Journal of IoT in Social, Mobile, Analytics, and Cloud (ISSN: 2582-1369)
www.irojournals.com/iroismac/

XEnDroid: Explainable Stacking Ensemble Learning for Dynamic Android Malware Detection

Muralidharan B.[1], Vignesh R.[2], Surjith K.[3], Yagavarman S.[4]

[1]Assistant professor, [2,3,4]Student, Department of Computer Science and Engineering, NPR College of Engineering and Technology, Natham, Dindigul, India.

E-mail: [1]muralidharanb@nprcolleges.org, [2]vigneshr920822104117@nprcolleges.org, [3]surjithk920822104111@nprcolleges.org, [4]yagavarmans920822104124@nprcolleges.org

Abstract

Android malware attributes to the increasing usage of code obfuscation, polymorphism, and dynamic execution. In this study, we propose XEnDroid (Xplainable Ensemble Droid), an advanced stacking ensemble approach for detecting malware on Android phones by combining random forest, convolution neural network, and transformer models under a logistic regression model. Dynamic behavioral features such as API, system call, and network activities are used for extracting features that are later reduced using PCA and processed through SMOTE. Model transparency is ensured by incorporating SHAP and LIME, which helps in understanding both global feature significance and prediction explanation locally. An experimental analysis using the CICMalDroid2020 dataset proves the effectiveness of the proposed method, as it results in 94.6% accuracy, 94.1% precision, 94.2% recall, 94.1% F1-score, and 0.986 AUC score.

Keywords: Android Malware Detection, Stacking Ensemble, Explainable Artificial Intelligence (XAI), Random Forest (RF), Convolutional Neural Network (CNN), SHAP, LIME.

1. Introduction

Android is established as the most popular mobile operating system and therefore it is a profitable target for malware attacks that have grown more advanced over time. The Android malware makes use of techniques such as code obfuscation, code loading, and polymorphism to bypass traditional signature-based antivirus programs, making it difficult to detect new and

Journal of IoT in Social, Mobile, Analytics, and Cloud, September 2026, Volume 8, Issue 3, Pages 308-323
DOI: https://doi.org/10.36548/jismac.2026.3.008
version of article received from **International Conference on Smart Innovations in Engineering Technology (ICSIET 2026)**
Received: 13.06.2026, received in revised form: 18.07.2026, accepted: 06.08.2026, published: 18.08.2026

emerging threats. This implies that intelligent malware detection algorithms that learn behavior patterns are vital.

Machine Learning and Deep Learning techniques have shown significant potential in detecting malicious Android apps through runtime behavioral analysis such as API calls, system calls, and networking operations. Nevertheless, the current techniques generally suffer from several drawbacks, including high-dimensional feature space, class imbalance, inadequate use of complementary learning algorithms, and lack of decision interpretability. Although deep neural networks show impressive detection capability, the fact that they are black boxes diminishes the trust in security-related applications.

To address these issues, this research work presents XEnDroid, an improved stacking ensemble architecture which uses Random Forest, CNN, and Transformer algorithms in combination with logistic regression-based meta-learning. The architecture utilizes PCA in order to reduce the number of features, SMOTE for dealing with class imbalance, and SHAP & LIME for explaining the model prediction results on a global and local level. Thus, XEnDroid incorporates heterogeneous machine learning models along with explainable AI techniques.

The key contributions from the current study are enumerated below:

- Proposed design of an ensemble of heterogenous classifier using Random Forest, CNN, and Transformers classifiers for detection of malware in Android devices.
- Use of PCA and SMOTE techniques to enhance the efficiency of the learning process and minimize the problem of class imbalance.
- Application of SHAP and LIME techniques for global and local interpretability of the malware predictions.
- Conducted experiments that prove the efficacy of the proposed approach relative to standalone classifiers and traditional ensembles.

2. Related Works

Malware detection has evolved from conventional signature matching toward behavior-driven and learning-based approaches capable of identifying previously unseen and obfuscated threats. Previous works have grouped the current malware analysis methods into

three broad categories: static, dynamic, and hybrid approaches, where static approaches are said to be computationally efficient but prone to malware obfuscations such as packing, code obfuscation, and polymorphic code [1]. The combination of static and dynamic characteristics using machine learning is thus considered to be an area of interest since the observation at run time might reveal the malicious behavior of the application not known through its binary [2]. Previous studies in behavior-oriented methods proved that the machine learning techniques could automatically recognize malware from behavior-based information [3].

