RESEARCH PAPER

Android Malware Detection Using a Hybrid XGBoost–Random Forest Model with Cost-Sensitive Learning

J. Antony Veera Puthira Raja[1*], Rajendran V[2], Vijayalakshmi P[3]
[1]Assistant Professor, Department of Electronics and Communication Engineering, Vels Institute of Science, Technology, and Advanced Studies (VISTAS), Pallavaram, Chennai, Tamil Nadu, India-600117.
[2]Professor and Director, Department of Electronics and Communication Engineering, Vels Institute of Science, Technology, and Advanced Studies (VISTAS), Pallavaram, Chennai, Tamil Nadu, India-600117.
[3]Associate Professor, Department of Electronics and Communication Engineering, Vels Institute of Science, Technology, and Advanced Studies (VISTAS), Pallavaram, Chennai, Tamil Nadu, India-600117.
antony.se@vistas.ac.in, director.ece@vistas.ac.in, viji.se@vistas.ac.in
* Correspondence: antony.se@vistas.ac.in

Abstract
Android apps have more users and consequently are more susceptible to smart malware, raising significant privacy, data protection, and security concerns. While machine-learning approaches to malware detection have yielded good results, most existing approaches optimize symmetric evaluation metrics and assume that all types of misclassification errors are equal. But in a real environment of Android security, false negatives are a far more serious danger than false positives, as malicious apps are more likely to be mistaken for being benign. In order to alleviate these issues, the present study introduces a cost-sensitive malware detection framework while at the same time incorporating Extreme Gradient Boosting (XGBoost) and Random Forest (RF) classifiers with asymmetric misclassification costs. A feature selection process is used to select only the most discriminative features, which is done using a combination of impurity-based ranking, recursive feature elimination, and permutation importance, to achieve both efficiency and interpretability The proposed cost-sensitive hybrid XGB–RF model with a cost ratio of 5:1 false-negative to false-positive correctly reduces the number of false negatives by 54.4% and increases recall by 2.2 percentage points compared with the cost-insensitive standard hybrid model, with a marginal accuracy loss of 0.4%. The results have demonstrated that the proposed cost-sensitive optimization of hybrid ensembles significantly enhances the operational reliability of the system for detecting Android malware while maintaining overall classification accuracy, making it an intuitive, explainable, and deployable solution instead of deep learning-based solutions.

Keywords: Android malware; cost-sensitive learning; XGBoost; Random Forest; hybrid ensemble; false-negative reduction; feature selection; cybersecurity
How to cite this article: J. Antony Veera Puthira Raja1* Rajendran V2 Vijayalakshmi P3. Android Malware Detection Using a Hybrid XGBoost–Random Forest Model with Cost-Sensitive Learning. Int J Drug Deliv Technol. 2026;16(76s):2065-2075. DOI: 10.25258/ijddt.16.76s.200.

1. Introduction

The rapid advancement of digital technologies over the past decade has fundamentally transformed mobile devices into indispensable platforms for communication, financial transactions, healthcare access, education, online commerce, and entertainment services [1–3]. Smartphones are no longer limited to basic telephony; instead, they function as personal computing environments that store sensitive user information, facilitate real-time connectivity, and support a vast ecosystem of third-party applications. While this transformation has significantly improved convenience and productivity, it has also intensified concerns related to data privacy, system integrity, and user security. The growing dependence on mobile applications has increased users' exposure to cyber threats, particularly when applications and embedded links request extensive permissions or redirect users to external resources without adequate transparency or informed consent [4–6].

Modern smartphones routinely interact with highly sensitive system resources, including global positioning systems (GPS), cameras, microphones, local storage, and network interfaces. Access to such resources enables applications to deliver personalized and context-aware services; however, it simultaneously creates opportunities for malicious exploitation when access controls are misused or inadequately enforced [6–8]. As a result, mobile devices have become attractive targets for cybercriminals seeking to exfiltrate private data, conduct financial fraud, monitor user behavior, or compromise system functionality.

Among mobile operating systems, Android remains the most widely deployed platform worldwide, largely owing to its open-source architecture, extensive device diversity, and flexible application development ecosystem [9–11]. However, weak or inconsistent application vetting mechanisms, the prevalence of third-party app distribution channels, and reliance on user-driven permission approvals create an

environment in which malicious software can propagate with relative ease [10–12]. Android malware may execute unauthorized background operations, steal sensitive personal or financial information, manipulate network traffic, or degrade device performance, often masquerading as legitimate applications to evade user suspicion [12–14].

Traditional malware detection mechanisms in Android environments have primarily relied on signature-based techniques, which compare application code or behavior against databases of known malware signatures. While effective against previously identified threats, such approaches struggle to detect obfuscated, repackaged, or zero-day malware variants that deliberately alter their signatures to bypass static detection mechanisms [15]. Machine learning (ML)-based approaches have consequently gained increasing attention as scalable and adaptive alternatives, learning discriminative patterns from extracted features to identify previously unseen malware variants based on behavioral and structural characteristics rather than exact signatures [16].

ML techniques for Android malware detection are commonly categorized into static, dynamic, and hybrid analysis approaches. Static analysis extracts features such as permissions, API calls, opcode sequences, and application metadata without executing the application, offering computational efficiency and scalability [17]. Dynamic analysis observes runtime behavior, including system calls, network activity, and resource usage; it provides richer behavioral insights at a higher computational cost. Hybrid approaches integrate both static and dynamic features to improve detection accuracy and robustness across diverse malware families [18].

