2 p\,(x) \qquad [1]$$

where $p\,(x)$ represents the relative frequency of byte value \(x\) in the complete byte sequence. The calculated entropy value is then normalized to the range [0, 1] to maintain consistency in feature scaling across datasets. Additional static features are extracted to support malware analysis. These include checking for the presence of PE headers, such as the MZ header, identifying suspicious strings, and examining the byte-level structure of files. Previous studies have demonstrated that opcode sequences and other static code characteristics can be effective features for machine-learning-based malware detection (Santos et al., 2013). A binary feature is also generated to indicate the presence or absence of headers in each file. For the Android datasets, pre-extracted feature sets were used for analysis, and preprocessing was kept to a minimum. Consequently, the analysis relied primarily on the available static features, with limited scope for behavioral interpretation. Dynamic sandboxing was considered but not implemented due to hardware limitations and the need to support scalable analysis.

Feature Engineering and Preprocessing

There is a significant difference in the number of features (dimensions) between the datasets (108-2,381). To achieve consistency, numerical features were standardised using StandardScaler and fitted only to the training data, as shown in Equation 2:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma} \qquad [2]$$

Duplicate samples from each dataset were removed. Hash identifiers and metadata fields were not included to prevent data leakage into the training dataset. No dimensionality reduction was applied to the data; although PCA improved interpretability, it showed inconsistent performance across the datasets. Due to computational constraints, the data were randomly subsampled in a stratified manner for training on CPU(s) only. For example, the EMBER dataset was reduced to 50,000 records, while the MH- 100K dataset was

reduced to 10,000 records. Subsampling was done to maintain the class distribution in the data while reducing training time.

Machine Learning Models

Four distinct models were evaluated without hyperparameter optimisation to allow direct, repeatable comparison between them. A random state of 42 was used for each model.

Random Forest

The Random Forest classifier was configured with 100 decision trees, balanced class weights, and parallel processing to improve computational efficiency. These settings are represented in Equation 3.

$$N_{\text{trees}} = 100, \quad \text{class_weight} = \text{balanced}, \quad \text{n_jobs} = -1 \qquad [3]$$

Logistic Regression

The Logistic Regression classifier was configured using the LIBLINEAR solver, a regularisation parameter (C) of 1.0, a maximum of 1000 iterations, and balanced class weights to address class imbalance. The configuration settings are summarised in Table 3.

Table 3
Logistic Regression Configuration Parameters

Parameter	Value
Solver	LIBLINEAR
Regularisation parameter (\(C\))	1.0
Maximum iterations	1000
Class weights	Balanced

XGBoost

The XGBoost classifier was evaluated using the logarithmic loss (logloss) metric, while the default model parameters were retained to maintain consistency during cross-dataset modelling and avoid model-specific hyperparameter optimisation. The evaluation setting is presented in Table 4.

Table 4
XGBoost Evaluation Configuration.

Parameters	Value
Evaluation metric	Logarithmic loss (logloss)
Model parameters	Default settings

Exploratory Autoencoder

An Autoencoder was implemented using TensorFlow in a CPU-based environment as an exploratory unsupervised model for comparison with the supervised classifiers. The encoding dimension was defined as shown in Equation 4:

$$d_e = \min\left(64, \frac{d}{4}\right) \qquad [4]$$

where d denotes the input feature dimension and d_e represents the encoding dimension. The autoencoder followed a symmetric architecture, expressed as:

$$d \rightarrow 2d_e \rightarrow d_e \rightarrow 2d_e \rightarrow d$$

The hidden layers employed the Rectified Linear Unit (ReLU) activation function as shown in Equation 5:

$$f(x) = \max(0, x) \quad [5]$$

while the output layer used the sigmoid activation function as shown in Equation 6:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad [6]$$

The reconstruction error was minimised using Mean Squared Error (MSE) as shown in Equation 7:

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad [7]$$

represents the original input and $\hat{y}_i$ represents the reconstructed output. The Autoencoder was trained using the Adam optimiser with a learning rate as shown in Equation 8:

$$\eta = 0.001 \quad [8]$$

The Autoencoder was used only for exploratory analysis and comparison with the supervised classifiers and was not used as the primary malware classification model.