In addition to that, deep learning has enhanced the process of malware analysis in the context of Android by understanding complicated dependencies between the features with little reliance on manual feature engineering and decision rules. DL-Droid was able to illustrate the potential of deep learning architecture by utilizing the behavior information of the actual Android devices and emphasizing the need for realism in malware analysis [4]. Moreover, hybrid approach to the problem of malware classification has also been addressed by SAMADroid where different levels of analytics are considered [5]. In a more generalized way, MalScan has utilized the graph-based approach of social network centrality measures for performing efficient malware scanning of the entire market. It shows that dependency relationships between application components can be useful for performing the malware classification task [6]. In that line, HYDRA has been proposed by utilizing multimodal deep learning approach to malware detection task [7].

The sequential behavioral modeling proves to be applicable as the malicious behaviors typically come in the form of a sequence of API calls rather than discrete instances. Sequential API call alignment and visualization were reported to enhance malware detection and classification as they help preserve the dependencies between the execution events [8]. More advanced approaches incorporate gated recurrent networks and generative adversarial learning to develop API-call sequences [9]. While sequence modeling architectures prove to be efficient in capturing the temporal dependencies, they can be impacted by the unevenly balanced classes. The research on supervised class imbalance learning revealed the existence of biased decision boundaries, lack of minority class recognition, and inadequate evaluation methods [10].

Another important issue is interpretability. Since most successful deep models give minimal justification of security decisions made. Explainable artificial intelligence in cybersecurity stresses the importance of feature attribution, analysis of decision locally, and

providing analyst explanations as a key component for trustworthiness of the model [11]. At the same time, network-based characterization of Android malware showed that network traffic behaviors could help to detect malicious actions and provide additional features [12]. The aforementioned developments make possible the introduction of the proposed framework XEnDroid based on heterogeneous Random Forest, CNN, and Transformer models, stacking, PCA and SMOTE pre-processing, and SHAP and LIME explainability. Hence, this approach solves several drawbacks found in the existing literature by analyzing behavioral diversity, sequence dependencies, ensembles, class imbalance, and explainability in one Android malware detection model.

3. Proposed Work

The proposed XEnDroid architecture for Android malware detection integrates dynamic behavior analysis, dimensional reduction, class imbalance learning, heterogeneous base classifiers, stacking technique, interpretable AI, and LLM-powered mitigation.

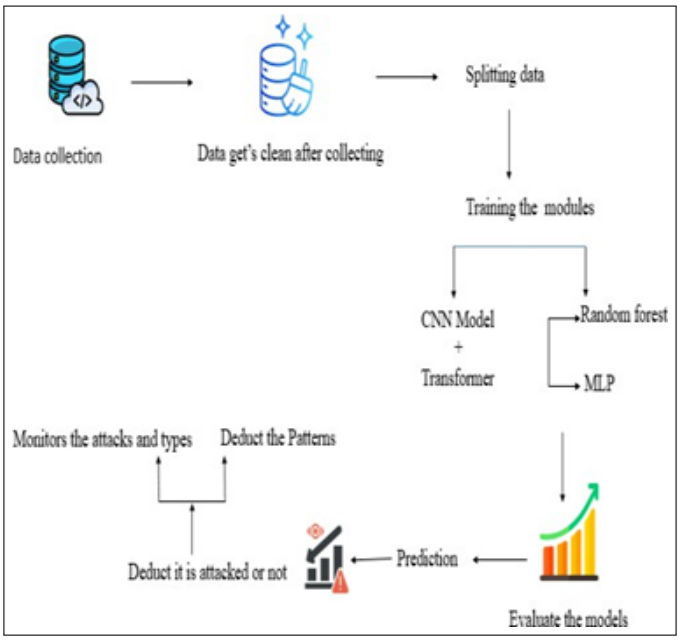

Figure 1. Overall architecture of XEnDroid

As shown in Figure 1, an Android APK is initially analyzed with respect to behavioral data collection and feature generation. The resulting representations are supplied to Random Forest, CNN, and Transformer base models. Their prediction probabilities are subsequently combined by a logistic-regression meta-learner. The final decision is interpreted using SHAP

and LIME, while the remediation module converts the detected threat characteristics into human-readable response recommendations.