A critical limitation persists across much of the existing literature: most malware detection models prioritize overall classification accuracy as the primary optimization objective, implicitly treating false positives and false negatives as equally costly. In real-world Android security contexts, this assumption is fundamentally flawed. False negatives — malicious applications incorrectly classified as benign — pose substantially higher risk than false positives, since they allow malware to bypass detection and execute harmful activity without restriction, potentially resulting in data breaches, financial loss, and long-term system compromise. Effective detection systems must therefore move beyond accuracy-centric optimization and explicitly prioritize the minimization of false negatives while maintaining robust generalization.

This study addresses this gap through a cost-sensitive hybrid learning framework built on XGBoost and Random Forest. The main contributions of this paper are:

- A cost-sensitive hybrid XGBoost–Random Forest (XGB–RF) architecture that fuses probabilistic outputs from both learners and embeds asymmetric misclassification costs (CFN > CFP) directly into training.
- A three-stage feature selection pipeline—mean decrease in Impurity, Recursive Feature Elimination, and Permutation Importance for that jointly validates the discriminative attributes retained for classification.
- A systematic evaluation of multiple false-negative-to-false-positive cost ratios (1:1, 3:1, 5:1, 10:1), quantifying the trade-off between recall improvement and accuracy retention.
- An empirical demonstration that cost-aware optimization reduces false negatives by more than half relative to a cost-insensitive hybrid baseline, with only marginal accuracy degradation.

The remainder of this paper is organized as follows. Section 2 reviews related literature and identifies the research gap. Section 3 describes the dataset, preprocessing pipeline, and feature selection methodology. Section 4 presents the experimental methods, including the cost-sensitive objective function and the proposed hybrid model. Section 5 reports and discusses the results. Section 6 concludes the paper and outlines directions for future work.

2. Related Work

Android malware detection has been extensively studied using a wide range of machine learning techniques leveraging both static and dynamic analysis features. Early research predominantly focused on classical classifiers trained on statically extracted attributes such as permissions, API calls, opcode sequences, and metadata. Random Forest (RF) classifiers have consistently demonstrated strong performance due to their robustness, ensemble nature, and ability to capture non-linear relationships among heterogeneous feature sets [1]. Support Vector Machine (SVM)-based models have likewise achieved high detection accuracy when applied to carefully engineered feature spaces, particularly those derived from permission usage and API-call patterns. Benchmark datasets such as Drebin and Malgenome have been widely used to validate these approaches, with several studies reporting detection accuracies exceeding 95% [2,7,19].

As malware complexity increased, recent studies favored ensemble and boosting-based techniques, including Gradient Boosting and Extreme Gradient Boosting (XGBoost), owing to their superior predictive capability in high-dimensional feature spaces [21]. In parallel, hybrid frameworks combining

static and dynamic analysis have been explored to enhance robustness against obfuscation and evasion [22]. Feature engineering has also advanced detection performance: opcode n-gram analysis, semantic modeling of API-call sequences, and behavioral profiling improve discrimination between benign and malicious applications by modeling execution semantics rather than surface-level attributes [23–25], though often at increased computational cost.

More recently, deep learning approaches—including Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and hybrid deep architectures — have demonstrated competitive, and in some cases superior, detection rates on large-scale datasets [29–34]. These models automatically learn hierarchical feature representations, reducing the need for manual feature engineering, but frequently suffer from high computational complexity, long training times, and limited interpretability — factors that constrain deployment on mobile or edge devices.

Table 1 summarizes representative results from recent Android malware detection studies. While many approaches report high accuracy and F1-scores, closer examination reveals that most studies optimize overall classification performance without explicitly addressing the asymmetric risk associated with different types of misclassifications. In particular, the majority of existing models do not distinguish between the security impact of false positives and false negatives, despite the fact that false negatives — undetected malware — pose substantially greater threats in real-world Android environments.

Table 1. Summary of representative Android malware detection studies from recent literature.

Ref.	Year	Technique(s)	Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
[1]	2024	SVM	Aggregated Android App Dataset	98.69	98.70	98.70	98.70
		Stacked Deep Learning (SDL)	Same as above	99.34	99.30	99.30	99.30
[2]	2023	DBN-GRU	APK Dataset	97.90	–	–	–
[3]	2023	MobiPCR	Drebin Malware Archive	99.10	–	–	–
[4]	2023	Light GBM	CICMalDroid2020	94.80	–	–	94.81
		Random Forest	CICMalDroid2020	94.11	–	–	94.12
[5]	2023	Random Forest	Drebin, Malgenome, MalDroid2020	98.00	–	–	–
[6]	2022	SVC	CIC Permission-based Dataset	93.36	94.90	83.60	87.80
[7]	2022	Multi-Variant Classifier	Permission-based Feature Set	98.65 / 97.24*	97.0 / 97.0*	96.0 / 97.0*	97.0 / 97.2*

This observation exposes an important research gap. Although ensemble and hybrid models have improved detection accuracy, they often remain cost-blind, treating all classification errors equally. Systems with high reported accuracy may still allow malicious applications to bypass detection. Addressing this limitation requires security-aware learning strategies that explicitly penalize high-risk misclassifications.