Model Configuration and Hyperparameter Considerations

The models were evaluated using predefined/default configurations to ensure a consistent comparison across heterogeneous datasets. Consequently, the reported results should be interpreted as comparative baseline performance rather than optimised performance for each individual classifier. No model-specific hyperparameter optimisation was performed, thereby avoiding differences in optimisation effort across classifiers and maintaining a uniform experimental protocol.

Explainability

A feature-level explanation module was developed using SHAP (Shapley Additive exPlanations) to provide interpretable explanations of model predictions. The TreeExplainer

algorithm was employed for ensemble-based models due to its computational efficiency, whereas KernelExplainer was not used for high-dimensional datasets because of the substantial computational overhead of SHAP value estimation (Lundberg & Lee, 2017).

Conduct of Experiments

All experiments were conducted using a CPU-based computing environment. The experimental pipeline was implemented in Python 3.10 on a 64-bit Linux-based environment (Ubuntu). Scikit-learn 1.3.2 was used for conventional machine-learning classifiers; XGBoost 2.0.3 was used for gradient-boosted classification; TensorFlow 2.15.0 was used for the Autoencoder implementation; and SHAP 0.44.1 was used for model explanation. The experiments were executed on an Intel(R) Xeon(R) CPU with approximately 12–13 GB of available RAM. A fixed random seed of 42 was used wherever applicable to reduce variation caused by random initialisation and data partitioning. The preprocessing pipeline, including removal of identifier-based features, scaling, feature selection where applicable, and stratified data partitioning, was applied consistently across the evaluated datasets. Training and testing data were kept separate before scaling and model evaluation to minimise the possibility of data leakage. For the EMBER dataset, the original partitions were combined and subsequently repartitioned using the same stratified evaluation procedure adopted for the other datasets. These implementation details are reported to improve the reproducibility and transparency of the experimental evaluation.

EXPERIMENTAL RESULTS

To further position the proposed framework within existing Android malware detection research, its classification performance was compared with representative studies that report results using related datasets or comparable evaluation metrics. The comparison focuses primarily on Android malware detection, as Android is the primary application domain of the proposed framework. Where studies used different datasets, preprocessing procedures, or evaluation protocols, the reported values are presented as published and are not treated as directly equivalent to the results obtained in the present study. Therefore, the comparison is intended to provide contextual positioning rather than to claim a strict one-to-one performance advantage. The classification performance of the proposed framework is compared with representative Android malware detection studies, as summarised in Table 5. The comparison considers the datasets, detection methods, reported performance metrics, explainability (XAI), and threat-intelligence or SOC integration. The proposed framework achieves an average classification accuracy of 95.9% across the evaluated datasets, while extending the detection process with SHAP-based explainability and multi-source threat-intelligence enrichment. However, the reported results should be interpreted cautiously because differences in datasets, feature representations, preprocessing, train–test splits,

and evaluation protocols may affect performance. Therefore, the comparison provides contextual positioning of the proposed framework rather than establishing a strict one-to-one performance advantage over existing methods as shown below.

Table 5
Quantitative comparison with representative Android malware detection studies

Study	Dataset	Detection Method	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)	XAI	Threat Intelligence / SOC Integration
Arp et al. (2014) – DREBIN	DREBIN	SVM-based static analysis	—	—	94.0	—	No	No
Aamir et al. (2024) – AMDDL model	DREBIN	CNN	99.92	98.61	99.16	98.88	No	No
Alkahtani and Aldhyani (2022)	DREBIN	LSTM	99.40	99.32	99.74	99.53	No	No
Alkahtani and Aldhyani (2022)	DREBIN	CNN-LSTM	97.20	97.72	97.92	97.82	No	No
D et al. (2024) – SecuDroid	DREBIN-215	Ensemble/ ML	See source	See source	See source	See source	No	No
AndroMD (Prasad et al., 2025)	KeyCount / ZeroOne / MNF	RF + DT + Bagging ensemble	Up to 99.88	—	—	—	No	Partial*
Han et al. (2025) – LEMSOFT	DREBIN	Soft-voting ensemble	99.89	99.77	99.62	99.68	No	No
Proposed Framework	SOMLAP, TUANDROMD, DREBIN, EMBER, MH-100K	RF / XGBoost + SHAP + Threat Intelligence	95.9†	—	—	—	Yes	Yes

The comparison indicates that the proposed framework achieves competitive classification performance while extending the detection pipeline with explainability and multi-source threat-intelligence enrichment. However, direct numerical comparisons should be interpreted with caution because differences in datasets, feature representations, preprocessing procedures, train-test partitions, and evaluation protocols can substantially

affect malware detection performance. The average classification performance or accuracy across datasets is 95.9%.

