Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.063646

ARTICLE

Enhancing Android Malware Detection with XGBoost and Convolutional Neural Networks

Atif Raza Zaidi[1], Tahir Abbas[1,*], Ali Daud[2,*], Omar Alghushairy[3], Hussain Dawood[4] and Nadeem Sarwar[5]

[1]Department of Computer Science, TIMES Institute, Multan, 60000, Pakistan
[2]Faculty of Resilience, Rabdan Academy, Abu Dhabi, 114646, United Arab Emirates
[3]Department of Information Systems and Technology, College of Computer Science and Engineering, University of Jeddah, Jeddah, 23218, Saudi Arabia
[4]School of Computing, Skyline University College, University City Sharjah, Sharjah, 1797, United Arab Emirates
[5]Department of Computer Science, Bahria University, Lahore Campus, Lahore, 54600, Pakistan

*Corresponding Authors: Tahir Abbas. Email: drtahirabbas@t.edu.pk; Ali Daud. Email: alimsdb@gmail.com

Received: 20 January 2025; Accepted: 16 April 2025; Published: 03 July 2025

ABSTRACT: Safeguarding against malware requires precise machine-learning algorithms to classify harmful apps. The Drebin dataset of 15,036 samples and 215 features yielded significant and reliable results for two hybrid models, CNN + XGBoost and KNN + XGBoost. To address the class imbalance issue, SMOTE (Synthetic Minority Over-sampling Technique) was used to preprocess the dataset, creating synthetic samples of the minority class (malware) to balance the training set. XGBoost was then used to choose the most essential features for separating malware from benign programs. The models were trained and tested using 6-fold cross-validation, measuring accuracy, precision, recall, F1 score, and ROC AUC. The results are highly dependable, showing that CNN + XGBoost consistently outperforms KNN + XGBoost with an average accuracy of 98.76% compared to 97.89%. The CNN-based malware classification model, with its higher precision, recall, and F1 scores, is a secure choice. CNN + XGBoost, with its fewer all-fold misclassifications in confusion matrices, further solidifies this security. The calibration curve research, confirming the accuracy and cybersecurity applicability of the models' probability projections, adds to the sense of reliability. This study unequivocally demonstrates that CNN + XGBoost is a reliable and effective malware detection system, underlining the importance of feature selection and hybrid models.

KEYWORDS: Malware detection; android security; CNN; XGBooast; machine learning; deep learning

1 Introduction

As Android applications rapidly evolve and proliferate, the rise in malware threats has become a significant concern, highlighting the urgent need for innovative and effective detection techniques. This section describes the context, motivation, objectives, and contributions of this research, along with the structure of the paper. The exploration of machine learning and deep learning methods for Android malware detection is studied in this context to explain the synergistic effect of hybrid approaches involving XGBoost with Convolutional Neural Network (CNN) and K nearest neighbor (KNN).

In the age of Android and its growing penetration with diverse tasks, from professional to personal, detecting Android malware is essential in cybersecurity. Modern malware is too sophisticated for traditional detection methods, which often fail to protect because of its ability to bypass standard signature-based

systems. To tackle this, powerful tools such as machine learning and deep learning have emerged to deal with big data and discover complex patterns in massive datasets. XGBoost, a boosting algorithm, enhances feature selection, and CNNs efficiently learn entangled data representations. Hence, this study exploits these strengths and proposes a hybrid approach to leveraging this superiority for highly accurate and robust malware detection.

Detecting Android Malware is crucial because it protects international mobile users' privacy, security, and integrity. Because billions of active Android devices worldwide, their vulnerabilities are targets for malicious software, such as stealing sensitive data, spying on users' activity, or causing irrevocable system damage. Not only do individual users need to detect the malware, but companies, governments, and critical infrastructures also depend heavily on Android-powered devices [1]. The study explores methods for detecting malware on Android devices, combining deep learning and traditional machine learning algorithms, resulting in improved detection success and accuracy [2].

Android malware is highly diversified and evolves with time, making it hard to detect. Nowadays, malware is so advanced that it uses obfuscation to hide malicious intent, making typical signature-based detection mechanisms useless. Besides, the Android ecosystem has many legitimate applications, making the difference between benign and malicious software more difficult [3]. Additionally, real-time malware analysis encounters resource constraints on mobile devices, such as limited processing power and storage. In addition, detection systems must be scalable and capable of quickly processing large datasets, minimizing error rates [4].

Researchers have recently transitioned to advanced machine learning (ML) and deep learning (DL) methods to deal with such challenges. To demonstrate that algorithms such as XGBoost can, with proper data handling, extract meaningful patterns and probabilistic features from complex datasets, improving malware detection. Recent advances in DL, like Convolutional Neural Networks (CNNs), can be applied to extract complex relationships from input data, and there are many applications to malware classification problems. Finally, hybrid approaches are proposed to use the advantages of ML and DL approaches to improve detection system robustness. Another benefit of feature importance analysis is identifying key indicators of malicious behavior that result in high detection accuracy and low computational overhead. This is a new era of Android malware detection, focusing on adaptability and precision [5].

The XGBoost model is a powerful tool for malware detection on Android devices, combining XGBoost with Adaptive Particle Swarm Optimization for superior performance in security testing against real-world Android apps and benchmark datasets [6]. An exponential rise in Android applications is accompanied by increased malware aimed at user privacy and system integrity. Existing detection methods cannot scale or adapt to ever-changing malware. The requirement for solutions based on advanced machine learning and deep learning techniques motivates us. In this work, we aim to expand the feature space (using XGBoost) and perform classification (using CNN) to create a model that can reliably classify benign or malicious applications, improving detection performance and resilience to adversarial attacks.

This study aims to determine if XGBoost can be used as a feature enhancement technique for Android malware detection. It first evaluates the performance of hybrid models that blend XGBoost with Convolutional Neural Networks (CNN) and then compares their performance to XGBoost-enhanced K-Nearest Neighbors (KNN) classifiers. In addition, it seeks to investigate how feature selection influences detection accuracy and identify important features that play a key role in malware classification.

This study makes several key contributions to the field of Android malware detection, including:

- Proposes a novel hybrid approach combining XGBoost and CNN for improved Android malware detection.

- Demonstrates the superior performance of CNN + XGBoost over KNN + XGBoost across key metrics.
- Highlights the importance of feature selection in enhancing detection accuracy.
- Employs a robust methodology using the "Drebin" dataset and a 6-fold cross-validation strategy for validation in malware detection.

The remainder of this paper is organized as follows: Section 2 reviews related work in Android malware detection and hybrid modeling, and Section 3 details the methodology, including dataset preprocessing, feature enhancement, and model training. Section 4 presents the results and a comparative analysis of the proposed models. Section 6 discusses key findings, limitations, and future directions. Finally, Section 7 concludes the study and highlights its contributions to cybersecurity.

2 Related Literature

Android malware has grown to become sophisticated, requiring enhanced detection methods. HMLAD strategies, which combine ML and DL, have been developed. Such approaches use the pros of different algorithms in related tasks, including the feature extraction properties of DL models, including CNNs, and the classification potential of ML models, including XGBoost, to enhance the detectors' accuracy, extensiveness, and stability.

The open-source structure of Android combined with its leading market position makes it an attractive environment for cybercriminals. The Android ecosystem experiences frequent occurrences of trojans and adware and riskware and spyware which demonstrates the necessity for detection systems [7]. The number of Android applications has prompted a rise in cybersecurity challenges, and sophisticated malware is getting good at avoiding traditional signature-based detection. As a result, Machine Learning (ML) and Deep Learning (DL) hybrid approaches have been proposed to solve this problem. As one type of method, they use the complementarity between ML algorithms (XGBoost for feature selection) and hierarchical pattern recognition algorithms (Convolutional Neural Networks (CNNs)) like DL [8]. A study by the author of [9] demonstrated a high detection accuracy of 81.47% and the robustness of such hybrid approaches in mitigating modern malware challenges.

Malware threats are becoming increasingly complex, necessitating automated detection methods. Deep learning techniques, like polymorphism and code obfuscation, are being used, with the EfficientNet-B4 CNN-based model outperforming all others in Android malware detection [10]. Both feature ranking and classification accuracy have shown XGBoost's attributes of gradient boosting to perform excellently. Hence, Ref. [11] proposed hybrid CNNXGBoost, where the predicted output further feeds to the XGBoost for classification, and detection accuracies of 98.66% in hybrid CNN XGBoost models are reported as the integration reduces the complexity in the datasets efficiently. Additionally, Ref. [12] used traditional signature-based detection, which has shown difficulty keeping pace with increasingly sophisticated ransomware attacks, making it a real cybersecurity threat. Better alternatives include behavior-based techniques such as analyzing what people are doing on the system—for instance, looking at system commands or what downloads they were making of files.

Malware detection on Android devices is improved through a novel method using TuneDroid-enabled CNNs for detecting the utmost visual patterns, converting Android binaries to images, achieving 99.44% accuracy at training and 98.00% at validation [13]. The Android system faces security challenges due to open source and fragmentation. A new malware detection framework based on Convolutional Neural Network (CNN) was proposed, achieving a detection accuracy of 99.68%, surpassing other algorithms in large-scale and automation detection [14]. Android malware detection is crucial due to increasing mobile device usage. A deep learning framework using convolutional neural networks achieves 97.76% accuracy with minimal information requirements [15].

The study proposes a model using TF-IDF word representation and the XGBoost method to detect malicious applications on Android, classifying them as malware or benign in 2.75 s [16]. In particular, Ref. [17] contributed to the evolution of Android malware detection from static and dynamic analysis to creative image-based approaches. This led to a model based on the shape of CNN and ResNet, which convert apps or dex parts of apps to images and improve detection accuracy. ANNs [18] have also been evaluated for Android malware identification and have shown strong performance. Training and testing accuracy, along with average accuracy and precision, reached 0.99, while recall scored 0.98. This highlights the effectiveness of the ANN model.

