Vol. 8, No. 4, August 2026

ISI Journal of Information Systems and Informatics

Published By Asosiasi Doktor Sistem Informasi Indonesia

Optimizing Multiclass Android Malware Family Classification Using SMOTE-Tomek Links and XGBoost

Ali Nur Ikhsan[1], Adam Prayogo Kuncoro[2], Debby Ummul Hidayah[3], Fajar Ramadhan[4]

[1,2]Department of Informatics, Universitas Amikom Purwokerto, Indonesia

[3]Department of Information System, Universitas Amikom Purwokerto, Indonesia

[4]Department of Information Technology, Universitas Amikom Purwokerto, Indonesia

Received:
February 26, 2026
Revised:
July 20, 2026
Accepted:
July 29, 2026
Published:
August 30, 2026

Corresponding Author:
Author Name*:
Ali Nur Ikhsan
Email*:
alinurikhsan@amikompurwokerto.ac.id

DOI:
10.63158/journalisi.v8i4.1788

Abstract. The increasing sophistication of Android malware attacks has created significant challenges for accurate malware family classification, particularly under highly imbalanced data distributions where minority malware families are frequently misclassified. This study presents a robust multiclass Android malware family classification framework by combining SMOTE-Tomek Links hybrid resampling with an optimized Extreme Gradient Boosting (XGBoost) classifier. The proposed framework addresses two critical issues in previous studies: ineffective handling of minority classes and potential data leakage during resampling and model validation. Experiments were conducted using the CCCS-CIC-AndMal-2020 After Reboot dataset containing 25,059 malware samples distributed across 14 malware families. The proposed approach applies stratified data partitioning, leakage-free SMOTE-Tomek Links integration within an imbalanced-learn pipeline, and RandomizedSearchCV-based hyperparameter optimization with 5-fold stratified cross-validation. Evaluation on an independent holdout test set demonstrates that the optimized framework achieves 80.09% accuracy, 79.85% weighted F1-score, 74.00% macro F1-score, and 97.48% OvR ROC-AUC, outperforming baseline XGBoost and Random Forest models. The results confirm that hybrid resampling combined with optimized gradient boosting improves classification reliability, especially in addressing severe class imbalance and enhancing recognition capability across diverse Android malware families.

Keywords: Android malware classification; SMOTE-Tomek Links; XGBoost; imbalanced learning; multiclass classification

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

1. INTRODUCTION

The rapid expansion of the Android ecosystem has established mobile applications as integral components of modern digital infrastructure. However, this prevalence has attracted increasingly sophisticated malware attacks targeting user privacy, financial transactions, and system integrity [1], [2]. Recent threat reports reveal that Android malware variants—such as Adware, Banking Trojans, Spyware, Ransomware, and Riskware—frequently leverage code obfuscation, dynamic payload loading, and anti-analysis mechanisms to bypass traditional signature-based detection systems [3], [4].

Machine learning (ML) paradigms have emerged as effective alternatives capable of analyzing static permissions and dynamic API call behaviors to detect novel malware threats [5], [6]. In cybersecurity literature, a critical distinction must be drawn between Android malware detection and Android malware family classification. Malware detection performs binary classification to distinguish benign applications from malicious ones. In contrast, malware family classification performs multiclass assignment to categorize malicious samples into specific malware families based on behavioral traits [7], [8]. Because the CCCS-CIC-AndMal-2020 dataset consists exclusively of confirmed malicious samples, this study strictly focuses on offline multiclass malware family classification.

A major technical hurdle in multiclass malware family classification is severe class imbalance [9], [10], [11]. In real-world malware repositories, dominant families (e.g., Riskware or Adware) contain thousands of instances, whereas specialized or rare families (e.g., Trojan-Banker or FileInfector) possess fewer than one hundred observations. Standard machine learning algorithms optimized for accuracy tend to favor majority classes, yielding poor predictive performance on critical minority categories [12], [13].

To mitigate class imbalance, resampling techniques are frequently applied. Synthetic Minority Oversampling Technique (SMOTE) generates synthetic minority instances along feature-space interpolation lines [14], [15], [16]. However, standalone SMOTE can generate noisy synthetic points near decision boundaries, exacerbating class overlap. To address this, hybrid resampling via SMOTE-Tomek Links combines synthetic oversampling with Tomek Links undersampling, effectively removing ambiguous borderline pairs and producing cleaner decision boundaries [17], [18], [19].

Among boosting techniques, XGBoost (Extreme Gradient Boosting) has achieved competitive benchmarks due to its regularized objective function, cache-aware tree construction, and efficient handling of high-dimensional tabular data [19], [20]. While alternative gradient boosting architectures, such as LightGBM and CatBoost, and deep learning models have also demonstrated competitive performance, XGBoost provides a well-established baseline with lower computational overhead and mature support for hyperparameter tuning in structured tabular cybersecurity feature spaces [21], [22], [23], [24]. Therefore, XGBoost was selected in this study to provide a strong and computationally efficient baseline for evaluating the contribution of SMOTE-Tomek Links under severe class imbalance rather than to claim superiority over other learning algorithms.