Threat Intelligence Integration

Using multiple threat intelligence platforms, such as VirusTotal, AlienVault OTX, MalwareBazaar, Hybrid Analysis, ANY.RUN, MetaDefender, and AbuseIPDB provide broader behavioural and reputation-based coverage than relying on a single detection engine. Fusing the outputs of these engines into an aggregate score creates a more reliable detection process by reducing reliance on any one engine and providing real-world contextual intelligence in the final decision-making stage. Each engine's verdict, with its accompanying confidence level, is converted into normalised numeric values via a weighted aggregation method, then mapped to one of four operational threat levels (critical, high, medium, and low).

Explainable AI Integration

The SHAP explanation framework helps you understand which features were important in determining a model's outcome. The explainability layer provides an explanation of the most important factors that determine the outcome of a given event. Examples of types of explanations include (but are not limited to) entropy-based indicators, patterns of abnormality (exceeding a threshold) in the strings of characters (API endpoints), patterns of abnormality (exceeding a threshold) in the use of API types, and patterns of structural anomalies (anomalies in structure) in an executable. The design for the explainability module has been completed as a component to allow for producing an individual per-sample detailed explanation output while running the system, increasing the understanding and transparency of how a model reached its conclusion, providing more trust in the model by analysts, and aiding analysts in making decisions when evaluating cybersecurity work products.

To demonstrate the practical role of the explainability layer, SHAP analysis was used to identify the features influencing the model’s predictions. Figure 2(a) presents the confusion matrix, showing the classification outcomes for benign and malicious samples. Figure 2(b) shows the ROC curve, illustrating the model’s ability to distinguish between the two classes, while Figure 2(c) presents the Precision–Recall curve, highlighting the relationship between precision and recall. Together, these visualisations provide complementary information about the model’s classification performance. SHAP explanations further help analysts understand the contribution of individual features to a prediction. These explanations support the interpretation of model outputs but do not replace expert judgement.

Table 6 presents a summary of results demonstrating stable performance across all datasets despite varying levels of complexity.

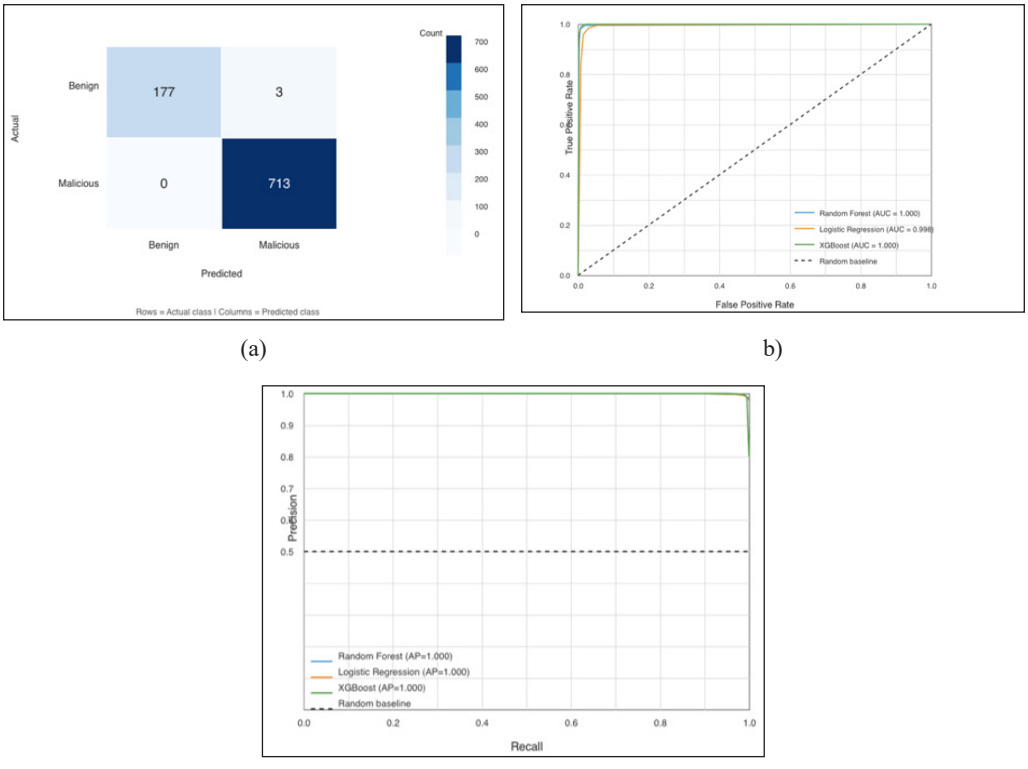

Figure 2. Representative SHAP explanation for an individual malware sample (a) Confusion Matrix, (b) ROC Curve, (c) Precision-Recall curve