The experimental setup makes use of the CICMalDroid2020 dataset, which has been created by the Canadian Institute for Cybersecurity of the University of New Brunswick [13]. In total, the initial dataset consists of 17,341 Android apps collected from various sources and classified into five categories: Adware, Banking malware, SMS malware, Riskware, and Benign apps. Dynamic analysis was conducted via the CopperDroid platform; out of the 13,077 applications that have executed properly, only 11,598 samples were left for further analysis after removing non-parsable analysis results.

Table 1. Malware category distribution

Category	**Samples**	**Proportion**
Adware	1,253	10.80%
Banking malware	2,100	18.10%
SMS malware	3,904	33.60%
Riskware	2,546	22.00%
Benign	1,795	15.50%

For the binary detection task considered in this study, the four non-benign categories are grouped into a single malware class, yielding 9,803 malicious samples and 1,795 benign samples. The dataset contains heterogeneous views that can be used for behavioral analysis of malware, namely, a 470-dimensional representation of behaviors based on system calls, Binder communication, and compound behaviors; a 139-dimensional system call representation; and a 50,621-dimensional static view consisting of permissions, intents, services, receivers, packages, sensitive APIs, and other application properties.

To address the resulting class imbalance, SMOTE is employed to the training partition, while PCA is fitted only on the training data and subsequently used to transform validation and test samples. This procedure prevents from information leakage and ensures that all performance measures to be calculated on the unseen test set.

For API-call analysis, an application is represented by the ordered sequence

$$S = \{s_1, s_2, \ldots, s_n\} \tag{1}$$

where $s_i \in A$ and A denotes the vocabulary of 1,284 unique API functions. Preserving event order is important because malicious behavior may be characterized not only by the presence of individual calls but also by their local and long-range execution dependencies.

System-call activity is aggregated over $500-ms$ observation windows. For m unique system calls, the histogram component corresponding to call j is computed as

$$H_j = \sum_{t=1}^{T} I\left(\text{syscall}_t = j\right), \qquad j = 1, \ldots, m \tag{2}$$

$I(\cdot)$ denotes the indicator function and T is the number of observed systems call in the window. This representation captures the occurrence of low-level actions while maintaining temporal locality through windowing. The behavior representations generated are then used along with network activity features to constitute the input domain of XEnDroid.

The combined behavioral feature space exceeds 2,000 dimensions. To reduce redundancy and computational cost, Principal Component Analysis (PCA) is applied before model training. For a centered data matrix $X \in \mathbb{R}^{n \times p}$, the covariance matrix is computed as

$$C = \frac{1}{n-1} X^T X \tag{3}$$

If λ_k denotes the eigenvalue associated with the k-th principal component, its explained-variance ratio is

$$EV_k = \frac{\lambda_k}{\sum_{j=1}^{p} \lambda_j} \tag{4}$$

The smallest number d of principal components is selected such that

$$\frac{\sum_{j=1}^{d} \lambda_j}{\sum_{j=1}^{p} \lambda_j} \geq 0.95 \tag{5}$$

Due to the unequal class distributions of the dataset, XEnDroid applies the Synthetic Minority Oversampling Technique (SMOTE) to the training data. Given a minority-class sample x_i and one of its nearest neighbors x_{nn}, a synthetic sample is generated as

$$x_{\text{new}} = x_i + \delta\left(x_{\text{nn}} - x_i\right), \qquad \delta \sim U(0,1) \tag{6}$$

The first base learner is a Random Forest designed primarily for the reduced tabular behavioral representation. For an ensemble of T decision trees, the predicted class is determined by

$$\hat{y}_{RF} = \text{mode}\{h_1(x), h_2(x), \ldots, h_T(x)\}. \tag{7}$$

Tree construction uses Gini impurity,

$$\text{Gini}(t) = 1 - \sum_{k=1}^{K} p_k^2 \tag{8}$$

where p_k is the proportion of class k at node t. The configuration reported in the manuscript uses 200 trees with a maximum depth of 30.

The second base learner is a CNN designed to capture local motifs within API-call sequences. The architecture shown in Fig. 2 contains three parallel convolutional branches with kernel sizes 3, 5, and 7.