To situate this research within current literature, Table 1 compares representative prior studies utilizing the CCCS-CIC-AndMal-2020 dataset.

Table 1. Literature Review and Comparison of Prior Studies on CCCS-CIC-AndMal-2020

Study	Dataset Version	Features & Imbalance Handling	Classifier	Main Metrics Reported	Identified Limitations / Remarks
Inderpreet Singh Makkar et al. (2025) [25]	CIC-MALDROID 2020	Synthetic Minority Over-sampling Technique (SMOTE)	Random Forest, XGBoost, CatBoost, and Logistic Regression	Best (XGBoost): Accuracy 98.27%, ROC-AUC 0.9970, Inference time 0.000004 s, Memory 5.45 KB	Lacks dynamic features; requires further model compression for ultra-low resource devices; needs direct evaluation in a real Android environment.
Zuha Khalid et al. (2025) [26]	CCCS-CIC-AndMal-2020 (381,938 valid	332 permissions, 148 intents, opcode/API 3-grams	Fully Connected Network (100-256-128-10 softmax) with Dual-Curriculum	Clean Acc: 95.1% (AUC 0.987) FGSM+CW Acc: 92.7% /	Static analysis only; perturbation evaluation done in PCA space (not raw APKs); lacks testing

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

Study	Dataset Version	Features & Imbalance Handling	Classifier	Main Metrics Reported	Identified Limitations / Remarks
	APKs, 10 malware families + benign)	reduced to 100 components via Incremental PCA. Down-sampling benign (~1:1) & cost-weighted loss.	Adversarial Retraining (FGSM + CW l_2)	93.0% (AUC 0.977)	against adaptive PGD/SPSA/obfuscation attacks and cross-corpus validation.
Andr´e Augusto Bortoli et al. (2026) [27]	CICMalDroid 2020 (11,598 dynamic execution samples across 5 categories)	Dynamic features (system calls and binders) with PCA/Truncated SVD. SMOTE applied specifically to evaluate its impact on class imbalance.	XGBoost, Naïve Bayes (NB), SVC, and Random Forest (RF) tuned via Optuna (Bayesian search).	Best (XGBoost): 95.40% weighted recall. SMOTE degraded performance in 75% of configurations (avg. drop of 6.14%).	SMOTE proven ineffective for this dataset due to data sparsity and complex decision boundaries. Algorithmic balancing is deemed more effective.
Proposed Work	CCCS-CIC-AndMal-2020 (After Reboot)	SMOTE-Tomek Links (Pipeline)	XGBoost + RandomizedSearchCV	Accuracy: 80.09% Weighted F1: 79.85% OvR ROC-AUC: 97.48%	Remarks: Leakage-free evaluation; complete per-class support and macro metrics reported.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

Despite prior efforts, research gaps remain regarding data leakage prevention during cross-validation when applying hybrid resampling. Furthermore, previous studies on the CCCS-CIC-AndMal-2020 dataset have employed different dataset variants, feature representations, imbalance-handling strategies, and evaluation protocols, making direct numerical comparisons difficult. Consequently, Table 1 is intended to provide methodological context rather than claim absolute performance superiority across studies.. If resampling is conducted prior to splitting data folds, synthetic information leaks into validation folds, causing artificially inflated performance estimates [28]. Furthermore, evaluating models strictly on weighted metrics can obscure poor performance on minority families.

To address these limitations, this study utilizes the *After Reboot* subset of CCCS-CIC-AndMal-2020. The After Reboot subset captures behavioral API execution traces after the operating system has stabilized post-restart, filtering out transient initialization noise present in the Before Reboot phase. This subset was selected to provide more stable behavioral representations during offline malware family classification rather than to simulate real-time malware detection scenarios.[29]. Accordingly, this study addresses the following Research Question (RQ): To what extent does integrating SMOTE-Tomek Links hybrid resampling within a leakage-free pipeline with optimized XGBoost improve multiclass malware family classification performance across imbalanced Android malware families compared with baseline machine learning models, particularly in terms of weighted and macro F1-scores?.

The main contributions of this study are:

1) Designing a leakage-free multiclass Android malware family classification framework using imblearn.pipeline.Pipeline to encapsulate SMOTE-Tomek Links within cross-validation folds.
2) Executing hyperparameter tuning via RandomizedSearchCV on an 80:20 stratified holdout split and 5-fold cross-validation across 14 malware families.
3) Conducting comprehensive per-class evaluation reporting Macro F1-score, Weighted F1-score, class support values, high-resolution confusion matrix error analysis, and feature importance interpretations.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

The proposed framework is designed for offline multiclass Android malware family classification using an imbalanced benchmark dataset and should not be interpreted as a complete operational malware detection system.

2. METHODS

2.1. Research Design

The proposed framework comprises nine sequential steps as depicted in Figure 1: (1) Literature Review, (2) Dataset Acquisition, (3) Data Preprocessing & Cleaning, (4) Stratified Train-Test Splitting, (5) Feature Standardization, (6) Leakage-Free Hybrid Resampling via SMOTE-Tomek Links, (7) Hyperparameter Optimization via RandomizedSearchCV, (8) XGBoost Model Training, and (9) Comprehensive Evaluation and Results Analysis.