Table 6
Datasets Summary

Dataset	FPR	FNR	CV F1
SOMLAP	0.4%	0.6%	0.993±0.001
TUANDROMD	0.1%	0.1%	0.994±0.004
Drebin	1.1%	1.6%	0.982±0.002
EMBER	5.7%	5.9%	0.940±0.003
MH-100K	15.1%	13.3%	0.862±0.008

DISCUSSION

The experimental findings show that the classifiers' performance varies across datasets. Instead, it largely depends on the types and characteristics of the features available in each dataset. In the present experiments, Random Forest produced the best results for SOMLAP and TUANDROMD, while XGBoost performed better on DREBIN, EMBER, and MH-100K. This variation indicates that ensemble-based models can effectively capture

nonlinear relationships among malware-related features, particularly when the underlying feature space is complex.

The results also highlight the importance of the dataset and feature representation when interpreting classification performance. The Android datasets primarily describe characteristics of mobile applications, whereas EMBER contains static features extracted from Windows Portable Executable (PE) files. These datasets, therefore, represent different malware environments and feature spaces. Consequently, the performance obtained across Android and Windows datasets should not be interpreted as a direct comparison of which operating system is easier to classify. Similar differences in malware characteristics and feature representations have also been reported in prior malware detection studies (Arp et al., 2014; Zhou & Jiang, 2012).

Another important observation is that classification performance alone may not be sufficient for an operational malware-analysis environment. A high accuracy or F1-score indicates how effectively a model performs classification, but it does not explain the reason behind an individual prediction. The use of SHAP in the proposed framework addresses this issue by identifying the features that contribute to a particular prediction (Lundberg & Lee, 2017). This information can provide analysts with additional evidence when investigating why an application has been classified as malicious or benign.

The threat-intelligence component further extends the analysis beyond the machine-learning prediction. Information obtained from external intelligence sources can provide additional context, such as malware reputation and previously observed indicators associated with an application. Cyber threat intelligence has been increasingly recognised as a useful source of contextual information for cybersecurity analysis (Sun et al., 2023). In the proposed framework, this information is used alongside the classifier output and SHAP explanation rather than being treated as an independent lookup process.

Therefore, the main contribution of the proposed framework is the integration of these complementary capabilities within a single analytical workflow. The framework enables the analysis process to move from malware classification to feature-level explanation and subsequently to contextual threat-intelligence enrichment. This design is intended to provide security analysts with more useful evidence for malware triage than a classification result alone. However, the current experimental evaluation demonstrates the framework's technical feasibility rather than its effectiveness in a production-scale SOC environment. Validation using live SOC data, real-time workloads, and analyst feedback remains an important direction for future work.

LIMITATIONS

While achieving strong classification performance, the present Application remains subject to limitations. First, the current Application is based on static analysis and therefore

would not capture the runtime behavioural patterns of Dynamic Code Loading (DCL) or Environment-Based Evasion Techniques (EBET). Second, when working with External Threat Intelligence APIs, there is a level of dependency on the External API for Access to Network Connectivity. Third, sampling for computational feasibility was performed on a variety of Large-Scale Datasets during the development of this framework, which may limit the overall scope of this framework's executability for Exhaustive Analysis. Finally, this framework was not subject to formalised benchmarking of Real-Time Performance Metrics. The Consideration and Resolution of the limitations described above will be an integral part of our immediate next steps. Although promising results have been observed, deployment in the real world could introduce various Operational Limitations. In addition, the evaluation is based primarily on publicly available benchmark datasets and may therefore not fully represent the diversity and evolving characteristics of malware encountered in operational Security Operations Centres. The current study also does not include a formal usability evaluation involving SOC analysts or deployment in a live SOC environment. Consequently, claims regarding analyst trust and operational decision support are based on the framework's functional capabilities and representative analytical outputs rather than on controlled human-subject evaluation. Future work will evaluate the framework using recent real-world malware samples, analyst-centred usability studies, large-scale threat intelligence feeds, and live or simulated SOC environments.