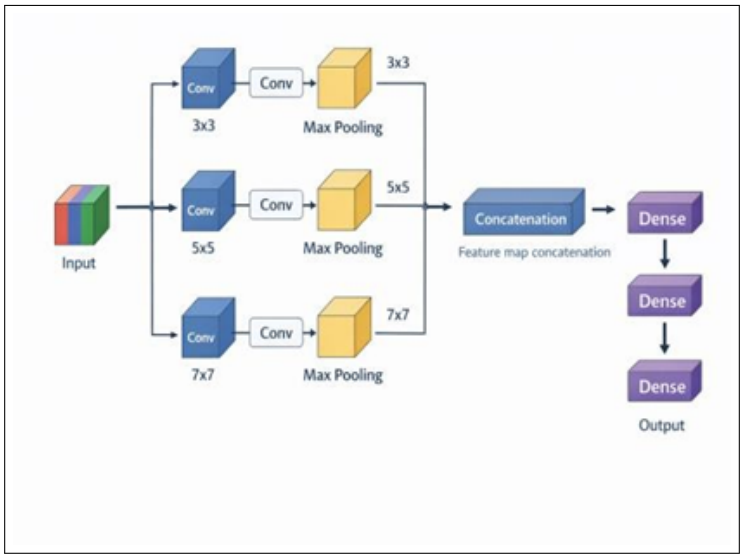

Figure 2. CNN architecture with three parallel convolutional branches

This multi-scale design enables the model to identify short-, medium-, and longer-local behavioral patterns.

For a convolution filter $w \in \mathbb{R}^{k \times 128}$, the feature response at position i is

$$c_i = \text{ReLU}\left(w \cdot E[i : i+k-1] + b\right) \tag{9}$$

where E denotes the embedded API-call sequence. Global max pooling produces

$$\hat{c} = \max_i c_i \tag{10}$$

after which the pooled branch representations are concatenated and passed through dense layers. The final CNN probability is expressed as

$$\hat{y}_{\text{CNN}} = \sigma\left(W_2\,\text{ReLU}\left(W_1 f_{\text{CNN}} + b_1\right) + b_2\right) \tag{11}$$

The architecture therefore complements RF by learning discriminative local sequential patterns directly from behavioural traces.

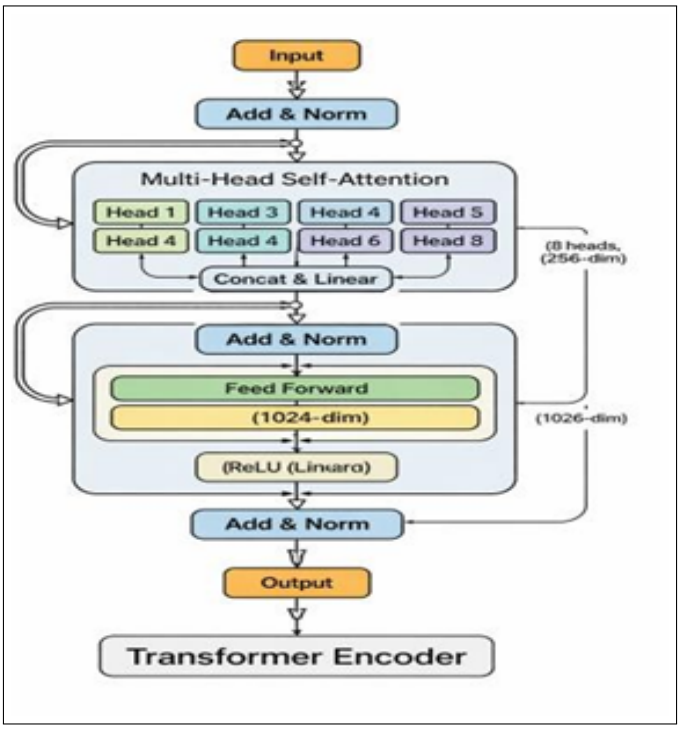

Figure 3. Transformer encoder architecture

The third base learner models' long-range dependencies in behavioural sequences. As shown in Fig. 3, the Transformer encoder uses multi-head self-attention, feed-forward processing, and residual connections.