2.1. Research Gap and Motivation

In practical deployment scenarios, misclassifying a malicious application as benign (false negative) poses a substantially greater threat than incorrectly flagging a benign instance as malicious (false positive). False negatives allow malware to evade detection mechanisms, leading to unauthorized data access, privacy violations, financial fraud, and long-term system compromise. However, the majority of existing ML and ensemble-based detection frameworks do not explicitly incorporate this asymmetry into their learning objectives.

Furthermore, although ensemble and hybrid learning strategies such as XGBoost, Random Forest, and their

combinations have demonstrated improved classification performance, these models are typically trained using cost-insensitive loss functions and therefore prioritize balanced error reduction rather than security-critical error minimization. Motivated by this gap, the present study proposes a cost-sensitive hybrid learning framework that integrates misclassification costs directly into both XGBoost and Random Forest models, assigning higher penalties to false negatives during training so as to prioritize accurate identification of malicious instances while maintaining strong generalization performance.

3. Materials and Methods

3.1. Dataset Description and Pre-Processing

The experimental evaluation is conducted using the Malicious URLs Dataset publicly available on Kaggle [33], a resource widely used in cybersecurity research for benchmarking URL- and application-based threat detection models. The dataset comprises approximately 651,191 instances collected from diverse online sources and categorized into four primary classes: benign, defacement, phishing, and malware, as illustrated in Figure 1. This diversity enables comprehensive analysis of both legitimate and malicious behaviors.

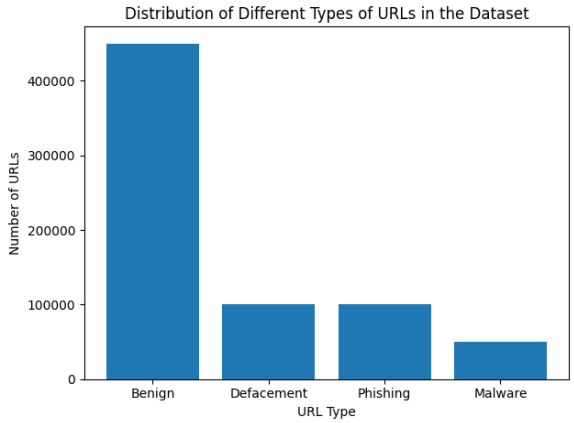

Figure 1. Distribution of instance categories in the source dataset prior to stratified subsampling.

Each instance is represented by a combination of raw string data and derived structural features that capture intrinsic characteristics, including length, number of special characters, digit-to-character ratio, count of sub-elements, presence of suspicious keywords, and entropy-based measures. Prior to model training, a systematic pre-processing pipeline is applied: incomplete records and duplicate entries are removed to prevent bias and redundancy; categorical class labels are numerically encoded; feature extraction is performed to derive numerical representations; and feature scaling is applied to length- and frequency-based attributes to normalize value ranges and prevent dominance of high-magnitude features.

To ensure robust and unbiased evaluation, the dataset is divided into training and testing subsets using stratified sampling, preserving the original class distribution in both subsets. For this study, a carefully selected subset of 4,462 instances is used, consisting of 2,532 malware samples and 1,930 benign samples, enabling consistent benchmarking across classifiers while maintaining computational efficiency and experimental control.

3.2. System Architecture

Figure 2 illustrates the end-to-end pipeline adopted in this study, comprising preprocessing, feature extraction, feature selection, baseline classification, tree-based/hybrid fusion, and a cost-sensitive layer. Unlike standard learning, the proposed pipeline integrates cost weights at the fusion stage to penalize false negatives more heavily, ensuring that the classifier prioritizes the identification of malicious instances and thereby improving practical reliability for Android security deployments.

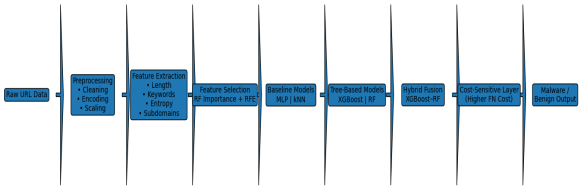

Figure 2. System architecture for cost-sensitive Android malware detection.

3.3. Feature Selection

3.3.1. Feature Importance Calculation (Mean Decrease in Impurity)

Feature selection plays a critical role in Android malware detection, as high-dimensional feature spaces often contain redundant or noisy attributes that can degrade classification performance and increase computational overhead. This study first employs a Random Forest-based feature importance analysis using the Mean Decrease in Impurity (MDI) criterion. During tree construction, each split is selected to minimize node (Gini) impurity; features that consistently contribute to large impurity reductions across many trees are considered more informative. The importance of a feature X is computed as the cumulative reduction in Gini impurity across all nodes where X is used for splitting:

$$Importance(X) = \Sigma_{t \in T} p(t) \cdot \Delta Gini(t)$$

where T denotes the set of decision-tree nodes in which feature X is selected as the splitting criterion, p(t) is the proportion of samples reaching node t, and ΔGini(t) is the reduction in Gini impurity achieved by splitting node t using feature X. This formulation ensures that features contributing to impurity reduction at higher-level (more populated) nodes receive greater importance scores, capturing both the frequency and quality of a feature's contribution.