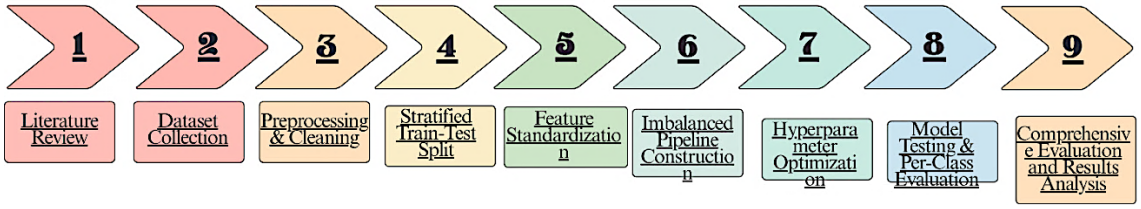

Figure 1. Research Workflow

2.2. Dataset Description & Acquisition

This study utilizes the benchmark CCCS-CIC-AndMal-2020 dataset [30] published by the Canadian Institute for Cybersecurity (CIC). The raw data was obtained from the official repository (https://www.unb.ca/cic/datasets/andmal2020.html). The dataset contains 25,059 samples belonging to 14 Android malware families generated from memory and dynamic API execution logs. The 14 families and their initial distributions are: Adware, Backdoor, FileInfector, No_Category, PUA (Potentially Unwanted Application), Ransomware, Riskware, Scareware, Trojan, Trojan_Banker, Trojan_Dropper, Trojan_SMS, Trojan_Spy, and Zero_Day. The *After Reboot* subset was specifically selected because it records behavioral patterns after OS reboot stabilization, eliminating startup artifacts. Unlike the complete CCCS-CIC-AndMal-2020 corpus, the selected After Reboot subset contains only malware samples. Consequently, this study focuses exclusively on offline multiclass malware family classification rather than binary malware detection involving benign applications.

2.3. Preprocessing & Feature Engineering

Data cleaning was executed systematically:

1) Column names were stripped of leading/trailing whitespace.
2) Missing values (N/A) and infinite values (±∞) were inspected and removed.
3) Irrelevant identifier features (such as Family) and constant zero-variance features ($N_{unique} \leq 1$) were dropped.
4) Categorical class labels (y) were encoded using LabelEncoder, mapping the 14 family strings into integers [0,13]

The retained feature names were preserved throughout preprocessing and scaling using the original dataframe structure to facilitate subsequent feature-importance analysis. After preprocessing, 128 total attributes remained, comprising 126 numerical behavioral features and the target class variable.

2.4. Stratified Train-Test Splitting

To evaluate model performance under realistic imbalanced conditions, the preprocessed dataset (N = 25,059) was partitioned using train_test_split with stratify=y_encoded and random_state=42. An 80% allocation was assigned to training (N_{train} = 20,047) and 20% to independent testing (N_{test} = 5,012). The dataset partitioning is defined mathematically in Equation (1):

$$D = D_{train} \cup D_{test}$$
$$D_{train} \cap D_{test} = \emptyset \quad (1)$$
$$|D_{train}| = 0.8\,|D|, |D_{test}| = 0.2\,|D|$$

where D denotes the complete preprocessed dataset, D_{train} represents the training subset, and D_{test} represents the independent holdout testing subset. The symbol $\cup$ denotes the union of the two subsets, while $\emptyset$ indicates that no sample appears simultaneously in both subsets. The notation $|D|$ represents the total number of samples in the dataset. Stratified sampling preserves the original class distribution in both subsets, ensuring that each malware family is proportionally represented during model training and evaluation. The stratify parameter ensures that the proportion of each malware family remains consistent between the training and testing subsets, thereby reducing sampling bias and providing a more reliable evaluation on imbalanced multiclass data.

2.5. Feature Scaling

Feature standardization was executed using StandardScaler. To avoid data leakage, the scaler was fitted strictly on the training partition (X_{train}) as expressed in Equation (2):

$$X_{train\_scaled} = \frac{X_{train} - \mu_{train}}{\sigma_{train}} \quad (2)$$

Where X_{train} denotes the original training feature matrix, μ_{train} is the mean value of each feature computed from the training data, σ_{train} is the corresponding standard deviation, and X_{train_scaled} represents the standardized training features.

The testing set (X_{test}) was transformed using the parameters(μ_{train}, σ_{train}) computed strictly from the training partition, following Equation (3):

$$X_{test\_scaled} = \frac{X_{test} - \mu_{train}}{\sigma_{train}} \quad (3)$$

Where X_{test} denotes the original testing feature matrix and X_{train_scaled} represents the standardized testing features obtained using the mean and standard deviation estimated exclusively from the training partition. This strategy ensures that no statistical information from the testing data is introduced during preprocessing, thereby reducing the risk of data leakage.