This paper proposes a multi-dimensional feature fusion malicious application detection method based on Logistic Regression and XGBoost for Android applications. The technique extracts framework layer API call information, applies Logistic Regression, and fuses probability features with basic statistical features, improving accuracy and reducing time expenses in Android malicious application detection [19]. Because CNNs are designed to identify complex spatial patterns, Android's popularity makes it a significant target for malware attacks, necessitating advanced detection methods. This study combines machine learning, neural networks, and Federated Learning to enhance Android malware detection while ensuring data privacy. Techniques like majority voting and models such as Random Forest, XGBoost, and CNN-LSTM are evaluated to identify harmful behaviors effectively [20]. The study presents an advanced Android malware detection model using Convolutional Neural Networks (CNN) and a Light Gradient Boosting Machine (LGBM), achieving 96.73% accuracy and an F1 score, demonstrating the potential of integrating these technologies [21].

An XGBoost-based phishing detection system is developed by considering optimal feature selection to achieve 99.80% accuracy and high robustness in improving cybersecurity against phishing attacks [22]. Despite the effectiveness of machine learning (ML) for Android malware detection, as evidenced by recent studies, overly optimistic claims about detection accuracy typically arise from unrealistic experimental designs. In this thesis, we apply Explainable AI (XAI) to understand what the ML model learns during its training, and we demonstrate that temporal sample inconsistencies inflate performance metrics. The real-world relevance of ML-based malware detection emphasizes the need for realistic evaluation methods and XAI to explain ML methods in such scenarios [23]. Explainable Artificial Intelligence (XAI) enhances malware detection on Android devices by providing interpretable insights into machine learning model decisions, facilitating reliability and adoption in real-world applications [24].

Malware is a significant threat to computing systems, and researchers are working on new defense mechanisms to counter it. Hardware performance counters (HPCs) have been proposed, but their effectiveness is limited when sampled in realistic scenarios. Proper data preprocessing and the XGBoost classifier can improve malware detection performance by at least 15%, detecting malware early and maintaining low false positive rates [25]. For instance, recent advances in Android malware classification through deep learning automate feature extraction [26], allowing us to overcome the problems above. We propose an attention-based multimodal network (ANN) to utilize grayscale images and network traffic features by integrating cross-modal attention modules and transfer learning to help improve performance on classification tasks. Mobile *Ad-hoc* Network (MANET) security is enhanced with short digital signatures for efficient authentication in Secure AODV (SAODV), thereby effectively tackling the security problems without sacrificing the computational and resource constraints [27].

Android botnet detection has migrated from static and dynamic analysis to a hybrid approach. AKANDO is a typical case that combines static and dynamic analysis to understand app behavior comprehensively. Based on this technology, AKANDO uses a multi-layer neural network that adapts to threats in real-time while minimizing false positives, making the detection more accurate [28]. Android malware

detection is a complex task involving feature extraction, processing, performance evaluation, and datasets. AMD focuses on developing effective algorithms and models, particularly convolutional neural networks (CNN), to protect users' privacy and data security. This article reviews AMD techniques based on CNN, systematically reviewing data collection, preprocessing, feature extraction, representation, training, and evaluation [29].

A hybrid multimodal framework combining static and dynamic analysis with deep learning leverages model fusion to enhance Android malware detection. Using SHAP and t-SNE ensures explainability, while the approach consistently achieves over 99% detection accuracy across multiple models [30]. Security in Mobile *Ad-hoc* Networks (MANETs) is enhanced with Secure AODV (SAODV), which incorporates short digital signatures for efficient authentication, effectively addressing security threats while optimizing computational resources [31]. The increasing threat of Android malware has led to the development of deep learning-based classifiers to detect and mitigate malicious applications. However, these classifiers are vulnerable to adversarial attacks, such as Data Poisoning with Noise Injection (DP-NI) and Gradient-based Data Poisoning (GDP). A novel defense mechanism, Differential Privacy-Based Noise Clipping (DP-NC), is proposed to enhance the robustness of these classifiers. Experiments on three Android datasets show that DP-NC significantly reduces false-positive rates and improves classification accuracy, reducing false-positive rates by 45.46% and 7.67%, respectively [32].

Malware attacks result in privacy breaches and financial losses for all Androids. Malware detection requires deep learning approaches like XGBoost. The integration of feature enhancement is effective using a hybrid model composed of feature enrichment with CNNs [33]. In this study, Ref. [34] examines static, dynamic, and machine learning-based malware detection approaches and uses this knowledge to determine how such techniques can minimize the risk posed by malware. Thus, the CMD_2024 dataset unifies the static and dynamic attributes as a framework for analyzing different malware categories in the cloud environment. The work tackles the class imbalance problem using Conditional Tabular GANs and employs advanced Machine Learning and Deep Learning for effective malware detection, especially for rare malware types [35].

The study assesses the effectiveness of imputation by feature importance using various algorithms on soil radon and thoron gas concentration data over 14 months. The deep boosting model (DBP) consistently outperforms other models, revealing hidden patterns and recommending its use in the framework [36]. Therefore, it is necessary to detect Android malware to protect confidential information and not misrepresent facts. There are challenges, including obfuscation techniques and dataset issues, and static analysis, deep learning, and machine learning models are used [37]. This study introduces a novel approach using interval-bound propagation to enhance the resilience of both shallow and deep neural networks in intrusion detection systems (IDS). The study found that ReLu and Leaky-ReLu functions improve resistance in shallow networks, while Tanh functions perform better in deep networks [38].

Finally, real-time adaptive detection systems must be developed to keep pace with changing threats. Promising avenues for achieving minimal latency and robust detection performance using reinforcement learning and other adaptive techniques are discussed. These techniques are incorporated into hybrid frameworks to guarantee that malware detection remains robust against changing threats, creating a path toward resilient, future-ready cybersecurity systems.

Table 1 compares the methods discussed in this section to understand better how existing Android malware detection approaches differ. The comparison is made on the execution time, artificial intelligence techniques used, datasets, and key performance metrics. The strengths and limitations of each method can be seen in the table, and these can be used to get an idea of how well different approaches work in malware detection.

Table 1: Comparative study

Study	AI method used	Dataset	Execution time	Accuracy (%)	Advantages	Limitations
[39]	MSCN + ResNet	CICMalDroid2020 dataset	Fast	99.20%	High accuracy	Computational cost
[40]	Stacking technique	CIC-AndMal2017	Moderate	95.45%	Simple, interpretable	Struggles with high-dimensional data
[41]	Different deep learning techniques	Smote, SmoteTomek, Cluster-Centroid	Slow	85%–99%	Use different datasets	Hyperparamteter Tuning on DL and ML Techniques
[42]	CNN and hilbert space-filling curve	Custom dataset	Moderate	98.50	Robust, interpretable	Sensitive to noisy data

Research Gap

While significant achievements are observed in Android malware detection with hybrid Machine Learning (ML) and Deep Learning (DL) models, much is left untouched. While the current approaches (such as integrating Convolutional Neural Networks (CNNs) and XGBoost) already achieve impressive accuracy, they face difficulty in scale and adaptability in real-world applications. However, most studies concentrate on static or synthetic datasets that do not reflect real-world malware's dynamic and evolution characteristics. In addition, existing models are still challenging regarding adversarial resilience since they are susceptible to sophisticated evasion techniques. However, this does not scale under computational inefficiencies for real-time deployment within resource-limited environments such as mobile devices. Furthermore, while there has been some research into explainability in the context of XAI, it is used minimally. End users/practitioners do not have actionable information on how these complex models reached their decisions. Addressing these gaps to advance Android malware detection is crucial to developing lightweight, scalable, and interpretable hybrid frameworks that can handle real-time adversarial threats under the diverse Android malware landscape.

3 Methodology

The methodology for this study involves using the Drebin dataset [43] for malware detection, which consists of 15,036 samples and 215 features, with 5560 samples labelled as '1' (malware) and 9476 samples labelled as '0' (benign). The dataset is split into training and test sets, with the training set balanced using SMOTE (Synthetic Minority Oversampling Technique) to address class imbalance. The top 10 essential features for classification are identified using XGBoost, and their distributions are visualized through Kernel Density Estimation (KDE) plots. The models are trained using a 6-fold cross-validation strategy, with two distinct combinations evaluated: CNN + XGBoost and KNN + XGBoost. Model performance is assessed through key metrics such as accuracy, precision, recall, F1 score, and ROC AUC, with a comparative analysis conducted to highlight the strengths of each model. The calibration curve is also used to evaluate the reliability of predicted probabilities. Fig. 1 shows the proposed study below.