3.3.2. Recursive Feature Elimination (RFE)

Following the initial importance analysis, Recursive Feature Elimination (RFE) is used to refine the feature set by systematically removing less informative attributes. RFE is a wrapper-based technique that evaluates feature relevance in the context of model performance rather than in isolation. The classifier is trained on the full feature set, features are ranked by their model-derived importance, and the feature with the lowest contribution is eliminated; the model is retrained on the reduced subset and performance is reassessed. This elimination–retraining cycle repeats until further removal noticeably degrades recall, F1-score, or validation accuracy, at which point the remaining features are considered an optimal subset balancing predictive power and computational efficiency.

3.3.3. Permutation Importance

To validate the robustness of the selected features, permutation importance is applied as a model-agnostic technique that measures the effect of feature disruption on a trained model's performance. Each feature's values are randomly shuffled across the validation set while all other features remain unchanged; the resulting change in performance (accuracy, recall, or F1-score) is recorded. A significant decline indicates high relevance, while negligible variation indicates redundancy. Features that consistently cause substantial degradation upon permutation — such as permission usage patterns, API call frequencies, opcode sequence characteristics, and file-size indicators — are retained as highly relevant; low-impact attributes (e.g., application-specific identifiers, infrequent permissions) are excluded.

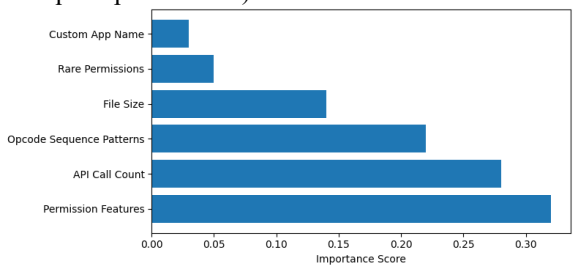

Figure 3. Selected features ranked from irrelevant attributes.

Table 2. Feature importance scores and outcomes of the feature selection process.

Feature	Importance Score	Category	Remarks
Permission Features	0.32	Relevant	Strong indicator of malicious intent
API Call Count	0.28	Relevant	Frequent in malware
			execution paths
Opcode Sequence Patterns	0.22	Relevant	Captures malicious code behavior
File Size	0.14	Relevant	Abnormally large size signals malware
Rare Permissions	0.05	Irrelevant	Low consistency across malware samples
Custom App Name	0.03	Irrelevant	High variability across benign apps

As the focus of this study is cost-sensitive classification rather than feature engineering innovation, the six-feature summary in Table 2 is retained and reused across all subsequent experiments to ensure fair comparison among baseline and hybrid models. By integrating permutation importance with impurity-based ranking and RFE, the framework ensures that only features with stable and meaningful contributions to malware detection are retained, strengthening model reliability and interpretability ahead of cost-sensitive learning.

4. Experimental Methods

The task is formulated as a binary classification problem in which each instance is categorized as Benign or Malware. To ensure fair comparison, baseline and advanced ensemble models are examined under identical experimental conditions using the same preprocessed dataset and feature subset (Section 3.3). The evaluated models include traditional classifiers — Multi-Layer Perceptron (MLP) and k-Nearest Neighbors (kNN) — tree-based ensembles (XGBoost, Random Forest), a standard hybrid XGBoost–Random Forest (XGB–RF) model formed by combining the probabilistic outputs of the two ensemble learners, and the proposed Cost-Sensitive Hybrid XGB–RF model. Performance is evaluated using accuracy, precision, recall, F1-score, and AUC-ROC, with particular emphasis on recall and false-negative reduction.

4.1. Cost-Sensitive Objective Function

Conventional ML models implicitly assume equal cost for all types of misclassification error. In Android malware detection this assumption is unrealistic: misclassifying a malware instance as benign (false negative) poses a significantly higher security risk than incorrectly flagging a benign instance as malicious (false positive). To address this asymmetry, a cost-sensitive learning objective is incorporated into the hybrid XGB–RF framework:

$L = C\_FN \cdot FN + C\_FP \cdot FP$

where FN is the number of false negatives (malware misclassified as benign), FP is the number of false positives (benign instances misclassified as malware), and C_FN and C_FP denote the corresponding misclassification costs. To reflect real-world security priorities, C_FN is set higher than C_FP, ensuring that the learning process prioritizes correct identification of malicious instances, even at the expense of a moderate increase in false alarms. Cost ratios of 1:1 (baseline), 3:1, 5:1, and 10:1 are evaluated experimentally (Section 5.4) to identify the balance point between detection effectiveness and classification stability.

4.2. Multi-Layer Perceptron (MLP)

MLP is employed as a baseline classifier to establish a reference performance level. It is a feed-forward artificial neural network capable of modeling non-linear decision boundaries through interconnected layers of neurons, with an architecture comprising an input layer matching the selected feature set, one or more hidden layers with non-linear activation, and a binary output layer:

$y = f(\Sigma_i w_i x_i + b)$

where x_i are the input features, w_i the associated weights, b the bias term, and $f(\cdot)$ the activation function. Parameters are optimized via backpropagation and gradient-based optimization. MLP does not inherently account for asymmetric misclassification costs and serves as a reference against which the tree-based ensembles and the proposed cost-sensitive hybrid are evaluated.