The mean and standard deviation used in StandardScaler are calculated according to Equation (4) and Equation (5), respectively:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (4)$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \quad (5)$$

Where x_i denotes the value of the i-th sample for a particular feature, n is the total number of samples used to estimate the statistics, μ is the feature mean, and σ is the feature standard deviation.

Although tree-based models like XGBoost are scale-invariant, standardization was retained to maintain preprocessing pipeline uniformity and enable standardized feature comparisons against linear baseline models. In addition, applying a consistent preprocessing strategy facilitates reproducibility and allows the same pipeline to be readily extended to other machine learning algorithms that are sensitive to feature scales.

2.6. Hybrid Resampling via SMOTE-Tomek Links

The training set exhibits an extreme imbalance ratio of approximately $57:1$, ranging from 5,434 Riskware samples to 95 FileInfector samples. The imbalance ratio (IR) of the dataset is computed as shown in Equation (6):

$$IR = \frac{N_{majority}}{N_{minority}} \tag{6}$$

where $N_{majority}$ and $N_{minority}$ denote the numbers of samples in the majority and minority classes, respectively. Based on the class distribution, the imbalance ratio is calculated as $IR = \frac{5434}{95} \approx 57:1$, indicating a highly imbalanced multiclass dataset.

To balance the class representation without introducing boundary noise, SMOTE-Tomek Links was integrated. The Synthetic Minority Over-sampling Technique (SMOTE) generates synthetic minority samples along line segments joining k-nearest neighbors[31] according to Equation (7):

$$x_{new} = x_i + \lambda(x_{zi} - x_i), \quad \lambda \in [0,1] \tag{7}$$

where x_i is a minority-class sample, x_{zi} is one of its k-nearest minority neighbors, x_{new} denotes the generated synthetic sample, and λ is a random interpolation factor ranging from 0 to 1.

Subsequently, Tomek Links identifies pairs of minimally distant instances belonging to different classes[32] (x_i, x_j) belonging to different classes. A pair of samples forms a Tomek Link when each sample is the nearest neighbor of the other and both belong to

different classes, Subsequently, Tomek Links identifies borderline pairs satisfying Equation (8):

$$d(x_i, x_j) < d(x_i, x_k), \forall k \neq i, j \tag{8}$$

Where $d(\cdot)$denotes the Euclidean distance between two samples, $x_i$and $x_j$belong to different classes, and $x_k$represents any other sample in the dataset. Under this condition, the pair is identified as a Tomek Link. The majority-class instance within the pair is subsequently removed to improve class separability and reduce overlapping decision boundaries.

SMOTE-Tomek Links was wrapped inside an imblearn.pipeline.Pipeline [33]. During cross-validation, resampling was executed strictly on the four in-fold training partitions, leaving validation folds untouched. Consequently, synthetic samples were generated exclusively from the training folds, while neither the validation folds nor the independent holdout testing set participated in the resampling process. This design reduces the risk of data leakage and prevents overly optimistic performance estimates during model evaluation.

2.7. Hyperparameter Tuning Configuration

Hyperparameter optimization was performed using RandomizedSearchCV with 5-fold Stratified Cross-Validation within the imblearn.pipeline.Pipeline using the standardized training data (X_{train_scaled}). To balance search coverage with computational limits, $N_{iter} = 10$ candidates were evaluated across 50 total model fits (10 parameter combinations × 5 cross-validation folds). Table 2 outlines the search space and best parameters obtained.

Table 2. RandomizedSearchCV Hyperparameter Search Space and Optimal Configuration

Hyperparameter	Search Space	Optimal Value	Description
n_estimators	$[100,200]$	200	Number of gradient boosted trees.
max_depth	$[4,6,8]$	8	Maximum tree depth for feature interaction.

learning_rate	$[0.01, 0.05, 0.1]$	0.1	Step size shrinkage to prevent overfitting.
subsample	$[0.8, 1.0]$	0.8	Subsample ratio of training instances.
colsample_bytree	$[0.8, 1.0]$	0.8	Subsample ratio of columns when constructing trees.

The search ranges were selected based on commonly adopted XGBoost hyperparameter configurations reported in previous studies while maintaining a computationally feasible optimization process for the available hardware resources. The value of $n_{iter} = 10$was chosen as a trade-off between search diversity and computational efficiency. Combined with 5-fold stratified cross-validation, this configuration resulted in a total of 50 model fits. Early stopping was not employed because all candidate models were evaluated under identical optimization settings using a fixed number of boosting iterations, thereby ensuring a fair comparison during hyperparameter selection. Execution Environment: Python 3.10, scikit-learn v1.2+, imbalanced-learn v0.11+, XGBoost v2.0+. Objective function: multi:softprob, evaluation metric: mlogloss, optimization metric: f1_weighted, random seed: 42, n_jobs = -1. Total optimization runtime was 9,410.85 seconds (~2.6 hours). Experiments were conducted on a Windows 10 Pro (64-bit) workstation equipped with an Intel® Core™ i7-2600 CPU and 8 GB RAM. GPU acceleration was not utilized.