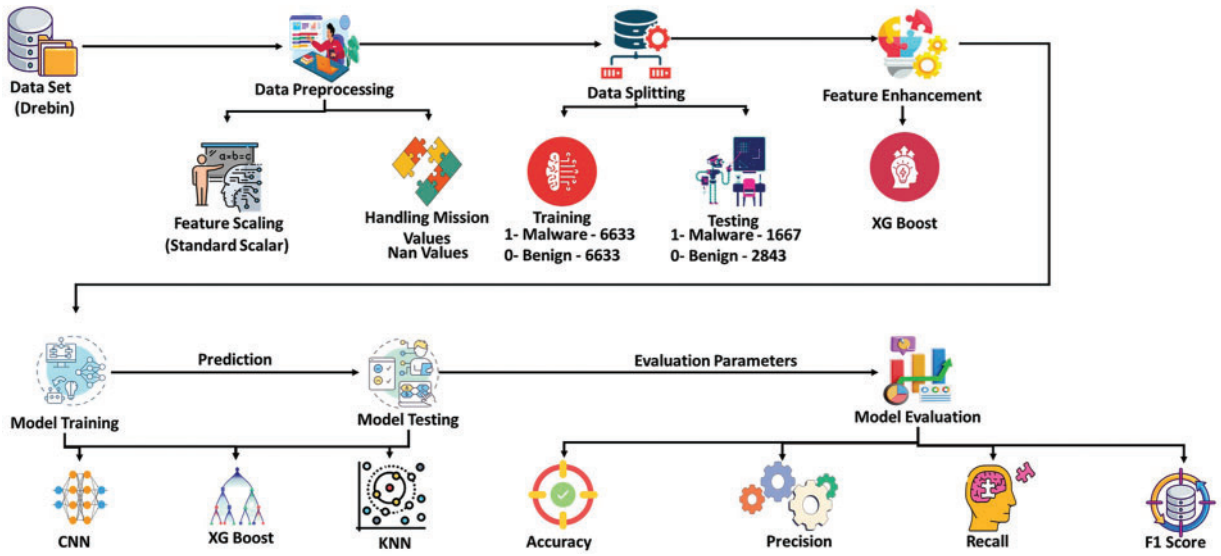

Figure 1: Proposed methodology

3.1 Dataset Preprocessing

While in the preprocessing stage, all missing values were handled, and feature scaling was enforced to ensure the quality and consistency of the dataset. The preprocessing of the Drebin dataset involves several key steps to ensure the data is ready for model training. First, the dataset is divided into two sets: training and test. The training set is further processed using SMOTE (Synthetic Minority Oversampling Technique) to balance the class distribution, ensuring equal representation of malware (class '1') and benign (class '0') samples. This helps mitigate bias toward the majority class (benign) during model training. The class distributions in the training set after SMOTE balancing are 6633 samples for malware and 6633 for benign, leading to 13,266 samples. The test set remains unaltered, with the original distribution: 1667 samples for malware and 2843 samples for benign, totaling 4510 samples. Feature extraction is performed next, with XGBoost to identify the top 10 critical features, which are then visualized using Kernel Density Estimation (KDE). These preprocessed steps ensure the dataset is well-structured for model evaluation while addressing potential issues related to class imbalance and feature relevance.

3.2 Dataset Splitting

The Drebin dataset is split into two main subsets: training and test sets. The training set is processed using SMOTE (Synthetic Minority Oversampling Technique) to balance the class distribution, ensuring that both classes—malware (class '1') and benign (class '0')—have an equal number of samples. After applying SMOTE, the training set consists of 13,266 samples, with 6633 samples for each class. This balanced training set allows the model to learn both classes effectively, preventing bias toward the majority class.

The test set remains unaltered to evaluate the model's performance objectively. It consists of 4510 samples, with 1667 samples for malware and 2843 samples for benign, reflecting the original distribution of the dataset. This separation ensures that the model is evaluated on data it hasn't seen during training, providing a fair assessment of its generalization capabilities. The training and test sets are then used for model development and evaluation through cross-validation and performance metrics.

3.3 Feature Enhancement Using XGBoost

Feature enhancement using XGBoost involves utilizing the model's ability to assess feature importance to identify and prioritize the most relevant features for malware detection. In this study, XGBoost is applied to the preprocessed Drebin dataset, a robust foundation containing 215 features, to generate importance scores for each feature based on its contribution to the model's predictive power. These scores help determine which features play the most significant role in distinguishing between malware and benign samples. The top-10 most important features identified by XGBoost include items like WRITE_EXTERNAL_STORAGE, TelephonyManager.getDeviceId, and SEND_SMS, all of which are strongly indicative of suspicious behaviors associated with malicious applications. To visualize and further understand the role of these features, Kernel Density Estimation (KDE) plots are used, showing the distribution of feature values across the dataset. These plots highlight how the top features vary and cluster, providing valuable insights into the characteristics that influence malware detection. This feature enhancement process aids in refining feature selection, potentially improving model accuracy and performance by focusing on the most influential variables.XGBoost, operating as an ensemble gradient boosting method, plays a pivotal role in our study. It minimizes an objective function consisting of two parts, showcasing its robustness and versatility in feature selection for malware detection.

$$£(\Theta) = \sum_{i=1}^{n} l\,(y_i + \hat{y}_i) + \sum_{k=1}^{k} \Omega(f_k) \tag{1}$$

where:

- $l\,(y_i + \hat{y}_i)$ is the loss function (e.g., log loss for classification),
- $\Omega(f_k)$ is a regularization term controlling model complexity,
- $\hat{y}_i$ is the predicted probability output by XGBoost for a given input sample.

XGBoost updates predictions using:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)2 \tag{2}$$

where $f_t(x_i)$ is the newly added weak learner (a decision tree) at iteration t. The three splits are determined by optimizing the following gain function:

$$G = \frac{1}{2}\left(\frac{\sum g_i}{H + \lambda}\right) \tag{3}$$

where:

- $g_i = \frac{\partial l}{\partial \hat{y}_i}$ is the first order gradient of the loss,
- $H = \sum h_i$ where $h_i = \frac{\partial^2 l}{\partial \hat{y}_i^2}$ is the second-order gradient (Hessiann),
- λ is the regularization parameter.

The final probabilistic output from XGBoost, $P(x)$, is computed using the sigmoid function:

$$P\,(x) = \frac{1}{1 + e^{-yi}} \tag{4}$$

These probability scores are concatenated with the original feature set before being passed into the CNN model.

This plot shows the Kernel Density Estimation (KDE) of the top features identified by XGBoost. Each curve represents the distribution of a specific feature's standardized values across the dataset. Features

like WRITE_EXTERNAL_STORAGE and TelephonyManager.getDeviceId have higher density near specific values, indicating their strong influence on classification. This graph highlights how the features vary and cluster within the dataset, helping to understand which features play a key role in distinguishing malware from benign samples. Fig. 2 shows the important XGBoost KDE distribution features.

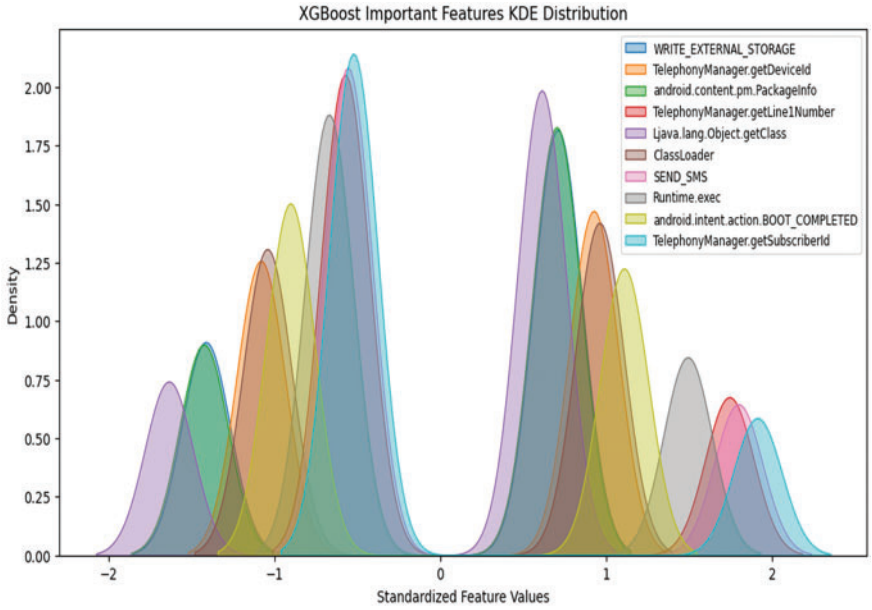

Figure 2: XGBoost important features KDE distribution

This plot adds an extra curve (XGBoost_Prob, shown in black dashed lines) representing the probability output of XGBoost for malware detection. The probability distribution indicates the model's confidence in classifying samples as malware (close to 1) or benign (close to 0). The top features distributions are shown alongside the probability, illustrating how they contribute to the model's decision-making process. Fig. 3 shows the XGBoost-enhanced KDE Distributed Features.

To improve the malware detection performance, XGBoost is used not only for feature selection but also to produce probabilistic output to participate in the classification of malware detection. In particular, with the dataset, the XGBoost model was trained to output probability scores describing how likely a given sample will belong to the malware class (Class 1). It then utilized these probability values as additional features to improve the learning capability of CNN. The preprocessing of Probabilistic Features is as follows:

1. **Transformation:** The probability scores from XGBoost were extracted and appended as new feature columns to the original dataset, effectively expanding the feature space.
2. **Normalization:** Since raw probability values range between 0 and 1, they were standardized using MinMax scaling to ensure numerical consistency with the other input features. This transformation ensures that the probabilistic outputs do not dominate CNN's learning process.
3. **Feature Concatenation:** After normalization, the transformed probability scores were concatenated with the original scaled feature set. This new augmented feature set was then reshaped into a format suitable for CNN processing.
4. **Input Formatting for CNN:** Since CNN requires a structured input format, the final dataset was reshaped into a 1D feature matrix before being passed through convolutional layers.

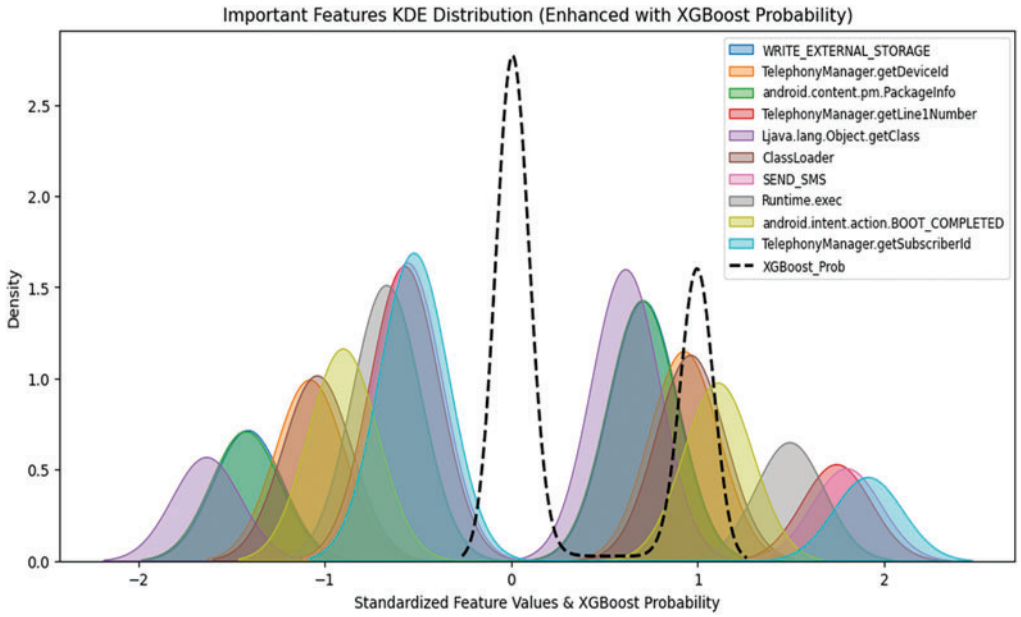

Figure 3: Important features KDE distribution (Enhanced with XGBoost Probability)

A calibration curve (also known as a reliability diagram) compares the predicted probabilities from our model with the actual observed probabilities. It helps determine whether the probabilities are well-calibrated. The blue line (XGBoost Calibration) generally aligns well with the gray line, indicating that the model's predicted probabilities are fairly reliable. Fig. 4 shows the calibration curve for the XGBoost probabilities curve.