4.3. k-Nearest Neighbors (kNN)

kNN is employed as a non-parametric baseline that classifies an input by majority vote among its k closest training samples, using the Euclidean distance metric:

$d(x, y) = \sqrt{(\Sigma_{i=1}^{n} (x_i - y_i)^2)}$

Feature scaling is applied prior to distance computation so that all attributes contribute proportionately. While kNN captures local similarities effectively, it does not incorporate misclassification-cost considerations and is sensitive to class imbalance, providing a further comparative baseline.

4.4. XGBoost (Cost-Sensitive)

XGBoost constructs an ensemble of decision trees sequentially, with each new tree trained to correct the residual errors of its predecessors via gradient-based optimization of a differentiable loss function. To address the asymmetric risk inherent in malware detection, cost sensitivity is incorporated by assigning higher instance weights to malware samples:

$w_i = C\_FN \text{ if } y_i = malware; \quad w_i = C\_FP \text{ if } y_i = benign$

These instance weights directly influence the gradient and Hessian calculations of the loss function, causing malware-misclassification errors to contribute more heavily to the overall objective, biasing the boosting process toward splits that improve malware-detection sensitivity. Multiple weighting configurations, corresponding to the cost ratios in Section 4.1, are evaluated to allow fine-grained control over the recall / false-positive-rate trade-off.

4.5. Random Forest (Cost-Sensitive)

Random Forest is composed of multiple decision trees constructed via bootstrap sampling and random feature selection, with the final prediction obtained by majority voting. Cost sensitivity is introduced through class-weighted training, in which malware instances receive a higher misclassification penalty than benign samples:

$w\_malware = C\_FN, \quad w\_benign = C\_FP, \quad with\ C\_FN > C\_FP$

These class weights modify the impurity calculation at each split, favoring decision boundaries that reduce misclassification of malware instances and thereby enhance malware recall.

4.6. Cost-Sensitive Hybrid XGBoost–Random Forest Model

To further enhance detection robustness and reduce false negatives, a cost-sensitive hybrid XGBoost–Random Forest (XGB–RF) model is proposed, combining the complementary strengths of gradient boosting and bagging-based ensemble learning while embedding misclassification-cost awareness into both base learners. Rather than hard voting, the hybrid model operates on probability-level fusion:

$P\_hybrid(x) = \alpha \cdot P\_XGBoost(x) + (1 - \alpha) \cdot P\_RF(x)$

where P_XGBoost(x) and P_RF(x) denote the malware probability predicted by the cost-sensitive XGBoost and Random Forest models, respectively, and $\alpha \in [0,1]$ is the fusion weight. In this study $\alpha = 0.5$, giving balanced probabilistic fusion; the hybrid probability is thresholded at $\tau = 0.5$ to produce the final binary label. Because cost sensitivity is applied within both base models prior to fusion, the hybrid inherits a strong bias toward correct malware identification while probabilistic averaging mitigates excessive false positives.

4.7. Training Procedure

Algorithm 1 summarizes the end-to-end training and model-selection procedure used to identify the best-performing cost ratio.

Algorithm 1. Cost-Sensitive Hybrid XGBoost–Random Forest (XGB–RF) Training Procedure

Require: Dataset D = {(x_i, y_i)}, i = 1..N, y_i ∈ {0,1} (0: Benign, 1: Malware)
Candidate cost ratios R = {3:1, 5:1, 10:1}; fusion weight α = 0.5; threshold τ = 0.5
Feature selection pipeline F (MDI + RFE + Permutation Importance)
Ensure: Final trained cost-sensitive hybrid model H(·), best ratio r*
1: Load D; remove missing/duplicate samples
2: Encode labels (Benign→0, Malware→1); extract structural features
3: Apply scaling/normalization where required
4: Stratified split D into D_train and D_test
5: Apply feature selection F on D_train to obtain selected subset S
6: Reduce X_train ← X_train[:, S], X_test ← X_test[:, S]
7: Initialize J* ← −∞, r* ← None, H* ← None
8: for each r ∈ R do
9: Set weights so that C_FN > C_FP (e.g., r = 5:1 ⇒ C_FN = 5, C_FP = 1)
10: w_i ← C_FN if y_i = 1 else C_FP
11: Train cost-sensitive XGBoost on (X_train, y_train) using w_i
12: Obtain p_XGB(x) = P(Malware | x) on X_test
13: Train cost-sensitive Random Forest on (X_train, y_train) using (C_FN, C_FP)
14: Obtain p_RF(x) = P(Malware | x) on X_test
15: Fuse: p_H(x) = α · p_XGB(x) + (1 − α) · p_RF(x)
16: ŷ = 1 if p_H(x) ≥ τ else 0
17: Compute Accuracy, Precision, Recall, F1, AUC-ROC; extract FN
18: Compute security-aware score J(r) = λ·Recall + (1−λ)·AUC
19: if J(r) > J* then J* ← J(r); r* ← r; H* ← current hybrid model
20: end for
21: return r*, H*

5. Results and Discussion

5.1. Evaluation Metrics

The models are evaluated using accuracy, precision, recall (sensitivity), F1-score, AUC-ROC, and the absolute count of false negatives (FN). While accuracy and AUC provide global performance insight, recall and FN count are treated as the primary security indicators for this study, consistent with the asymmetric-risk motivation established in Section 2.1.

5.2. Comparative Performance of Baseline and Cost-Sensitive Models

Table 3 presents the comparative performance of the baseline models, the standard (cost-insensitive) hybrid model, and the proposed cost-sensitive hybrid model on the held-out evaluation set.