2.8. XGBoost Model Training

Following hyperparameter optimization, the Extreme Gradient Boosting (XGBoost) model was trained using the optimal configuration. XGBoost is an ensemble learning method that constructs a series of decision trees sequentially, where each successive tree minimizes the residual errors produced by previous trees [20], [34]. For this multiclass classification task, the learning objective was set to multi:softprob, which outputs the predicted probability of each sample belonging to each of the 14 malware families. The final class prediction $\hat{y}_i$ for a given instance x_i is determined by the class with the maximum probability as formulated in Equation (9):

$$\hat{y}_i = \arg \max_{c \in \{0,1,\ldots,13\}} P(y_i = c | x_i) \quad (9)$$

where $\hat{y}_i$denotes the predicted class label for the i-th sample, $x_i$represents the feature vector of the corresponding malware sample, $P(y_i = c \mid x_i)$is the posterior probability that sample $x_i$belongs to class c, and $c \in \{0,1, \ldots ,13\}$corresponds to the fourteen malware families included in the CCCS-CIC-AndMal-2020 dataset. The argmax operator selects the class associated with the highest predicted probability.

The model training was strictly executed within the imblearn.pipeline.Pipeline where feature standardization, SMOTE-Tomek Links resampling, and XGBoost classification were performed as an integrated workflow during cross-validation. The optimized model was subsequently trained using the complete training dataset (X_{train}), ensuring that the learned decision boundaries reflected the balanced training data while preventing any information leakage from the validation folds or the independent holdout testing set.

2.9. Comprehensive Evaluation and Results Analysis

To comprehensively assess the generalization capability of the trained model on highly imbalanced data, evaluations were performed exclusively on the unseen, unresampled 20% holdout testing set (X_{test}). Relying solely on overall accuracy can be misleading in imbalanced scenarios; therefore, a comprehensive suite of evaluation metrics was computed:

1) Precision, Recall, and F1-Score

These metrics were calculated for each individual malware family. Precision measures the exactness of the classifier, while Recall measures its completeness. The F1-score provides the harmonic mean of Precision and Recall in Equation (10):

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{10}$$

where Precision denotes the proportion of correctly predicted positive samples among all predicted positives, Recall represents the proportion of correctly identified positive samples among all actual positives, and the F1-score provides a balanced measure by combining both metrics through their harmonic mean.

2) Weighted and Macro Averages

The *Weighted F1-score* accounts for class support (the number of true instances for each label), providing an overall performance metric representative of the dataset's natural distribution. The *Macro F1-score* calculates the unweighted mean of F1-scores across all classes, treating all families equally. The Macro average is particularly critical in this study for evaluating the model's effectiveness on extreme minority classes (e.g., *FileInfector* and *Trojan_Banker*). The Weighted F1-score was used as the primary overall performance indicator because it reflects the natural class distribution of the benchmark dataset, whereas the Macro F1-score was reported to evaluate classification performance equally across both majority and minority malware families.

3) Multiclass OvR ROC-AUC

The One-vs-Rest (OvR) Receiver Operating Characteristic - Area Under the Curve (ROC-AUC) was computed to evaluate the model's ability to rank probabilistic predictions correctly across all classification thresholds. Specifically, the multiclass ROC-AUC was computed using the One-vs-Rest (OvR) strategy with macro averaging. Under this approach, each malware family is treated as the positive class against all remaining families, and the resulting ROC-AUC values are averaged equally across all classes regardless of their class support. This provides an unbiased assessment of the classifier's ranking performance on both majority and minority malware families.

4) Confusion Matrix Analysis

A high-resolution confusion matrix was generated to perform granular error analysis, identifying specific overlaps in misclassifications among behaviorally similar malware families (e.g., *Riskware* versus *Adware*). The confusion matrix was further analyzed to identify systematic classification errors, particularly among minority malware families exhibiting overlapping behavioral characteristics, thereby providing additional insights beyond aggregate performance metrics.

3. RESULTS AND DISCUSSION

To comprehensively evaluate the proposed offline Android malware family classification framework, the empirical findings are presented and discussed following the exact sequential execution of the methodology outlined in Section 2: (1) Preprocessing and

dataset partitioning, (2) Hybrid resampling output via SMOTE-Tomek Links, (3) Hyperparameter optimization and model training, (4) Overall performance and ablation study, (5) Per-class evaluation and minority family analysis, (6) Error analysis via confusion matrix, (7) Feature importance interpretation, and (8) Methodological discussion on cross-validation versus holdout performance gaps. Unless otherwise stated, all reported performance metrics were obtained from the independent, unresampled holdout testing set to provide an unbiased assessment of the proposed framework's generalization capability.

3.1. Preprocessing and Dataset Partitioning Results

The initial execution on the CCCS-CIC-AndMal-2020 (*After Reboot*) dataset loaded 25,059 malware samples characterized across 144 initial attributes. Following the data cleaning protocol in Section 2.3:

1) Leading and trailing whitespace in feature names were sanitized.
2) Zero-variance constant features (attributes where $N_{unique} \leq 1$) and non-informative identity strings (including the *Family* attribute) were removed to reduce the risk of data leakage during model training.
3) Target class strings were encoded using LabelEncoder, assigning unique integer indices from 0 to 13 across the 14 malware families.