CONCLUSION

This study presented Neurosec Malware Scanner, an integrated framework for Android malware analysis that combines static machine-learning classification, SHAP-based explainability, multi-source threat-intelligence enrichment, and SOC-oriented result presentation. The framework was evaluated on multiple publicly available benchmark datasets, with the results demonstrating that supervised machine learning models can achieve strong classification performance across heterogeneous malware feature spaces. The principal contribution of the work is the integration of complementary analytical capabilities into a single malware-triage workflow. Machine-learning classification provides automated detection, SHAP provides feature-level interpretation, and external threat intelligence adds contextual information to assist analysts during investigations. The inclusion of caching, structured logging, hashing, and dashboard visualisation further supports repeatable analysis. However, the present study should be considered a benchmark-validated research prototype rather than a fully deployed SOC solution. The evaluation does not include a live SOC environment, a formal analyst usability study, or extensive validation using continuously evolving real-world malware. In addition, the manually specified threat-intelligence weighting scheme and fixed baseline model configurations provide opportunities for further optimisation.

Future work will therefore focus on learned threat intelligence weighting, temporal and cross-dataset validation, recent real-world Android malware samples, adversarial and obfuscation robustness, automated hyperparameter optimisation, and controlled SOC-based evaluation. These extensions will provide stronger evidence regarding the framework's generalizability, operational efficiency, and suitability for real-world cybersecurity environments.

ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to their institution for providing the necessary research facilities and academic support. Special thanks are extended to the research guide and colleagues for their suggestions and encouragement, which contributed to the successful completion of this work.

LIST OF ABBREVIATION

SOC	Security Operations Centre
AI	Artificial Intelligence
ML	Machine Learning
DCL	Dynamic Code Loading
EBET	Environment-Based Evasion Techniques
DCL	Dynamic Code Loading
API	Application Programming Interface (API)
CPU	Central Processing Unit (CPU)
PCA	Principal Component Analysis (PCA)
ROC	Receiver Operating Characteristic (ROC)
AUC	Area Under the Curve (AUC)
FPR	False Positive Rate (FPR)
FNR	False Negative Rate (FNR)
ReLU	Rectified Linear Unit (ReLU)
MSE	Mean Squared Error (MSE)

REFERENCES

Aamir, M., Iqbal, M. W., Nosheen, M., Ashraf, M. U., Shaf, A., Almarhabi, K. A., Alghamdi, A. M., & Bahaddad, A. A. (2024). AMDDLmodel: Android smartphones malware detection using deep learning model. *PLOS ONE*, *19*(1), Article e0296722. https://doi.org/10.1371/journal.pone.0296722

Alkahtani, H., & Aldhyani, T. H. H. (2022). Artificial intelligence algorithms for malware detection in Android-operated mobile devices. *Sensors*, *22*(6), Article 2268. https://doi.org/10.3390/s22062268

Anderson, H. S., & Roth, P. (2018). *EMBER: An open dataset for training static PE malware machine learning models* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1804.04637

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 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2014.23247

Bayer, U., Milani Comparetti, P., Hlauschek, C., Kruegel, C., & Kirda, E. (2009). Scalable, behavior-based malware clustering. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society. https://www.ndss-symposium.org/ndss2009/scalable-behavior-based-malware-clustering/

Christodorescu, M., & Jha, S. (2003). Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium* (pp. 169-186). USENIX Association. https://www.usenix.org/events/sec03/tech/christodorescu.html

Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys*, *44*(2), Article 6. https://doi.org/10.1145/2089125.2089126

Galli, A., La Gatta, V., Moscato, V., Postiglione, M., & Sperlì, G. (2024). Explainability in AI-based behavioral malware detection systems. *Computers & Security*, *141*, Article 103842. https://doi.org/10.1016/j.cose.2024.103842

Han, Q., Shi, Z., Li, Y., & Zhang, T. (2025). LEMSOFT: Leveraging extraction method and soft voting for Android malware detection. *Mathematics*, *13*(21), Article 3569. https://doi.org/10.3390/math13213569