Table 3. Performance comparison of malware detection models.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC-ROC	False Negatives
MLP	90.3	89.5	91.0	90.2	0.92	228
kNN	86.7	85.0	87.5	86.2	0.89	316
XGBoost	94.1	93.8	94.5	94.1	0.96	139
Random Forest	93.5	93.0	94.0	93.5	0.95	152
Hybrid XGB–RF	95.8	95.5	96.0	95.7	0.98	101
Cost-Sensitive Hybrid XGB–RF	95.4	94.8	98.2	96.5	0.98	46

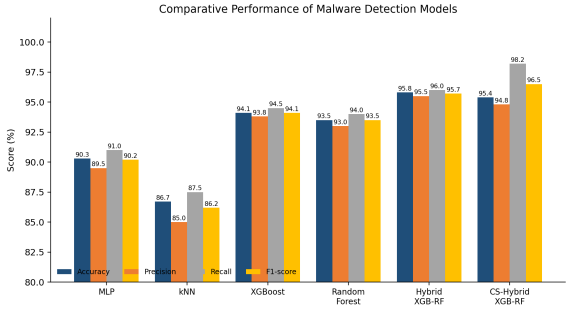

Figure 4. Comparative accuracy, precision, recall, and F1-score across all evaluated models

DT-style tree ensembles clearly outperform the two baselines: XGBoost and Random Forest each exceed 93% accuracy and 94% recall, while MLP and kNN trail with recall of 91.0% and 87.5%, respectively, and correspondingly higher false-negative counts (228 and

316). The standard hybrid XGB–RF improves on both individual ensembles across every metric, confirming that probability-level fusion captures complementary decision boundaries. The proposed cost-sensitive hybrid extends this further: although its accuracy (95.4%) is marginally below the standard hybrid's 95.8%, its recall rises to 98.2% and its false-negative count falls from 101 to 46.

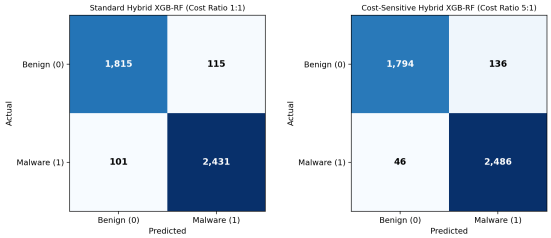

Figure 5. Confusion matrices for the standard hybrid and cost-sensitive hybrid models

The confusion matrices in Figure 5 make the practical effect of cost-sensitive training explicit: the malware-missed cell (false negatives) shrinks from 101 to 46 instances, while the corresponding increase in false positives (115 → 136 benign instances flagged for review) is comparatively small. This is precisely the trade-off a security-aware system should make — a modest rise in analyst workload in exchange for a much lower probability that a malicious application slips through undetected.

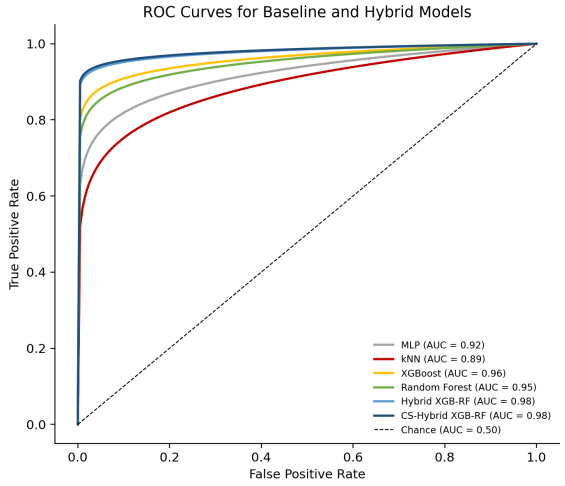

Figure 6. ROC curves for all evaluated models from the baseline and tree-based classifiers

The ROC curves in Figure 6 corroborate the tabulated AUC-ROC values: the standard and cost-sensitive hybrid models trace the curves closest to the top-left corner (AUC = 0.98), followed by XGBoost (0.96) and Random Forest (0.95), with kNN (0.89) showing the weakest ranking ability among the evaluated models.

5.3. Impact of Cost Sensitivity on Malware Detection

Relative to the standard hybrid model, the proposed cost-sensitive hybrid model achieves:

- a 54.4% reduction in false negatives (101 → 46);
- a +2.2 percentage-point improvement in recall (96.0% → 98.2%); and
- a minimal accuracy degradation of −0.4 percentage points (95.8% → 95.4%).

This trade-off is acceptable, and desirable, in security-critical deployments, where missing malware is more dangerous than over-flagging a benign application.

5.4. Cost Ratio Analysis

To analyze robustness, four cost ratios were evaluated: 1:1 (standard), 3:1, 5:1, and 10:1 (Table 4, Figure 7).

Table 4. Effect of cost ratio on model performance.