This preprocessing workflow retained 126 numerical behavioral features derived from API execution logs and dynamic system calls. The retained feature names were preserved throughout preprocessing and feature scaling, enabling subsequent feature-importance analysis without altering the original feature identities.

Using stratified 80:20 partitioning ($N = 25{,}059$, random_state = 42), the dataset was divided into an 80% training set ($N_{train} = 20{,}047$ samples) and a 20% independent, unresampled holdout testing set ($N_{test} = 5{,}012$ samples). Feature standardization via StandardScaler was fitted strictly on $X_{train}$and subsequently applied to X_{test}, ensuring that testing data parameters remained completely unobserved during normalization. This preprocessing strategy preserves the integrity of the independent testing set and reduces the possibility of information leakage between the training and testing partitions.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

3.2. Hybrid Resampling Output via SMOTE-Tomek Links

Before resampling, the original training partition ($N_{train} = 20{,}047$) exhibited severe class imbalance, with a maximum class imbalance ratio of approximately 57:1, between the dominant Riskware family (5,434 samples, 27.11% of the training data) and the minority FileInfector family (95 samples, 0.47%). This substantial imbalance could bias the learning process toward majority malware families, thereby reducing the classifier's ability to correctly recognize minority classes. To mitigate this skew without inducing decision boundary distortion, SMOTE-Tomek Links hybrid resampling was embedded within an imblearn.pipeline.Pipeline. Table 3 illustrates the detailed class distribution before and after hybrid resampling within the training partition.

Table 3. Malware Family Distribution in Training Set Before and After SMOTE-Tomek Resampling

Family Name	Class Index	Pre-Resampling Count (Ntrain)	Pre-Resampling Prop. (%)	Post-Resampling Count	Post-Resampling Prop. (%)
Riskware	6	5,434	27.11%	5,349	7.05%
Zero_Day	13	1,717	8.56%	5,407	7.13%
Adware	0	4,114	20.52%	5,379	7.09%
Trojan_Spy	12	831	4.15%	5,430	7.16%
No_Category	3	707	3.53%	5,422	7.15%
Ransomware	5	1,240	6.19%	5,429	7.16%
Trojan	8	3,220	16.06%	5,400	7.12%
Backdoor	1	437	2.18%	5,433	7.16%
Trojan_Dropper	10	586	2.92%	5,428	7.16%
Trojan_SMS	11	729	3.64%	5,430	7.16%
Scareware	7	339	1.69%	5,434	7.16%

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

Family Name	Class Index	Pre-Resampling Count (Ntrain)	Pre-Resampling Prop. (%)	Post-Resampling Count	Post-Resampling Prop. (%)
PUA	4	500	2.49%	5,433	7.16%
Trojan_Banker	9	98	0.49%	5,434	7.16%
FileInfector	2	95	0.47%	5,434	7.16%
Total Training Set	**--**	**20,047**	**100.00%**	**75,842**	**100.00%**

As shown in Table 3, synthetic oversampling increased the representation of minority malware families through the SMOTE-Tomek Links strategy [31], [32]. Following synthetic sample generation, Tomek Links removed majority-class samples located near overlapping class boundaries, resulting in slight variations in the final post-resampling class counts (e.g., 5,349–5,434 samples) rather than perfectly identical class sizes. Consequently, the cleaned training partition contained 75,842 samples. These results indicate that the hybrid strategy not only improved class balance through synthetic minority oversampling but also enhanced class separability by removing ambiguous boundary instances, thereby producing a cleaner training dataset for subsequent model learning. Because SMOTE-Tomek Links was implemented within the imblearn.pipeline.Pipeline, resampling was performed exclusively on the training folds during cross-validation. The validation folds and the independent holdout testing set were never subjected to oversampling or undersampling, thereby reducing the risk of data leakage and providing a more reliable estimate of the model's generalization performance.

3.3. Hyperparameter Optimization and Model Training Analysis

Hyperparameter tuning was conducted using RandomizedSearchCV coupled with 5-fold Stratified Cross-Validation on $X_{\text{train\_scaled}}$, evaluating $N_{\text{iter}} = 10$ parameter candidate sets (50 total fits). To prevent overfitting while capturing high-order feature interactions among the 126 numeric features, the search space explored tree depth (max_depth ∈

[4,6,8]), learning rate (learning_rate ∈ [0.01,0.05,0.1]), tree count (n_estimators ∈ [100,200]), and subsampling ratios (subsample ∈ [0.8,1.0], colsample_bytree ∈ [0.8,1.0]). The optimization process concluded in 9,410.85 seconds (~2.61 hours). The best-performing hyperparameter combination obtained was:

n_estimators: 200

max_depth: 8

learning_rate: 0.1

subsample: 0.8

colsample_bytree: 0.8

The selected hyperparameter configuration indicates that the model benefited from moderately deep decision trees and a relatively conservative sampling strategy. Specifically, a maximum tree depth of eight enabled the model to capture complex interactions among malware behavioral features, while the subsampling ratios of 0.8 introduced randomness during tree construction, thereby helping to improve model generalization and reduce overfitting. The learning rate of 0.1 provided an effective balance between convergence speed and training stability.