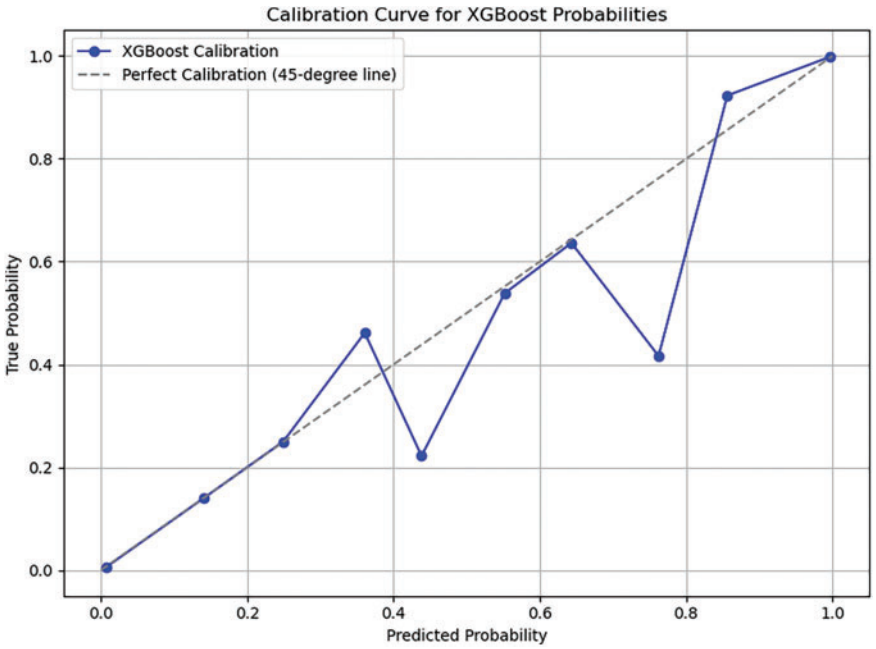

Figure 4: Calibration curve for XGBoost probabilities

3.4 Importance Feature Extraction (Top 10 Important Features)

XGBoost was used to extract the most important features for malware detection from the Drebin dataset, and the feature importance scores revealed which features significantly impact the classification of malware and benign samples. The top-10 most important features identified by XGBoost include WRITE_EXTERNAL_STORAGE, TelephonyManager.getDeviceId, SEND_SMS, Runtime. Exec, and others, many of which are associated with behaviors commonly exhibited by malicious applications. For instance, WRITE_EXTERNAL_STORAGE and SEND_SMS are frequently used by malware to gain unauthorized access to device storage or send unsolicited messages, while TelephonyManager.getDeviceId and TelephonyManager.getLine1Number are related to device identifiers that can be exploited for tracking and monitoring. These features play a crucial role in distinguishing malware from benign applications. Kernel Density Estimation (KDE) plots were generated to visualize their distributions across the dataset and gain deeper insights into how these features contribute to the classification process. These plots show how the values of these top features vary, helping to illustrate their distinctiveness and influence on the model's decision-making process. Features like WRITE_EXTERNAL_STORAGE and TelephonyManager.getDeviceId exhibit specific patterns in the dataset, enabling the model to better differentiate between malware and benign samples. Fig. 5 shows the important top 10 features.

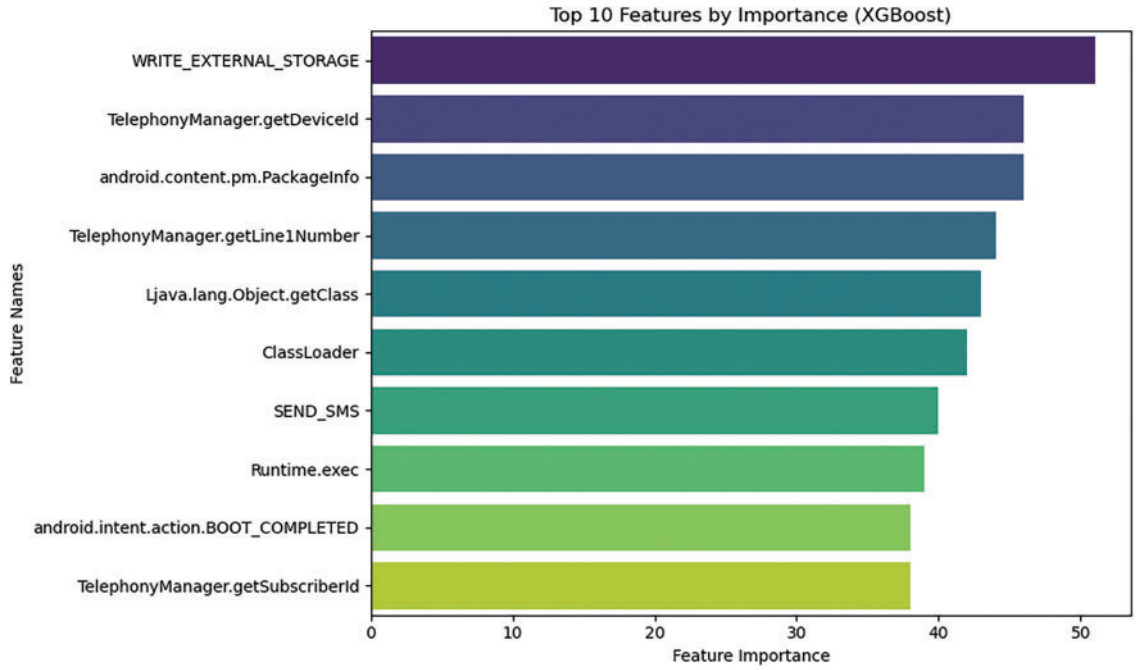

Figure 5: Top 10 features by importance

This feature importance analysis is a crucial step in understanding how the model makes decisions and guiding future efforts to refine the model or examine the behavior of specific features in malware detection. Fig. 5 and Table 2 show the importance of features XGBoost and KNN, respectively.

Table 2: Top 10 features extract using KNN

Feature	Importance
WRITE_EXTERNAL_STORAGE	51
TelephonyManager.getDeviceId	46
android.content.pm.PackageInfo	46
TelephonyManager.getLine1Number	44
Java.lang.Object.getClass	43
ClassLoader	42
SEND_SMS	40
Runtime. Exec	39
Android.intent.action.BOOT_COMPLETED	38

3.5 Model Training and Evaluation

Two separate models were developed and evaluated using a 6-fold cross-validation strategy.

3.5.1 CNN + XGBoost

The model training phase used a convolutional Neural Network (CNN) to classify the Android malware samples based on the previous step output feature set. The CNN architecture comprises key components to learn from the data and make accurate predictions. Conv1D layers were used to capture the spatial pattern of the sequence of features and included in the network. We followed the convolutional layers with MaxPooling1D layers to sample the feature maps, reduce the dimensionality, and focus only on the most essential features. Moreover, Dense layers were added for the network to learn more complex relationships between the features ending in the output classification. As for dropout regularization, it was introduced in Dense layers to ensure the model is not overfitting onto the training data by randomly setting the fraction of input units to zeros every epoch, enhancing the model's generalization ability.

Early stopping was also implemented to optimize the training process and skip unnecessary computation by monitoring the validation loss during training. Further, the training was stopped after a predefined number of epochs if the validation loss did not improve, as we did not want to overfit and waste extraneous epochs, thereby saving time and computational power.

Finally, the transforming features and the probabilistic outputs of XGBoost were included in the feature set where the CNN model was trained. Using convolutional layers, the model captured the data's spatial dependencies and complex relationships. It accepted the augmented feature space as its input layer to ensure that CNN was learning both the original feature representation and the XGBoost probability-driven insights.

CNNs operate by convolving input features through layers that extract hierarchical patterns. Given an input feature set X, the convolution operation is defined as:

$$Z_{i,j}^{(l)} = f\left(\sum_{m}\sum_{n} W_{m,n}^{(l)} X_{i+m,j+n}^{(l-1)} + b^{(l)}\right) \tag{5}$$

where:

- $W_{m,n}^{(l)}$ represents the convolution filter weights,
- $b^{(l)}$ is the bias term,
- f is the activation function (ReLU in this case),

- $X_{i+m,j+n}^{(l-1)}$ is the input from the previous layer.

For classification, CNNs use the softmax activation function in the final layer:

$$\hat{y}_i = \frac{e^{zi}}{\sum_j e^{zj}} \tag{6}$$

where:

zi represents the logit for class i.

The denominator normalizes the output as a probability distribution.

$$£_{CNN} = \sum_{i=1}^{N} y_i \log \hat{y}_i \tag{7}$$

where:

- y_i is the true class lable (malware or engin),
- $\hat{y}_i$ is the predicted probability.

Two separate models were developed and evaluated using a 6-fold cross-validation strategy shown below, see Fig. 6.