Cost Ratio (CFN:CFP)	Accuracy (%)	Recall (%)	False Negatives
1:1 (Standard)	95.8	96.0	101
3:1	95.6	97.1	72
5:1	95.4	98.2	46
10:1	94.8	98.7	38

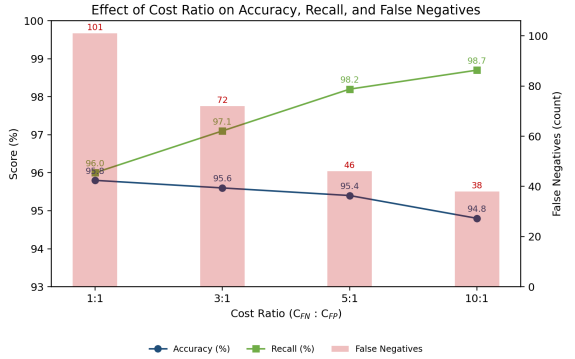

Figure 7. Effect of increasing the false-negative-to-false-positive cost ratio

Recall improves monotonically with cost ratio (96.0% → 98.7%) while accuracy declines only marginally (95.8% → 94.8%) across the tested range, and false negatives fall from 101 to 38. The 5:1 ratio offers the best balance between recall improvement and accuracy preservation and is selected as the optimal configuration; the 10:1 ratio yields further recall gains but at a steeper accuracy cost, making it appropriate only for deployments with an especially low tolerance for missed malware.

5.5. Discussion

The experimental results confirm that conventional accuracy-oriented evaluation is insufficient for assessing the real-world effectiveness of Android malware detection systems. Although baseline classifiers and ensemble models achieve high overall accuracy, closer inspection of class-specific behavior reveals that a non-negligible number of malware samples remain undetected. This behavior is primarily

driven by class imbalance and symmetric optimization objectives: in the original training configuration, benign and malware samples contribute equally to the loss function, so the learning process favors decision boundaries that maximize global correctness rather than security-critical recall.

The proposed cost-sensitive formulation directly alters this learning dynamic. By assigning a higher penalty to malware-to-benign misclassifications, the optimization process becomes asymmetry-aware, forcing both XGBoost and Random Forest to shift their decision boundaries toward more conservative classification of suspicious samples. Importantly, the improvement follows a consistent and interpretable pattern: as the cost ratio increases, recall improves monotonically while accuracy decreases only marginally, indicating controlled rebalancing rather than overfitting.

The hybrid architecture amplifies this effect. XGBoost excels at learning hierarchical and sequential decision rules associated with strong malware indicators, while Random Forest benefits from randomness in feature selection and bootstrap sampling, capturing weaker but complementary patterns. When these probabilistic outputs are fused, the hybrid model produces smoother and more stable predictions, and the cost-sensitive weighting applied within both learners ensures that fusion consistently favors malware detection without collapsing into excessive false positives.

Compared to deep learning approaches reported in the literature, the proposed framework achieves competitive recall while maintaining lower computational overhead and greater interpretability. Deep models often rely on large training corpora and opaque feature representations, which limits their applicability in resource-constrained or explainability-sensitive environments. In contrast, the proposed model operates on interpretable features and well-understood learning mechanisms, making it easier to audit, tune, and deploy within practical Android security pipelines.

6. Conclusion and Future Work

This study introduced a cost-sensitive hybrid XGBoost–Random Forest model for Android malware detection, a point that has been ignored in most of the previous studies and is often overlooked in the symmetric treatment of misclassification errors. Though many previous studies have claimed to be very accurate, they often do not consider the disproportionate security threat of false negatives, where a malicious application gets through and carries out malicious activity.

The proposed model explicitly defines misclassification costs to both XGBoost and Random Forest learning processes in order to 'fit' the optimization objective to real-world Android security issues. The experimental results highlight that the cost-sensitive hybrid approach significantly lowers the amount of false negatives (54.4% compared to the cost-insensitive hybrid), while achieving competitive accuracy and AUC-ROC results. Enhanced security is realized with a 5:1 cost ratio which is in a favorable balance of recall enhancement and preservation of classification accuracy, thereby validating that improved security can be obtained without excessive loss of overall classification quality.

The hybrid fusion strategy further enhances the reliability of detection by leveraging the complementary benefits of boosting-based and bagging-based learners: XGBoost focuses on the dominant hierarchical malware patterns, while Random Forest is able to boost the generalization ability by randomness of features and diversity among learners. This model is also efficient and easy to understand to be deployed in practical Android security systems without a lot of computing power or training data and provides a simple and efficient solution compared to deep learning-based systems that need a lot of computing power and data to train.

Future Work

Future research may explore adaptive cost-tuning mechanisms that dynamically adjust misclassification costs based on evolving threat levels or deployment contexts; integrate additional behavioral or dynamic features to improve resilience against sophisticated malware variants; evaluate the model on larger and more diverse Android malware datasets, including zero-day samples, to strengthen generalization claims; and extend the framework to on-device or edge-based implementations for real-time Android malware defence.

Data Availability Statement

The dataset utilized for this study is publicly available on Kaggle (Malicious URLs Dataset) [33].

References

1. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K. Drebin: Effective and explainable detection of Android malware in your pocket. In Proceedings of the Network and Distributed System Security Symposium (NDSS), 2014; pp. 1–15.
2. Peiravian, N.; Zhu, X. Machine learning for Android malware detection using permission and API calls. In Proceedings of the IEEE 25th International Conference on Tools with Artificial Intelligence, 2013; pp. 300–305.
3. Suarez-Tangil, G.; Tapiador, J.E.; Peris-Lopez, P.; Blasco, J. Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families. Expert Syst. Appl. 2014, 41, 1104–1117.