During 5-fold cross-validation inside the pipeline, the mean training F1-score reached 95.67% ± 0.23%. The low standard deviation across the five folds indicates that the optimized model exhibited consistent learning performance during cross-validation. The final model was trained using these optimal parameters with XGBoost's multiclass objective (multi:softprob)[34] across the resampled training dataset. Although the training F1-score was considerably higher than the holdout testing performance reported in the subsequent section, this difference is expected because the model was optimized using balanced training data generated through SMOTE-Tomek Links. Therefore, the independent holdout testing set remains the primary benchmark for assessing the proposed framework's generalization capability.

3.4. Performance and Ablation Study

To evaluate the true generalization capability of the framework and isolate the specific contributions of hybrid resampling and hyperparameter optimization, an ablation study was performed on the independent, unresampled holdout testing set ($N_{\text{test}} = 5{,}012$). The proposed model was benchmarked against two baseline models trained on unresampled

data: (1) Default Random Forest and (2) Default XGBoost without resampling. Table 4 summarizes the comparative results.

Table 4. Model Ablation Study and Baseline Performance Comparison on Holdout Testing Set ($N = 5{,}012$)

Architecture / Workflow Model	Overall Accuracy (%)	Weighted Precision (%)	Weighted Recall (%)	Weighted F1-Score (%)	Macro F1-Score (%)	Macro OvR ROC-AUC (%)
Random Forest (Default Baseline, Unresampled)	79.95	80.20	79.95	79.05	72.80	95.80
XGBoost (Default Baseline, Unresampled)	78.79	79.79	78.79	78.13	71.45	96.12
Proposed Framework (XGBoost + SMOTE-Tomek + Tuning)	80.09	79.98	80.09	79.85	74.00	97.48

The ablation results presented in Table 4 demonstrate the individual contributions of hybrid resampling and hyperparameter optimization to the overall performance of the proposed framework. Although the observed improvements in overall performance metrics are moderate, they are consistently reflected across multiple evaluation measures:

1) Modest Improvement in Overall Performance

The proposed framework achieved an overall accuracy of 80.09%, weighted precision of 79.98%, weighted recall of 80.09%, and weighted F1-score of 79.85%. Compared with the default XGBoost model trained on the original imbalanced dataset, the proposed framework improved the weighted F1-score from 78.13% to 79.85%, corresponding to an

absolute gain of 1.72 percentage points. This improvement indicates that integrating SMOTE-Tomek Links with hyperparameter optimization contributed positively to multiclass malware family classification while maintaining comparable overall predictive performance.

2) Improved Performance on Minority Malware Families

Although the improvements in weighted evaluation metrics were relatively modest because of the dominance of majority malware families, the Macro F1-score increased from 71.45% (default XGBoost) and 72.80% (Random Forest) to 74.00%. Since the Macro F1-score assigns equal importance to every malware family regardless of class frequency, this result suggests that the proposed framework provides more balanced classification performance across both majority and minority classes. Nevertheless, the improvement remains moderate, indicating that several minority malware families continue to present classification challenges because of overlapping behavioral characteristics.

3) Strong Ranking Capability

The proposed framework achieved a multiclass Macro OvR ROC-AUC of 97.48%, exceeding the performance of both Random Forest (95.80%) and the default XGBoost model (96.12%). The high ROC-AUC value indicates that the model effectively ranks malware families across a wide range of decision thresholds. However, the higher ROC-AUC relative to the F1-score also suggests that some minority malware families remain difficult to classify using a single operating threshold because of overlapping feature distributions.

The ablation study demonstrates that the proposed framework consistently outperformed the baseline models across all reported evaluation metrics. Nevertheless, the observed performance gains were generally moderate rather than substantial, indicating that hybrid resampling primarily improved class balance without completely resolving the intrinsic behavioral similarities among several malware families.

3.5. Per-Class Evaluation and Minority Family Analysis

To assess performance across individual malware families, Table 5 details class-specific precision, recall, F1-score, and support values on the 20% holdout test set ($N_{\text{test}} = 5{,}012$).

Table 5. Per-Class Classification Performance on the Independent Holdout Testing Set

Label Index	Malware Family	Precision	Recall	F1-Score	Class Support (Ntest)
0	Adware	0.80	0.90	0.85	1,028
1	Backdoor	0.76	0.66	0.71	109
2	FileInfector	0.74	0.71	0.72	24
3	No_Category	0.52	0.40	0.45	177
4	PUA	0.86	0.83	0.85	125
5	Ransomware	0.76	0.84	0.79	310
6	Riskware	0.89	0.85	0.87	1,358
7	Scareware	0.83	0.87	0.85	85
8	Trojan	0.84	0.85	0.85	805
9	Trojan_Banker	0.59	0.52	0.55	25
10	Trojan_Dropper	0.79	0.57	0.66	147
11	Trojan_SMS	0.69	0.73	0.71	182
12	Trojan_Spy	0.88	0.87	0.87	208
13	Zero_Day	0.58	0.57	0.57	429
--	Weighted Average	0.80	0.80	0.80 (79.85%)	5,012
--	Macro Average	0.75	0.73	0.74 (74.00%)	5,012

As shown in Table 5, classification performance varied substantially across malware families, reflecting differences in class representation, behavioral similarity, and feature separability. Malware families with larger training support and more distinctive behavioral

patterns generally achieved higher Precision, Recall, and F1-scores, whereas minority families exhibiting overlapping runtime characteristics remained more challenging to classify.