Hardy, W., Chen, L., Hou, S., Ye, Y., & Li, X. (2016). DL4MD: A deep learning framework for intelligent malware detection. In *Proceedings of the 2016 International Conference on Data Mining* (pp. 61–67). CSREA Press. https://www.worldcomp-proceedings.com/proc/proc2016/DMIN16_Final_Edition/DMIN16_Papers.pdf

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30, pp. 4765-4774). Curran Associates. https://papers.nips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

Moser, A., Kruegel, C., & Kirda, E. (2007). Limits of static analysis for malware detection. In *23rd Annual Computer Security Applications Conference (ACSAC 2007)* (pp. 421-430). IEEE. https://doi.org/10.1109/ACSAC.2007.21

Palma, C., Ferreira, A., & Figueiredo, M. (2024). Explainable machine learning for malware detection on Android applications. *Information*, *15*(1), Article 25. https://doi.org/10.3390/info15010025

Prasad, A., Chandra, S., Alenazy, W. M., Ali, G., Shah, S., & ElAffendi, M. (2025). AndroMD: An Android malware detection framework based on source code analysis and permission scanning. *Results in Engineering*, *28*, Article 107050. https://doi.org/10.1016/j.rineng.2025.107050

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge*

Discovery and Data Mining (pp. 1135-1144). Association for Computing Machinery. https://doi.org/10.1145/2939672.2939778

Rjoub, G., Bentahar, J., Abdel Wahab, O., Mizouni, R., Song, A., Cohen, R., Otrok, H., & Mourad, A. (2023). A survey on explainable artificial intelligence for cybersecurity. *IEEE Transactions on Network and Service Management*, *20*(4), 5115-5140. https://doi.org/10.1109/TNSM.2023.3282740

Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., & Ahmadi, M. (2018). *Microsoft malware classification challenge* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1802.10135

Santos, I., Brezo, F., Ugarte-Pedrero, X., & Bringas, P. G. (2013). Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences*, *231*, 64-82. https://doi.org/10.1016/j.ins.2011.08.020

Shimpi, P. M., & Pise, N. N. (2023a). An empirical study of efficient malware detection analysis on Android mobile phones using machine learning. In *2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)* (pp. 1520-1526). IEEE. https://doi.org/10.1109/ICSCDS56580.2023.10104900

Shimpi, P. M., & Pise, N. N. (2023b). Android malware detection model using the adaptive swarm optimization-based neural network. In *2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)* (pp. 1641-1648). IEEE. https://doi.org/10.1109/ICSSIT55814.2023.10061042

Shimpi, P. M., & Pise, N. N. (2026). Android malware detection using an optimized and explainable attentional convolutional LSTM model. *Journal of Intelligent Decision Making and Information Science*, *3*(2S), 212-233. https://doi.org/10.59543/jidmis.v3.639

Sun, N., Ding, M., Jiang, J., Xu, W., Mo, X., Tai, Y., & Zhang, J. (2023). Cyber threat intelligence mining for proactive cybersecurity defense: A survey and new perspectives. *IEEE Communications Surveys & Tutorials*, *25*(3), 1748-1774. https://doi.org/10.1109/COMST.2023.3273282

Vanusha, D., Singh, S., Jha, A. K., & Robinsha S, D. (2024). SecuDroid: Android malware detection using ML classifier on static features. In 2024 2nd International Conference on Networking and Communications (ICNWC) (pp. 1-9). IEEE. https://doi.org/10.1109/ICNWC60771.2024.10537417

You, I., & Yim, K. (2010). Malware obfuscation techniques: A brief survey. In *2010 International Conference on Broadband, Wireless Computing, Communication and Applications* (pp. 297-300). IEEE. https://doi.org/10.1109/BWCCA.2010.85

Yuan, Z., Lu, Y., Wang, Z., & Xue, Y. (2014). Droid-Sec: Deep learning in Android malware detection. *ACM SIGCOMM Computer Communication Review*, *44*(4), 371-372. https://doi.org/10.1145/2740070.2631434

Zaidi, A. R., Abbas, T., Daud, A., Alghushairy, O., Dawood, H., & Sarwar, N. (2025). Enhancing Android malware detection with XGBoost and convolutional neural networks. *Computers, Materials & Continua*, *84*(2), 3281-3304. https://doi.org/10.32604/cmc.2025.063646

Zhou, Y., & Jiang, X. (2012). Dissecting Android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy* (pp. 95-109). IEEE. https://doi.org/10.1109/SP.2012.16