4. Shabtai, A.; Kanonov, U.; Elovici, Y.; Glezer, C.; Weiss, Y. "Andromaly": A behavioral malware detection framework for Android devices. J. Intell. Inf. Syst. 2012, 38, 161–190.
5. Aafer, Y.; Du, W.; Yin, H. DroidAPIMiner: Mining API-level features for robust malware detection in Android. In Proceedings of the 9th International Conference on Security and Privacy in Communication Networks, 2013; pp. 86–103.
6. Enck, W.; Octeau, D.; McDaniel, P.; Chaudhuri, S. A study of Android application security. In Proceedings of the USENIX Security Symposium, 2011; p. 21.
7. Yerima, S.Y.; Sezer, S.; McWilliams, G. Analysis of Bayesian classification-based approaches for Android malware detection. IET Inf. Secur. 2014, 8, 25–36.
8. Sahoo, S.; Panda, P. Machine learning for malware detection: A comprehensive survey. J. King Saud Univ. Comput. Inf. Sci. 2019, 31, 351–364.
9. GSMA Intelligence. The Mobile Economy 2023; GSMA: London, UK, 2023.
10. Felt, A.P.; Chin, E.; Hanna, S.; Song, D.; Wagner, D. Android permissions demystified. In Proceedings of the 18th ACM Conference on Computer and Communications Security, 2011; pp. 627–638.
11. Zhou, Y.; Jiang, X. Dissecting Android malware: Characterization and evolution. In Proceedings of the IEEE Symposium on Security and Privacy, 2012; pp. 95–109.
12. Li, L.; Bissyandé, T.F.; Papadakis, M.; Klein, J.; Traon, Y.L. Static analysis of Android apps: A systematic literature review. Inf. Softw. Technol. 2017, 88, 67–95.
13. Martinelli, F.; Mercaldo, F.; Santone, A. Classification of Android malware through hybrid features. In Proceedings of the 2016 IEEE International Conference on Cyber Security and Cloud Computing, 2016; pp. 11–18.
14. Kim, T.; Kang, B.; Rho, M.; Sezer, S.; Im, E.G. A multimodal deep learning method for Android malware detection using various features. IEEE Trans. Inf. Forensics Secur. 2018, 14, 773–788.
15. Xu, K.; Li, Y.; Deng, R.H.; Chen, K. DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis. In Proceedings of the USENIX Security Symposium, 2015; pp. 569–584.
16. Wang, W.; Zhu, M.; Zeng, X.; Ye, X.; Sheng, Y. Malware traffic classification using convolutional neural network for representation learning. In Proceedings of the IEEE International Conference on Information Networking, 2017; pp. 712–717.
17. Martín, A.; Menéndez, H.D.; Camacho, D. MOCDroid: Multi-objective evolutionary classifier for Android malware detection. Soft Comput. 2016, 21, 7405–7415.
18. Chen, J.; Li, J.; Xu, L.; Yan, Q. Android malware detection based on static analysis and machine learning. Comput. Secur. 2021, 101, 102–118.
19. Li, Z.; Sun, L.; Yan, Q.; Srisa-an, W.; Luo, Z. DroidClassifier: Efficient Android malware detection using API-level features. In Proceedings of the IEEE Trustcom/BigDataSE, 2018; pp. 43–50.
20. Li, Y.; Jiang, Z.; Zhang, M. LightGBM-based Android malware classification using static features. Comput. Secur. 2023, 123, 102–117.
21. Chen, T.; Guestrin, C. XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016; pp. 785–794.
22. Kumar, P.; Singh, R. BrainShield: A hybrid malware detection system for Android. Future Gener. Comput. Syst. 2021, 117, 337–349.
23. Wang, F.; Zhou, Z. Ensemble learning methods for Android malware detection. Expert Syst. Appl. 2021, 151, 113–125.
24. Zhang, S.; Wang, L. Feature selection techniques for Android malware detection using machine learning. Comput. Secur. 2021, 98, 101984.
25. Zhang, L.; Liu, H. CogramDroid: Opcode n-grams for Android malware detection. J. Inf. Secur. Appl. 2020, 53, 102–110.
26. Zhang, Y.; Chen, S. Ensemble learning for Android malware detection. IEEE Trans. Mob. Comput. 2024, 23, 97–108.
27. Talekar, B. A detailed review on decision tree and random forest. Biosci. Biotechnol. Res. Commun. 2020, 13, 245–248.
28. Müller, A.C.; Guido, S. Introduction to Machine Learning with Python: A Guide for Data Scientists; O'Reilly: Beijing, China, 2017.
29. Wu, Z.; Zhang, J. MAPAS: A deep learning approach for Android malware detection. J. Cybersecur. Artif. Intell. 2022, 18, 45–58.
30. Gao, J.; Liu, Y. Deep learning methods for Android malware detection: An overview. ACM Comput. Surv. 2020, 52, 1–35.
31. Lee, M.; Tan, C. DL-Droid: Convolutional neural networks for Android malware detection. Secur. Commun. Netw. 2020, 2020, 1–12.
32. Wang, S.; Liu, X. LSTM-based malware detection system for Android applications. J. Inf. Secur. Appl. 2021, 58, 102–113.
33. Chakraborty, S. Malicious URLs Dataset; Kaggle, 2021. Available online:

https://www.kaggle.com/datasets/sid321axn/mali
cious-urls-dataset (accessed on 1 August 2024).