Positional information is incorporated through sinusoidal encodings:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \tag{12}$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \tag{13}$$

Scaled dot-product attention is computed as

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{14}$$

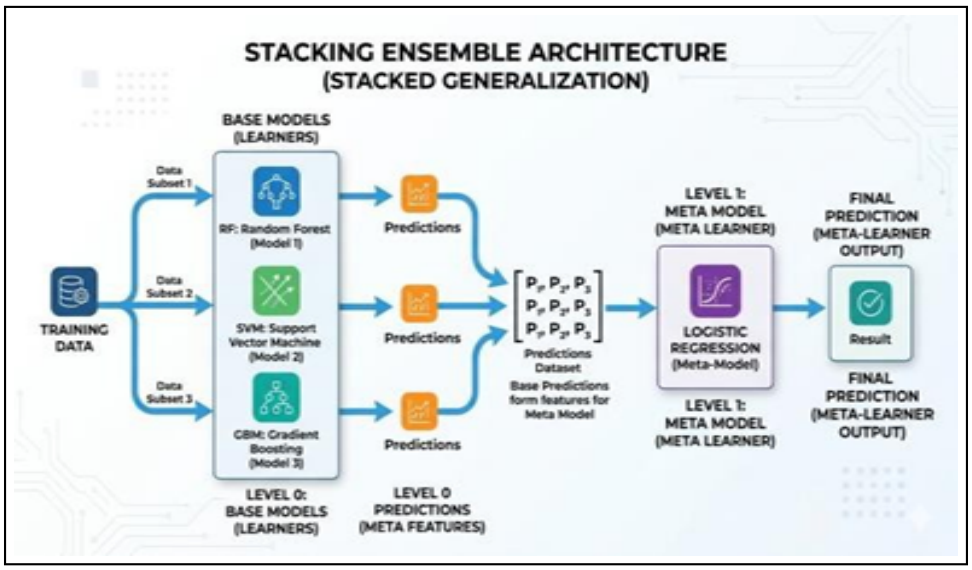

Figure 4. Stacking ensemble framework

The central component of XEnDroid is the stacking mechanism shown in Figure 4. Apart from assigning fixed weights to the base classifiers, the proposed approach learns how their outputs should be combined. Let

$$M = \{M_1, M_2, M_3\} \tag{15}$$

represent RF, CNN, and Transformer, respectively. For an input x, each model produces a malware probability

$$p_j(x) = M_j(x) \in [0,1], \qquad j = 1,2,3 \tag{16}$$

These probabilities are concatenated into the meta-feature vector

$$p(x) = [p_1(x), p_2(x), p_3(x)] \in \mathbb{R}^3 \tag{17}$$

A logistic-regression meta-learner then generates the final prediction:

$$\hat{y}_{stack} = L(z(x)) \stackrel{\downarrow}{=} \sigma(w^T z(x) + b) \tag{18}$$