Algorithm: XGBoost + CNN approach with SMOTE and Early Stopping

1: **Load the Drebin dataset** *(e.g., from dataset.csv)* into *data*.
2: Clean column names, replace "?" with NaN, and drop rows with missing values.
3: Map class labels: 'S' →1 (Malware), 'B' → 0 (Benign).
4: Convert columns to numeric and drop any remaining rows with NaN.
5: Separate features **X** and target **y**.
6: **Scale features X** via *StandardScaler* to get $\mathbf{X}_{scaled}$.
7: Train an **XGBoost** classifier on $\mathbf{X}_{scaled}$, **y**.
8: Obtain probability outputs $\mathbf{X}_{prob}$ from the trained XGBoost model.
9: Concatenate $\mathbf{X}_{scaled}$ and $\mathbf{X}_{prob}$ to form $\mathbf{X}_{combined}$.
10: Initialize *K*-fold cross-validator with *n splits* = 6.
11: **for** each fold in the *K*-fold cross-validation **do**
12: Split $\mathbf{X}_{combined}$ and **y** into training and testing sets.
13: **Apply SMOTE** to the training set ($\mathbf{X}_{train}$, $\mathbf{y}_{train}$) to handle class im- balance.
14: Reshape training and test features to (samples, features, 1) for CNN input.
15: Define and compile a **CNN model** with layers for convolution, pooling, dropout, and a final sigmoid output.
16: **Set Early Stopping** (monitor=val loss, patience=10) to prevent overfitting.
17: **Fit the CNN model** on the SMOTE-augmented training data, using validation on the test set.
18: **Predict on test set**, compute metrics (Accuracy, Precision, Recall, F1-score, ROC AUC).
19: **end for**
20: Report **average metrics** (Accuracy, Precision, Recall, F1, AUC) across the *K* folds.

Figure 6: XGBoost + CNN approach with SMOTE and early stopping

Training and validation loss and training and validation accuracy were plotted to monitor the model's performance across each cross-validation fold. These metrics directly illustrated the model's learning pace and were essential to fine-tuning it to its best performance. Fig. 7 shows the accuracy and loss curve for the proposed models.

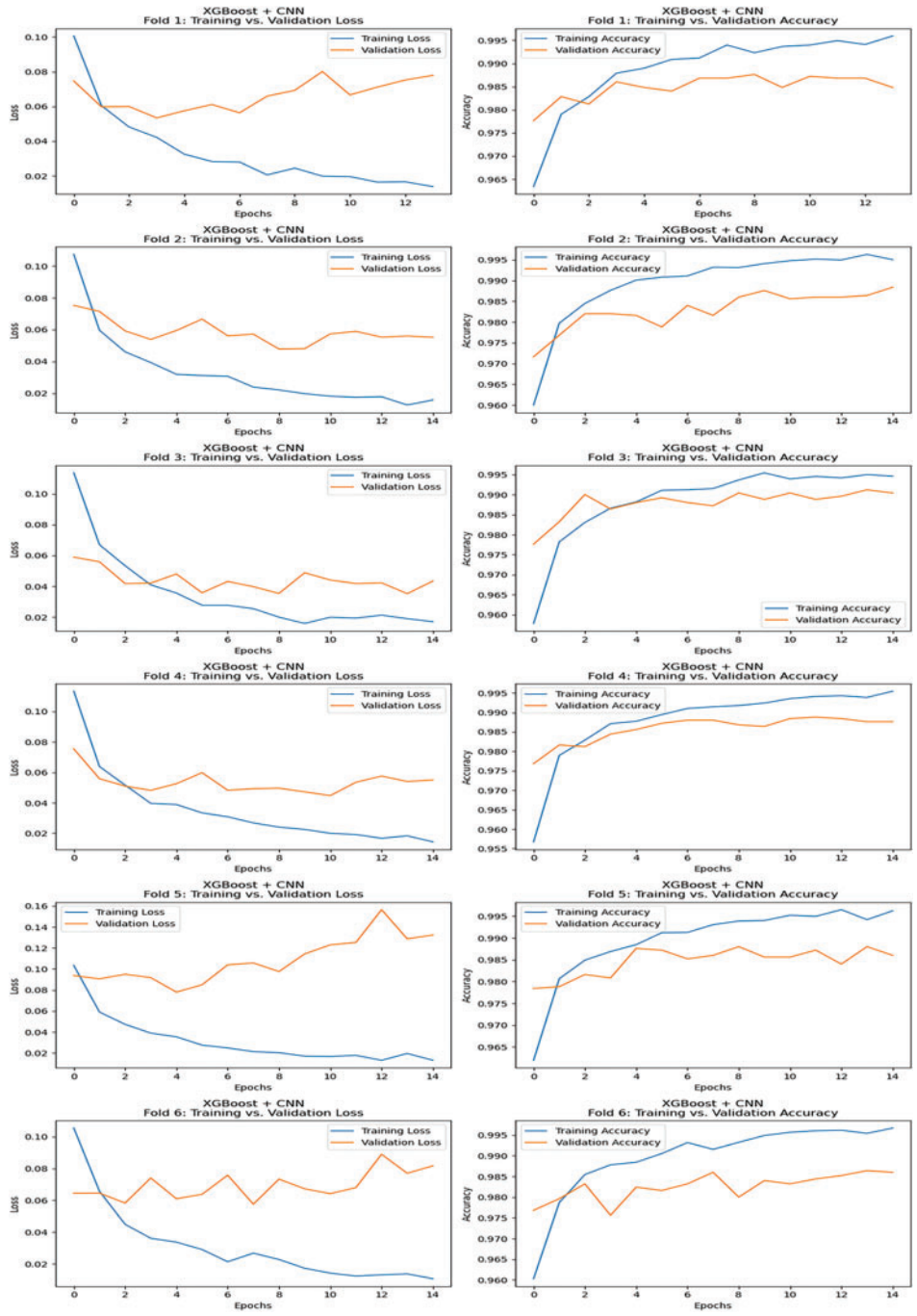

Figure 7: Training, validation loss, and accuracy curve

3.5.2 KNN + XGBoost

In addition to CNN, a K-Nearest Neighbors (KNN) classifier was used to predict the labels of the samples based on the features enhanced in the previous steps. This KNN classifies a sample by the majority label in the K nearest neighbors in the feature space in which the sample lies. In this study, the class of a given sample is decided by assigning the majority class of the 5 nearest data points in the feature space (number of neighbors K = 5).

Performance metrics were computed for the KNN model like the CNN model, including accuracy, precision, recall, F1 score, and ROC AUC for each fold of the cross-validation process. Using these metrics, we could evaluate the model and predict which samples were malware or benign. A good measure of how the KNN model performs was calculated by computing the average of these metrics across all folds. By doing this, we ensure that the results are not dependent on some particular fold and provide a more generalized view of the model's performance.

Additionally, we generated confusion matrices for both the CNN and KNN models. These matrices indicate the true positives, true negatives, false positives, and false negatives and visually represent how our models do well and where they have failed. In the Comparative Analysis section, we share the confusion matrices and compare performance metrics to compare the effectiveness of these two models directly. The algorithm for this technique is given below in Fig. 8.

Algorithm: XGBoost + KNN approach with SMOTE

1: **Load the Drebin dataset** (e.g., from dataset.csv) into *data*.
2: Clean column names, replace "?" with NaN, drop rows with missing values.
3: Map class labels: 'S' →1 (Malware), 'B' → 0 (Benign).
4: Convert columns to numeric and drop any remaining NaNs.
5: Separate features **X** and target **y**.
6: **Scale** the features **X** (e.g., *StandardScaler*) to get $\mathbf{X}_{scaled}$.
7: Train an **XGBoost** classifier on $\mathbf{X}_{scaled}$, **y**.
8: Obtain XGBoost probability outputs $\mathbf{X}_{prob}$.
9: Concatenate $\mathbf{X}_{scaled}$ and $\mathbf{X}_{prob}$ to form $\mathbf{X}_{combined}$.
10: Initialize *K*-fold cross-validator (e.g., *n splits* = 6).
11: **for** each fold in the *K*-fold cross-validation **do**
12: Split $\mathbf{X}_{combined}$ and **y** into training and testing sets.
13: **Apply SMOTE** to the training set to handle class imbalance.
14: Define and train a **KNN classifier** (e.g., *n neighbors* = 5) on the SMOTE-augmented training data.
15: Predict on the test set, and compute metrics (Accuracy, Precision, Recall, F1-score, ROC AUC).
16: **end for**
17: Report **average metrics** (Accuracy, Precision, Recall, F1, AUC) across the *K* folds.

Figure 8: XGBoost + KNN approach with SMOTE

4 Results & Comparative Analysis

In this section, we presents the experiments designed to measure the performance of the proposed hybrid models on Android malware detection. The performance of the models CNN + XGBoost and KNN + XGBoost were measured using several performance metrics, such as accuracy, precision, recall, F1-score, and ROC AUC. Furthermore, classification performance was also assessed based on confusion

matrices. All models were evaluated with a 6-fold cross-validation strategy to ensure robust and reliable results. In the following subsections, we present a performance comparison between the models and key metrics and conduct a detailed analysis of the efficacy of Android malware detection.

4.1 Comparison Average of Performance Metrics across All Folds

The performance of the CNN + XGBoost and KNN + XGBoost models was compared using multiple evaluation metrics, including Accuracy, Precision, Recall, F1 Score, and ROC AUC. The results indicate that the CNN + XGBoost model consistently outperformed the KNN + XGBoost model across all metrics. Specifically, CNN + XGBoost achieved a higher average accuracy of 98.76% compared to 97.89% for KNN + XGBoost, demonstrating its superior ability to classify malware and benign samples correctly. In terms of precision, the CNN + XGBoost model attained 98.39%, outperforming the KNN + XGBoost model, which scored 96.71%, indicating that CNN + XGBoost had fewer false positives. The recall of CNN + XGBoost was also slightly higher at 98.27% compared to 97.61% for KNN + XGBoost, showing its better capability in correctly identifying malware instances. Similarly, the F1 score, which balances precision and recall, was 98.33% for CNN + XGBoost, surpassing the 97.15% achieved by KNN + XGBoost. Lastly, the ROC AUC score, which reflects the model's ability to distinguish between malware and benign samples, was 0.9980 for CNN + XGBoost, significantly higher than 0.9919 for KNN + XGBoost, confirming that CNN + XGBoost is effective in malware classification. Table 3 illustrates the comparison average of performance metrics (e.g., accuracy, precision, recall, F1 Score, and ROC).