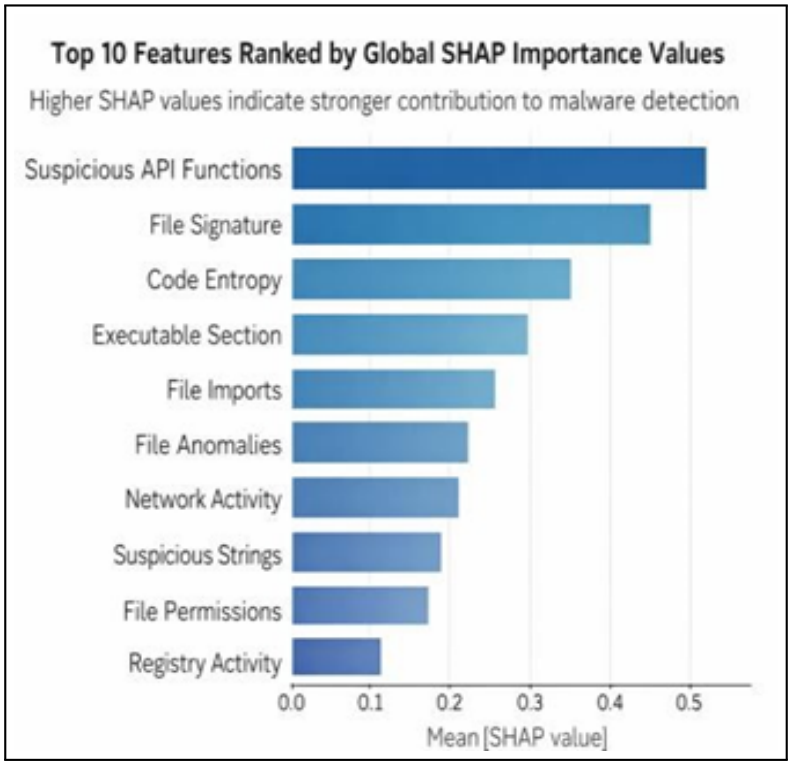

Figure 5. Top 10 features ranked by global SHAP importance values

The XEnDroid framework uses SHAP and LIME for explanation post-classification. The global SHAP analysis in Figure 5 provides the ranking of the most impactful features. It is evident from the figure that the suspicious API functions contribute most to the mean absolute SHAP value. The remaining features ranked include file signature, code entropy, executable section, file import, file anomaly, network, suspicious string, file permission, and registry. The ranking is reflective of the framework's objective to use both behavioral and structural features instead of one feature class alone.

For feature j, the SHAP value is defined as

$$\phi_j = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{j\}} \left(x_{S \cup \{j\}} \right) - f_S(x_S) \right] \tag{19}$$

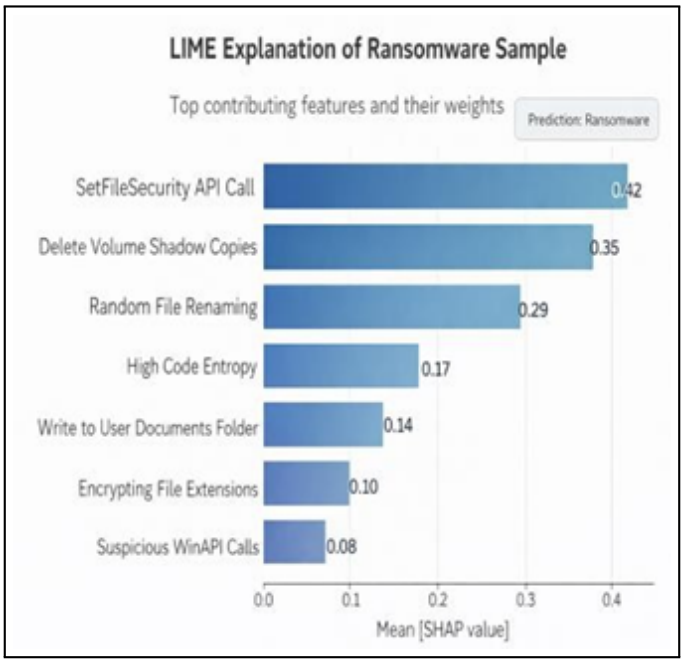

Figure 6. Example LIME explanation for a detected ransomware sample

LIME complements the global explanation by fitting a sparse local surrogate around a specific prediction:

$$\xi(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g) \tag{20}$$

where f is the original classifier, g is the interpretable local model, π_x defines locality around x, and $\Omega(g)$ penalizes model complexity.

The final stage converts detection data into structured advice for remediation. The remediation engine will receive the type of malware detected, the level of confidence, and the key factors explaining the detection. All these factors will be used to generate a structured prompt asking for four types of responses - immediate actions, system hardening, user recommendations, and prevention measures.

4. Results and Discussion

This section evaluates XEnDroid from complementary perspectives: base-model performance, ensemble effectiveness, comparison with existing systems, explainability, class balancing, base-model contribution, statistical significance, and computational efficiency.

Table 2. Performance comparison of different models

Model	Accuracy	Precision	Recall	F1-Score	AUC
Random Forest	0.891	0.878	0.885	0.881	0.945
CNN	0.912	0.901	0.908	0.904	0.962
Transformer	0.924	0.915	0.919	0.917	0.971
LSTM	0.898	0.887	0.892	0.889	0.953

Table 2 shows a comparison between RF, CNN, Transformer, and LSTM. This evidence further reinforces the architectural rationale for heterogeneous learning. RF is slightly inaccurate compared to the sequence models but adds a unique mechanism of decision making based on tabular data. CNN increases accuracy by modeling the local patterns of API calls while the Transformer achieves the best accuracy among the four models.

Table 3. Performance comparison of ensemble methods

Ensemble Method	Accuracy	Precision	Recall	F1-Score	AUC
Hard Voting	0.931	0.924	0.926	0.925	0.977
Soft Voting	0.935	0.928	0.930	0.929	0.979
Weighted Average	0.938	0.931	0.933	0.932	0.981
Stacking (Ours)	0.946	0.941	0.942	0.941	0.986

The comparisons of the ensembles in Table 3 show a gradual improvement from fixed combinations to learned stacking. The suggested stacking ensemble has proven to be the most effective one in terms of all metrics used: 0.946 accuracy, 0.941 precision, 0.942 recall, 0.941 F1-score, and 0.986 AUC. In comparison with the best-performing Transformer model, the stacking ensemble shows better accuracy and F1-score values - 0.946 versus 0.924 accuracy and 0.941 versus 0.917 F1-score. It is an improvement of 2.2 percentage points compared to the best performing single model. Compared to the RF, the improvement is 5.5 percentage points.

Table 4. PCA variance threshold analysis

Variance Threshold	Features	Accuracy	F1-Score	Train Time
90%	42	0.931	0.926	18.5 min
95%	67	0.946	0.941	22.3 min
99%	128	0.948	0.943	31.7 min
100% (No PCA)	2,156	0.949	0.944	58.2 min

The PCA analysis shown in Table 4 demonstrates that dimensionality reduction helps in reducing computational expense without compromising the performance of predictions. The PCA result with a 95% variance threshold cuts down the number of features from 2,156 to just 67 while maintaining an accuracy of 94.6%. Even though using all the features gives 94.9% accuracy, it increases the training time by over two times. Hence, PCA with a 95% variance threshold is the best approach to adopt.

Table 5. Comparison of data balancing methods

Balancing Method	Accuracy	Precision	Recall	F1-Score
No Balancing	0.865	0.812	0.758	0.784
Random Oversampling	0.901	0.885	0.878	0.881
SMOTE	0.918	0.905	0.901	0.903
ADASYN	0.915	0.901	0.898	0.899
Proposed (PCA + SMOTE + Stacking)	0.946	0.941	0.942	0.941

Table 5 illustrates the importance of handling class imbalance in Android malware datasets. Unbalanced dataset leads to classifier bias on majority class and consequently poor recall and worst F1-score value. On the other hand, random oversampling technique gives better results, but at the same time, increases overfitting probability because of duplicates. The ADASYN approach shows more effective class representation by creation of synthetic samples. The suggested approach based on SMOTE, feature reduction using PCA and stacking ensemble gives best performance results.

Table 6. Performance comparison of base models in stacking

Base Models in Stack	Accuracy	F1-Score	AUC
RF only	0.891	0.881	0.945
CNN only	0.912	0.904	0.962
Transformer only	0.924	0.917	0.971
RF + CNN	0.931	0.925	0.976
RF + Transformer	0.935	0.929	0.978
CNN + Transformer	0.938	0.933	0.980
RF + CNN + Transformer	0.946	0.941	0.986

Table 6 presents an ablation study of the proposed ensemble that shows the role of each of the components separately. Among the individual classifiers, the Transformer has the highest accuracy (92.4%), suggesting its capability to recognize long-term behavioral relations. The combination of Random Forest with either CNN or Transformer provides better performance since these classifiers have learned different types of feature representation. The proposed stacking approach that uses Random Forest, CNN, and Transformer altogether has provided the best performance with the accuracy of 94.6% and the AUC of 0.986.

Table 7. Computational resource and inference performance comparison

Model	Train Time	Inference	Model Size	Memory
Random Forest	0.25 hrs	2.1 ms	245 MB	1.2 GB
CNN	2.5 hrs	8.7 ms	28.5 MB	1.8 GB
Transformer	4.2 hrs	15.3 ms	52.3 MB	2.5 GB
XEnDroid	5.8 hrs	18.5 ms	325.8 MB	3.5 GB

Table 7 compares the computational requirements of the evaluated models. The Random Forest has the shortest training and testing time. This model is suitable for implementation on lightweight systems but has low accuracy of detections. Deep learning models need more time to train than others, but their predictions are more accurate. The suggested XEnDroid architecture requires the highest amount of computation as it runs three base models and meta-classifier, yet, the inference time of 18.5 ms is enough for real-time detection of malware.

Overall, the experiment shows that the proposed XEnDroid is superior to individual ML models, conventional ensemble techniques, and state-of-the-art frameworks designed for Android malware detection. The suggested heterogeneous stacking technique efficiently merges the distinctive advantages of the Random Forest, CNN, and Transformer approaches, while PCA and SMOTE increase computational efficiency and help achieve a class balance, correspondingly. Besides, the combination of SHAP and LIME makes the developed framework more interpretable due to providing global feature importance and local explanation for predictions. Thus, the proposed technique provides an appropriate trade-off between accuracy, interpretability, and computational efficiency, which makes it a suitable framework for detecting Android malware.