Table 3: Comparison average of performance metrics

Metric	CNN + XGBoost	KNN + XGBoost
Average accuracy	0.9876	0.9789
Average precision	0.9839	0.9671
Average recall	0.9827	0.9761
Average F1 Score	0.9833	0.9715
Average ROC AUC	0.9980	0.9919

4.2 Confusion Matrix XGBoost + CNN Model of All Folds

The performance of the CNN + XGBoost model remained consistently high across all six folds, demonstrating strong classification capabilities with minimal misclassifications. In Fold 1, the model achieved an accuracy of 98.6%, a precision of 97.9%, and a recall of 98.4%, indicating a well-balanced performance. Fold 2 exhibited similar robustness, with an accuracy of 98.8%, a precision of 98.7%, and a recall of 98.2%, highlighting its ability to minimize false positives and false negatives effectively. Fold 3 recorded the highest accuracy of 99.0%, with a precision of 98.5% and recall of 98.9%, demonstrating the model's reliability. In Fold 4, the model maintained an accuracy of 98.8%, a precision of 99.2%, and a recall of 97.5%, further proving its efficiency in detecting malware. Fold 5 followed a similar trend with 98.8% accuracy, 98.4% precision, and 98.1% recall, reinforcing the stability of the model. Lastly, Fold six showcased 98.6% accuracy, 97.6% precision, and 98.6% recall, confirming the model's consistency across multiple evaluations. The consistently high performance across all folds highlights the effectiveness of CNN + XGBoost in malware classification, ensuring reliable detection with minimal classification errors. Fig. 9 and Table 4 display six confusion matrices corresponding to the performance of an XGBoost + CNN model across six folds.

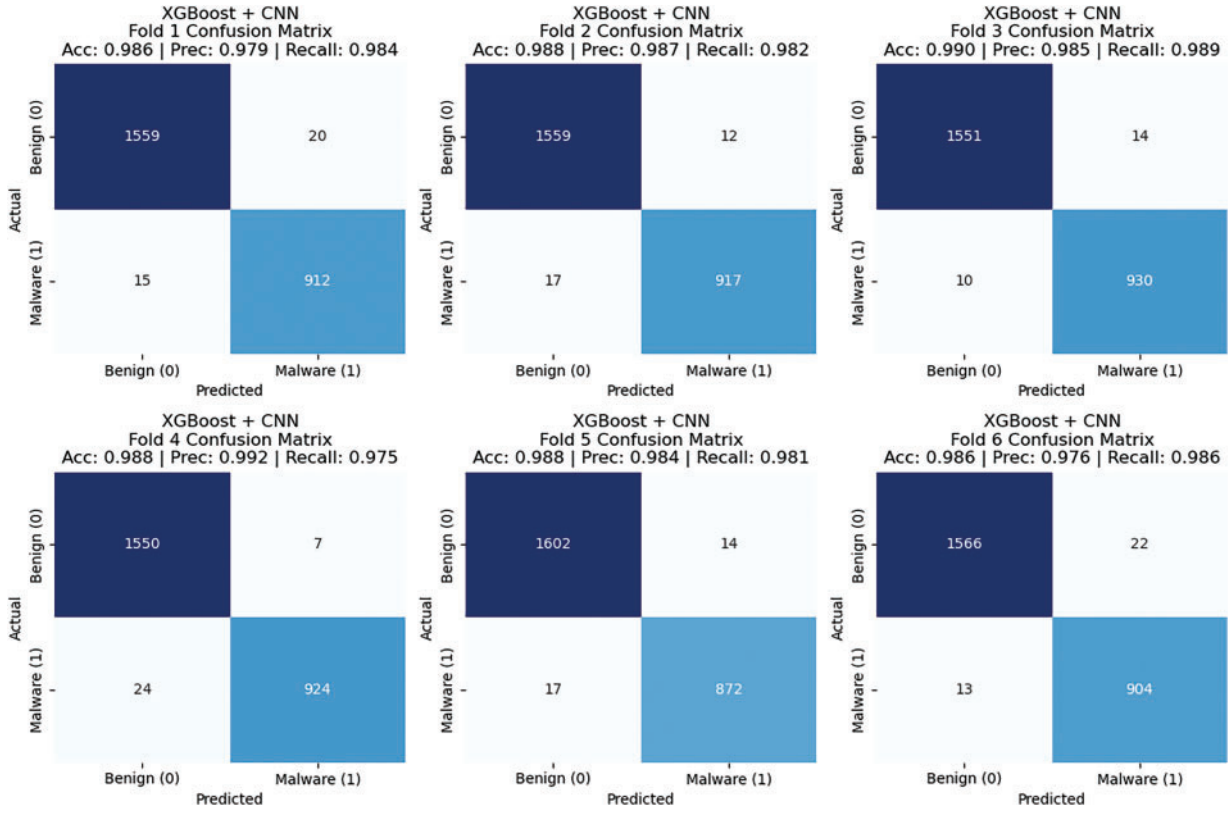

Figure 9: Confusion metrix for XGBoost + CNN 6 of all folds

Table 4: Performance metrics of XGBoost + CNN model across six K-Folds

Fold	Accuracy	Precision	Recall
Fold 1	0.986	0.979	0.984
Fold 2	0.988	0.987	0.982
Fold 3	0.99	0.985	0.989
Fold 4	0.988	0.992	0.975
Fold 5	0.988	0.984	0.981
Fold 6	0.986	0.976	0.986

The results confirm the effectiveness of the hybrid model CNN + XGBoost in discriminating between benign and malware samples. The metrics in these metrics are reinforced visually through confusion matrices from each fold and show a reliable and well-performing model across data splits.

4.3 Confusion Matrix XGBoost + KNN Model of All Folds

The XGBoost + KNN model demonstrated consistently strong performance across all six folds, achieving an accuracy ranging from 97.6% to 98.2%, indicating a stable and reliable classification capability. The model maintained high precision and recall values, with precision ranging between 95.7% and 97.7% and recall between 96.6% and 98.3%, ensuring that false positives and false negatives were kept to a minimum. Despite minor misclassifications, the model effectively distinguished between benign and malware samples,

with only a tiny percentage of benign instances misclassified as malware and *vice versa*. Compared to the XGBoost + CNN model, the XGBoost + KNN model performed slightly lower in key metrics but maintained overall robustness and consistency. These results highlight that the XGBoost + KNN model is a highly effective classifier, achieving an average accuracy above 97.5%, reinforcing its capability to distinguish malware from benign samples with high precision and recall. Fig. 10 and Table 5 display six confusion matrices corresponding to the performance of an XGBoost + KNN model across six folds.

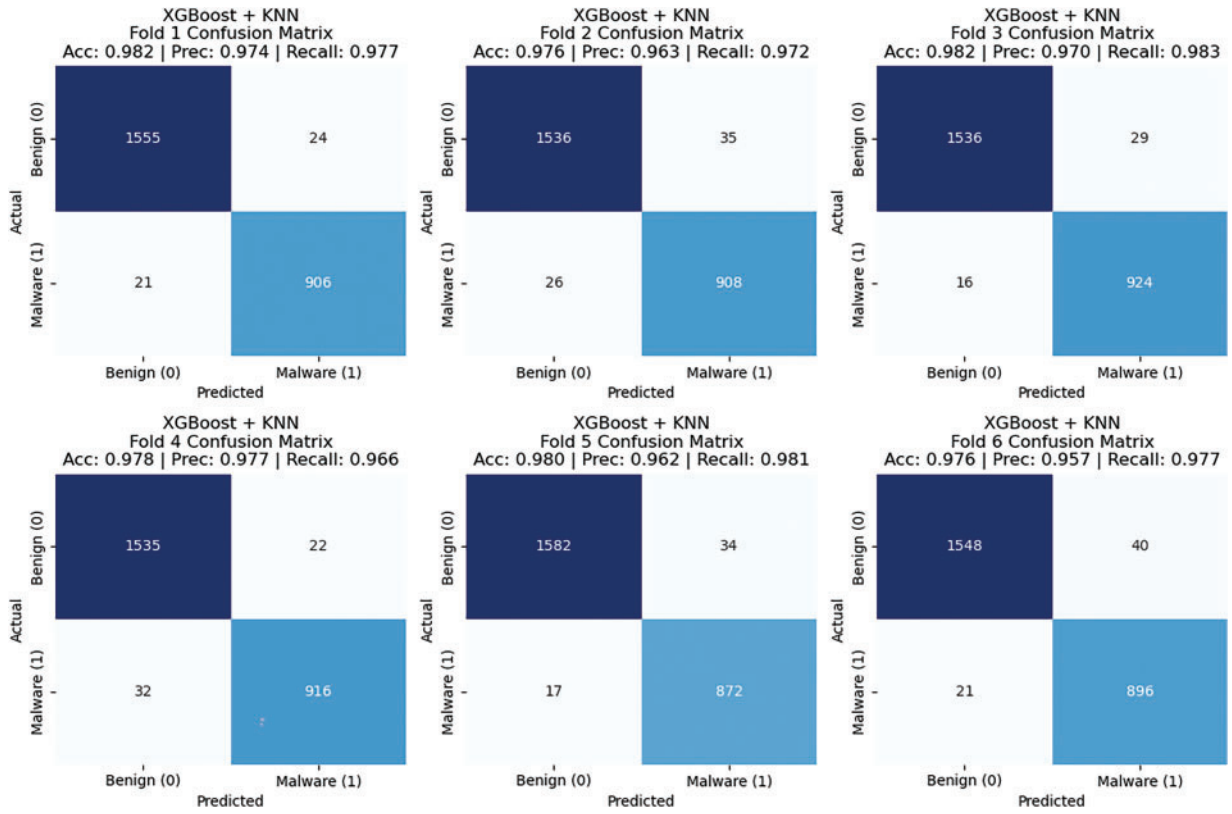

Figure 10: Confusion metrix for XGBoost + KNN 6 of all folds

Table 5: Performance metrics of XGBoost + KNN model across six K-Folds

Fold	Accuracy	Precision	Recall
Fold 1	0.982	0.974	0.977
Fold 2	0.976	0.963	0.972
Fold 3	0.982	0.97	0.983
Fold 4	0.978	0.977	0.966
Fold 5	0.98	0.962	0.981
Fold 6	0.976	0.957	0.977

The results above motivate the combination of feature extraction through XGBoost and the simplicity of KNN for robust malware detection.