5. Conclusion

XEnDroid framework presented in this study is an explainable stacking ensemble framework developed to improve the detection of Android malware through the integration of the complementary aspects of machine learning and deep learning classifiers. The proposed framework utilizes behavioral feature analysis, dimensional reduction, imbalance handling, ensemble learning, and explainable artificial intelligence to provide reliable and interpretable classification of malware. Instead of the use of single models which are traditionally used, the proposed framework provides the use of diverse feature representations in order to improve robustness of the model to the changing behaviors of malware attacks, as well as provide explanations of decision-making using SHAP and LIME. The experiments show that the integration of complementary classifiers with explainable techniques enhance the model effectiveness, and interpretability. The proposed framework will be useful to cybersecurity researchers and security analysts for explaining the decision-making processes and increasing the trust in automated malware detection system.

References

[1] Aslan, Ömer Aslan, and Refik Samet. "A Comprehensive Review on Malware Detection Approaches." IEEE Access 2020, vol. 8, 6249-6271.

[2] Ijaz, Muhammad, Muhammad Hanif Durad, and Maliha Ismail. "Static and Dynamic Malware Analysis using Machine Learning." In 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST), IEEE, 2019, 687-691.

[3] Rieck, Konrad, Philipp Trinius, Carsten Willems, and Thorsten Holz. "Automatic Analysis of Malware Behavior using Machine Learning." Journal of Computer Security 2011, vol. 19, no. 4, 639-668.

[4] Alzaylaee, Mohammed K., Suleiman Y. Yerima, and Sakir Sezer. "DL-Droid: Deep Learning based Android Malware Detection using Real Devices." Computers & Security 2020, vol. 89, 101663.

[5] Arshad, Saba, Munam A. Shah, Abdul Wahid, Amjad Mehmood, Houbing Song, and Hongnian Yu. "SAMADroid: a novel 3-level hybrid Malware Detection Model for android operating system." IEEE Access 2018, vol. 6, 4321-4339.

[6] Wu, Yueming, Xiaodi Li, Deqing Zou, Wei Yang, Xin Zhang, and Hai Jin. "Malscan: Fast Market-Wide Mobile Malware Scanning by Social-Network Centrality Analysis." 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2019, 139-150.

[7] Gibert, Daniel, Carles Mateu, and Jordi Planes. "HYDRA: A Multimodal Deep Learning Framework for Malware Classification." Computers & Security 2020, vol. 95, 101873.

[8] Kim, Hyunjoo, Jonghyun Kim, Youngsoo Kim, Ikkyun Kim, Kuinam J. Kim, and Hyuncheol Kim. "Improvement of Malware Detection and Classification using API Call Sequence Alignment and Visualization." Cluster Computing 2019, vol. 22, no. Suppl 1, 921-929.

[9] Owoh, Nsikak, John Adejoh, Salaheddin Hosseinzadeh, Moses Ashawa, Jude Osamor, and Ayyaz Qureshi. "Malware Detection based on API Call Sequence Analysis: A gated Recurrent unit–Generative Adversarial Network Model Approach." Future Internet 2024, vol. 16, no. 10, 369.

[10] Das, Swagatam, Sankha Subhra Mullick, and Ivan Zelinka. "On supervised class-imbalanced learning: An Updated Perspective and Some Key Challenges." IEEE Transactions on Artificial Intelligence 2022, vol. 3, no. 6, 973-993.

[11] Zhang, Zhibo, Hussam Al Hamadi, Ernesto Damiani, Chan Yeob Yeun, and Fatma Taher. "Explainable Artificial Intelligence Applications in Cyber Security: State-of-the-art in research." IEEE Access 2022, vol. 10, 93104-93139.

[12] Lashkari, Arash Habibi, Andi Fitriah A. Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A. Ghorbani. "Towards a Network-based Framework for Android Malware Detection and Characterization." 15th Annual Conference on Privacy, Security and Trust (PST), IEEE, 2017, 233-23309.

[13] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, Ali A. Ghorbani "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. ", The 18th IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC), 2020, 515-522.