5 Comparative Analysis with Existing Malware Detection Methods

Many Android malware detection investigations employ CNN or XGBoost separately. Both approaches have shown promise, but using them separately may have drawbacks. This study compares various methods to our CNN + XGBoost hybrid model based on performance measures, computational efficiency, and scalability.

CNNs are effective at feature extraction from raw data like byte sequences or API calls. According to several studies, the CNN-based technique has excellent accuracy and recall but low precision and F1 score because of its sensitivity to noisy data and overfitting on unbalanced datasets.

XGBoost is a robust gradient-boosting technique that can handle imbalanced datasets. This means it can effectively manage datasets where one class of data is significantly more or less frequent than the others. Beyond this, XGBoost may be unable to find CNNs' complex hierarchical features, such as intricate spatial and temporal patterns. Because of this, XGBoost-based models have higher precision but lower recall or more false negatives.

However, XGBoosts use fewer resources than CNNs. Despite its fast training time, it can train on CPU-based platforms with low memory utilization. It efficiently uses gradient boosting and tree-based learning methods and takes minutes on large datasets. Nevertheless, XGBoost models may still be quite sensitive to hyperparameter tuning and consequently demand computational effort.

CNNs are good at detection, but scaling to huge datasets is problematic. As the data set gets more significant, computing power (such as multi-GPU setups) must be more powerful. Retraining a model with new data is also difficult. CNNs typically need to be rewritten from scratch when new data is presented, which is time-consuming and resource-consuming. However, XGBoost is recognized for its scalability. Compared to CNNs, it can handle larger datasets more efficiently, notably in distributed environments where the processing can be split across numerous workstations or cloud-based platforms. XGBoost scales well even with millions of data points and can train datasets larger than memory with minimum system memory using out-of-core training.

Due to task separation, the CNN + XGBoost hybrid model scales effectively despite its complexity. XGBoost classifiers may use many processors to process massive datasets, while CNNs can extract features in parallel. This architecture helps the hybrid model scale better than the original model in commercial environments with massive datasets, providing a sense of security about its scalability.

The hybrid model is more scalable than CNN and XGBoost because it divides the workload between two complementary components that process data efficiently and have low computational complexity.

In conclusion, the CNN + XGBoost hybrid model has many advantages compared to the existing methods based on using either CNN or XGBoost alone. Key points include:

The hybrid model has improved accuracy, precision, recall, and F1 score compared to the CNN and XGBoost models. This is done by drastically minimizing the training time and resource utilization compared to CNN alone models while approaching near the performance of CNN solely models. The hybrid approach outperforms the interpretable model in scalability and performs well with high-dimensional datasets, leveraging distributed computing for efficient processing. This hybrid technique increases malware detection accuracy and provides computational efficiency and scalability for practical malware detection systems.

6 Discussion

The CNN + XGBoost model consistently demonstrated superior classification performance across all evaluation metrics. Achieving an average accuracy of 98.76%, it effectively identified malware and benign

samples with minimal errors. Although slightly less effective, the KNN + XGBoost model still exhibited strong classification capabilities, maintaining an accuracy of 97.89% across all folds. Precision and recall scores confirmed the CNN model's robustness, indicating fewer false positives and negatives than the KNN-based approach. The consistently high performance of CNN + XGBoost suggests that deep learning models offer significant advantages in malware detection when combined with powerful boosting algorithms.

Feature extraction played a key role in the classification process. XGBoost's feature selection revealed that attributes such as WRITE_EXTERNAL_STORAGE, TelephonyManager.getDeviceId, and SEND_SMS were critical in distinguishing malware from benign applications. These permissions and API calls are commonly exploited by malicious applications, highlighting the effectiveness of machine learning in identifying behavioral patterns associated with malware. The Kernel Density Estimation (KDE) plots provided further insights into how these top features varied within the dataset, showing clear distinctions between malware and benign samples.

Confusion matrix analysis across six K-Folds illustrated the models' performance stability. The CNN + XGBoost model had consistently fewer misclassifications, effectively detecting malware samples while maintaining a low false positive rate. In contrast, the KNN + XGBoost model showed slightly higher variability, occasionally misclassifying benign applications as malware. However, the overall classification performance remained strong, reinforcing the effectiveness of XGBoost's feature selection in improving traditional machine learning models.

The calibration curve analysis confirmed that the models provided well-calibrated probability outputs, ensuring reliability in malware detection. Properly calibrated models are crucial in real-world cybersecurity applications, where detection systems must confidently assess the likelihood of an application being malicious. The CNN-based approach exhibited better calibration, aligning closely with observed probabilities, making it a more dependable choice for threat detection systems.

Despite its slightly lower performance, the KNN + XGBoost model proved a stable and computationally efficient alternative. For scenarios where deep learning-based models may be impractical due to hardware constraints, the KNN-based approach remains a viable solution. However, in high-stakes environments where maximizing detection accuracy is critical, CNN + XGBoost remains the preferred model due to its superior generalization and robustness.

This study demonstrates the significant impact of feature selection, hybrid models, and deep learning architectures on malware classification performance. By integrating XGBoost for feature enhancement and leveraging the strengths of CNN and KNN, the models effectively identified malicious applications with high accuracy. The findings suggest that CNN + XGBoost should be further explored for large-scale malware detection systems, potentially incorporating real-time classification techniques for enhanced cybersecurity.

Finally, both the models showed good performance in terms of Android malware detection, and the CNN + XGBoost hybrid model outperformed the KNN + XGBoost model because of its ability to learn more sophisticated patterns in the data, good sensitivity for malware detection (reflected by higher recall and ROC AUC) and additional features produced by XGBoost. These results indicate that the CNN-based approach is best suited for complex high-dimensional data such as malware detection because of its ability to learn and extract appropriate features automatically. However, in this scenario, KNN is not quite as effective as that. Still, in some scenarios, with simpler feature spaces, or where interpretability is paramount, KNN can be a helpful tool.

As a valuable step towards a comparative understanding of different machine learning techniques applied towards Android malware detection and their respective strengths and limitations, it demonstrates

that hybrid models combining the strengths of various algorithms tend to outperform other models in complex tasks like malware classification.

7 Conclusion

This study tested CNN + XGBoost and KNN + XGBoost models for malware identification using 15,036 samples and 215 features from the Drebin dataset. XGBoost extracted the top-10 malware classification features after SMOTE balanced the dataset. The models were evaluated using 6-fold cross-validation, accuracy, precision, recall, F1 score, and ROC AUC. CNN + XGBoost outscored KNN + XGBoost in all evaluation measures. CNN + XGBoost had 98.76% accuracy, while KNN + XGBoost had 97.89%, demonstrating CNN's greater generalization over complicated feature representations. CNN had 98.39% precision compared to 96.71% for KNN + XGBoost, indicating less false positives. CNN has superior malware detection recall (98.27% vs. 97.61%) than KNN + XGBoost. CNN + XGBoost's F1 score, which balances precision and recall, was 98.33% and 97.15% for KNN + XGBoost, confirming the CNN model's malware detection reliability. Finally, CNN + XGBoost had a higher ROC AUC score (0.9980) than KNN + XGBoost (0.9919), proving its superior classification performance. CNN + XGBoost had fewer misclassifications than KNN across all six-folds, with few false positives and negatives. The calibration curve analysis confirmed CNN + XGBoost's reliable probability forecasts, making it a better model for cybersecurity applications. The KNN + XGBoost model had great accuracy and computational efficiency but higher classification variability. CNN + XGBoost is a robust and reliable malware detection model with high accuracy (98.76%), precision (98.39%), recall (98.27%), and F1 score (98.33%), making it suited for cybersecurity threat detection systems. The results suggest that CNN + XGBoost is best for high-stakes malware detection and KNN + XGBoost for lower-complexity scenarios. In future studies, real-time malware detection, adversarial robustness, and feature representation expansion can improve classification accuracy and reliability. Because malware threats evolve, this hybrid technique could grow to more enormous real-world datasets. Time detection frameworks for resource-constrained devices like smartphones and IoT systems should be lightweight and computationally efficient. Explanatory AI (XAI) techniques improve interpretability and expand cross-platform malware detection for Windows and iOS. The method's effectiveness against Android malware and internet safety would be proven.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: Atif Raza Zaidi, Tahir Abbas, and Ali Daud collected data from different resources. Atif Raza Zaidi, Ali Daud, Hussain Dawood, and Nadeem Sarwar contributed to writing—the original draft preparation. Ali Daud, Tahir Abbas, Atif Raza Zaidi, and Omar Alghushairy contributed to the writing—review and editing. Ali Daud, Omar Alghushairy, and Tahir Abbas supervised the paper. Hussain Dawood, Nadeem Sarwar, and Atif Raza Zaidi drafted pictures and tables. Tahir Abbas and Ali Daud performed revisions and improved the quality of the draft. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: This paper contains all the data.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Alzubaidi A. Detecting Android malware using deep learning algorithms: a survey. Comput Electr Eng. 2024;119(1):109544. doi:10.1016/j.compeleceng.2024.109544.
2. Alkahtani H, Aldhyani THH. Artificial intelligence algorithms for malware detection in Android-operated mobile devices. Sensors. 2022;22(6):2268. doi:10.3390/s22062268.
3. Smmarwar SK, Gupta GP, Kumar S. Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: a comprehensive review. Telematics Inform Rep. 2024;14:100130. doi:10.1016/j.teler.2024.100130.
4. Akhtar Z. Malware detection and analysis: challenges and research opportunities. arXiv:2101.08429. 2021.
5. Alzubaidi L, Zhang J, Humaidi AJ, Al-Dujaili A, Duan Y, Al-Shamma O, et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data. 2021;8(1):53. doi:10.1186/s40537-021-00444-8.
6. Kumar P, Singh S. Security testing of Android apps using malware analysis and XGBoost optimized by adaptive particle swarm optimization. SN Comput Sci. 2023;5(1):92. doi:10.1007/s42979-023-02411-x.
7. Kural OE, Kiliç E, Aksaç C. Apk2Audio4AndMal: audio based malware family detection framework. IEEE Access. 2023;11:27527–35. doi:10.1109/access.2023.3258377.
8. Yeboah C. Demalvertising: enhanced detection and mitigation of malvertising on Android devices. Berekuso, Ghana: Ashesi University; 2024.
9. Fatima N, Khan HF. A comprehensive analysis and evaluation of Android malware prediction using AI. In: 2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS); 2024 Jan 28–29; Manama, Bahrain. p. 1–5. doi:10.1109/ICETSIS61505.2024.10459543.
10. Yadav P, Menon N, Ravi V, Vishvanathan S, Pham TD. EfficientNet convolutional neural networks-based Android malware detection. Comput Secur. 2022;115(11):102622. doi:10.1016/j.cose.2022.102622.
11. Gadilohar P, Tomar DS, Dehalwar V, Sharma YK. Integrating CNN and XGBoost with synthetic samples for advanced Android malware detection. In: 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT); 2024 Jun 24–28; Kamand, India. p. 1–6. doi:10.1109/ICCCNT61001.2024.10725305.
12. Rahman MS, Ahmed Sabbir MS, Ghosh S. Ransomware attack detection using machine learning approaches. In: 3rd International Conference for Innovation in Technology (INOCON); 2024 Mar 1–3; Bangalore, India; 2024. p. 1–7. doi:10.1109/INOCON60754.2024.10512276.
13. Bakır H. A new method for tuning the CNN pre-trained models as a feature extractor for malware detection. Pattern Anal Applic. 2025;28(1):26. doi:10.1007/s10044-024-01381-x.
14. Wang Z, Li G, Chi Y, Zhang J, Yang T, Liu Q. Android malware detection based on convolutional neural networks. In: Proceedings of the 3rd International Conference on Computer Science and Application Engineering; 2019 Oct 22–24; Sanya, China. p. 1–6. doi:10.1145/3331453.3361306.
15. Lakshmanarao A, Shashi M. Android malware detection using convolutional neural networks. In: Bhateja V, Satapathy SC, Travieso-González CM, Aradhya VNM, editors. Data engineering and intelligent computing. Advances in intelligent systems and computing. Vol. 1407. Singapore: Springer; 2021. p. 151–62. doi: 10.1007/978-981-16-0171-2_15.
16. Ozogur G, Ali Erturk M, Gurkas Aydin Z, Ali Aydin M. Android malware detection in bytecode level using TF-IDF and XGBoost. Comput J. 2023;66(9):2317–28. doi:10.1093/comjnl/bxac198.
17. Lakshmanarao A, Madhuri PB, Dasari K, Babu KA, Sulthana SR. An efficient Android malware detection model using Convnets and Resnet Models. In: 2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS); 2024 Aug 23–24; Hassan, India. p. 1–6.
18. Habeeb MA, Khaleel YL. Enhanced Android malware detection through artificial neural networks technique. Mesopotamian J CyberSecurity. 2025;5(1):62–77. doi:10.58496/mjcs/2025/005.
19. Li S, Huang X. Android malware detection based on logistic regression and XGBoost. In: 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS); 2019 Oct 18–20; Beijing, China. p. 528–32. doi:10.1109/icsess47205.2019.9040851.

20. Purkayastha BS, Rahman MM, Shahpasand M. Android malware detection using machine learning and neural network: a hybrid approach with federated learning. In: 2024 7th International Conference on Advanced Communication Technologies and Networking (CommNet); 2024 Dec 4–6; Rabat, Morocco. p. 1–5. doi:10.1109/CommNet63022.2024.10793304.
21. Yin H. Android malware detection using convolutional neural networks and light gradient boosting machine: a hybrid method. In: 2024 6th International Conference on Internet of Things, Automation and Artificial Intelligence (IoTAAI); 2024 Jul 26–28; Guangzhou, China. p. 75–9. doi:10.1109/IoTAAI62601.2024.10692620.
22. Reggi Tresna Utami M, Hilman MH, Yazid S. Enhancing phishing detection: integrating XGBoost with feature selection techniques. [cited 2025 Jan 1]. Available from: https://ssrn.com/abstract=5087049.
23. Liu Y, Tantithamthavorn C, Li L, Liu Y. Explainable AI for Android malware detection: towards understanding why the models perform so well? In: 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE); 2022 Oct 31–Nov 3; Charlotte, NC, USA. p. 169–80. doi:10.1109/ISSRE55969.2022.00026.
24. Baghirov E. A comprehensive investigation into robust malware detection with explainable AI. Cyber Secur Appl. 2025;3(1):100072. doi:10.1016/j.csa.2024.100072.
25. Elnaggar R, Servadei L, Mathur S, Wille R, Ecker W, Chakrabarty K. Accurate and robust malware detection: running XGBoost on runtime data from performance counters. IEEE Trans Comput Aided Des Integr Circuits Syst. 2022;41(7):2066–79. doi:10.1109/TCAD.2021.3102007.
26. Yang Y, Wu H, Wang Y, Wang P. AMN: attention-based multimodal network for Android malware classification. In: 2024 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE International Conference on Robotics, Automation and Mechatronics (RAM); 2024 Aug 8–11; Hangzhou, China. p. 7–13. doi:10.1109/CIS-RAM61939.2024.10672730.
27. Abbas MT, Khan MA, Khaliq A, Saqib NA, Ahmad J, Rehman S. Secure AODV protocol for mobile networks using short digital signatures. In: 2017 International Conference on Computational Science and Computational Intelligence (CSCI); 2017 Dec 14–16; Las Vegas, NV, USA. p. 645–50. doi:10.1109/CSCI.2017.111.
28. Doris A. AKANDO: a hybrid approach for effective Android botnet detection. Innov Int Multidiscip J Appl Technol. 2024;2(5):123–35.
29. Shu L, Dong S, Su H, Huang J. Android malware detection methods based on convolutional neural network: a survey. IEEE Trans Emerg Top Comput Intell. 2023;7(5):1330–50. doi:10.1109/TETCI.2023.3281833.
30. Asmitha KA, Vinod P, Rafidha Rehiman KA, Raveendran N, Conti M. Android malware defense through a hybrid multi-modal approach. J Netw Comput Appl. 2025;233(3):104035. doi:10.1016/j.jnca.2024.104035.
31. Kaleem M, Mushtaq MA, Jamil U, Ramay SA, Khan TA, Patel S, et al. New efficient cryptographic techniques for cloud computing security. Migr Lett. 2024;21(S11):13–28.
32. Taheri R, Shojafar M, Arabikhan F, Gegov A. Unveiling vulnerabilities in deep learning-based malware detection: differential privacy driven adversarial attacks. Comput Secur. 2024;146(2s):104035. doi:10.1016/j.cose.2024.104035.
33. Zaidi AR, Abbas T, Ramay SA, Islam IU, Irfan M. Hybrid permission-based Android malware detection using deep learning-enhanced CNNs and XGBoost. Migr Lett. 2024;21(S14):647–60.
34. Mehta S, Gadhavi LJ. Anticipating threats through malware detection approaches to safeguard data privacy and security: an in-depth study. In: 2024 3rd International Conference for Innovation in Technology (INOCON); 2024 Mar 1–3; Bangalore, India. p. 1–8. doi:10.1109/INOCON60754.2024.10511971.
35. Nguyen PS, Huy TN, Tuan TA, Trung PD, Long HV. Hybrid feature extraction and integrated deep learning for cloud-based malware detection. Comput Secur. 2025;150:104233. doi:10.1016/j.cose.2024.104233.
36. Bashir N, Mir AA, Daud A, Rafique M, Bukhari A. Time series reconstruction with feature-driven imputation: a comparison of base learning algorithms. IEEE Access. 2024;12(2):85511–30. doi:10.1109/access.2024.3416321.
37. Dahiya A, Singh S, Shrivastava G. Android malware analysis and detection: a systematic review. Expert Syst. 2025;42(1):e13488. doi:10.1111/exsy.13488.
38. Nowroozi E, Taheri R, Hajizadeh M, Bauschert T. Verifying the robustness of machine learning based intrusion detection against adversarial perturbation. In: 2024 IEEE International Conference on Cyber Security and Resilience (CSR); 2024 Sep 2–4; London, UK. p. 9–15. doi:10.1109/CSR61664.2024.10679401.

39. Fu X, Jiang C, Li C, Li J, Zhu X, Li F. A hybrid approach for Android malware detection using improved multi-scale convolutional neural networks and residual networks. Expert Syst Appl. 2024;249(2):123675. doi:10.1016/j.eswa.2024.123675.
40. Alsharif E, Alharby M. An ensemble machine learning approach for detecting and classifying malware attacks on mobile devices. Arab J Sci Eng. 2025;2025(5):1–17. doi:10.1007/s13369-025-10011-5.
41. Yılmaz EK, Bakır R. Advanced Android malware detection: merging deep learning and XGBoost techniques. Bilişim Teknol Derg. 2025;18(1):45–61.
42. Mbungang BN, Ali Wacka JB, Tchakounte F, Polatidis N, Nlong JM, Tieudjo D. Detecting Android malware with convolutional neural networks and Hilbert space-filling curves. SN Comput Sci. 2024;5(7):810. doi:10.1007/s42979-024-03123-6.
43. Arp D, Spreitzenbarth M, Hübner M, Gascon H, Rieck K. Drebin: effective and explainable detection of Android malware in your pocket. In: NDSS Symposium 2014; 2014; San Diego, CA, USA.