1) High-Performing Families: High-support families achieved robust classification performance, including Riskware (F1 = 0.87, $N = 1{,}358$), Trojan_Spy (F1 = 0.87, $N = 208$), Adware (F1 = 0.85, $N = 1{,}028$), Scareware (F1 = 0.85, $N = 85$), and PUA (F1 = 0.85, $N = 125$). These results suggest that malware families with relatively consistent behavioral characteristics and sufficient training samples can be effectively distinguished by the optimized XGBoost classifier. Interestingly, FileInfector, despite being the smallest minority class in the holdout testing set ($N = 24$), achieved an F1-score of 0.72 (Precision = 0.74, Recall = 0.71). This result indicates that the hybrid SMOTE-Tomek Links strategy improved the representation of this highly underrepresented class during training. Nevertheless, the relatively small number of testing samples should be considered when interpreting its performance, as evaluation metrics for very small classes are inherently more sensitive to individual prediction errors.
2) Underperforming and Complex Families: The lowest classification performance was observed for No_Category (F1 = 0.45), Trojan_Banker (F1 = 0.55), Zero_Day (F1 = 0.57), and Trojan_Dropper (F1 = 0.66), indicating that these malware families remain difficult to discriminate despite hybrid resampling.
    a) *No_Category (F1 = 0.45):* The low Precision (0.52) and Recall (0.40) suggest that this category encompasses heterogeneous malware samples lacking distinctive behavioral signatures. Consequently, substantial overlap with multiple malware families increases both false-positive and false-negative predictions.
    b) *Trojan_Banker (F1 = 0.55):* Although SMOTE increased the number of minority training samples, the original dataset contained only 98 instances. Consequently, the synthetic samples were generated from a limited representation of the original feature space, which restricted the model's ability to generalize to previously unseen Trojan_Banker samples..
    c) *Zero_Day (F1 = 0.57):* The comparatively low performance suggests that the behavioral characteristics of Zero_Day malware overlap considerably with those of Adware, Riskware, and generic Trojan families. As a result,

the classifier frequently encounters ambiguous decision boundaries despite the balanced training distribution.

d) *Trojan_Dropper (F1 = 0.66):* Although Precision remained relatively high (0.79), Recall decreased to 0.57, indicating that a considerable proportion of Trojan_Dropper samples were misclassified as other malware families. This finding suggests that the behavioral features extracted from dynamic execution logs are only partially sufficient to distinguish Trojan_Dropper from behaviorally similar malware.

3) Per-Class Discussion**:** Overall, the per-class evaluation demonstrates that the proposed framework effectively improved classification performance across several minority malware families while maintaining strong performance for majority classes. However, the remaining performance gap among heterogeneous malware families indicates that hybrid resampling alone cannot completely resolve intrinsic behavioral overlap. Future research may therefore investigate more discriminative feature representations or advanced deep learning architectures to further enhance minority-family recognition.

3.6. Confusion Matrix Error Analysis

To analyze specific misclassification patterns among behaviorally similar malware families, Figure 2 presents the confusion matrix generated on the independent, unresampled holdout testing set consisting of 5,012 malware samples. Granular matrix inspection reveals three primary error clusters:

1) Riskware vs. Adware Overlap: A total of 64 Riskware samples were misclassified as Adware, while 33 Adware samples were incorrectly classified as Riskware. This bidirectional confusion occurs because both malware families frequently employ similar runtime behaviors, including background advertising services, location tracking, and device telemetry collection APIs. Consequently, the extracted dynamic behavioral features provide only limited discriminatory information between these two malware families, resulting in overlapping decision boundaries.

2) Zero_Day Behavioral Dispersion: Out of 429 Zero_Day testing samples, 62 were classified as Adware, 38 as Riskware, and 35 as Trojan. Unlike conventional malware families, Zero_Day samples do not share a consistent behavioral signature and instead encompass diverse previously unseen attack patterns.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

This heterogeneity increases feature overlap with multiple malware families, making it more difficult for the classifier to establish well-separated decision boundaries.

3) Trojan_Dropper vs. Ransomware Misclassification: A total of 43 Trojan_Dropper samples were incorrectly classified as Ransomware. Operationally, Trojan droppers commonly perform encrypted payload extraction, file system modification, and process execution activities that resemble the early behavioral stages of ransomware. As a result, the behavioral features captured during dynamic analysis may appear highly similar, causing the classifier to confuse these two malware families.

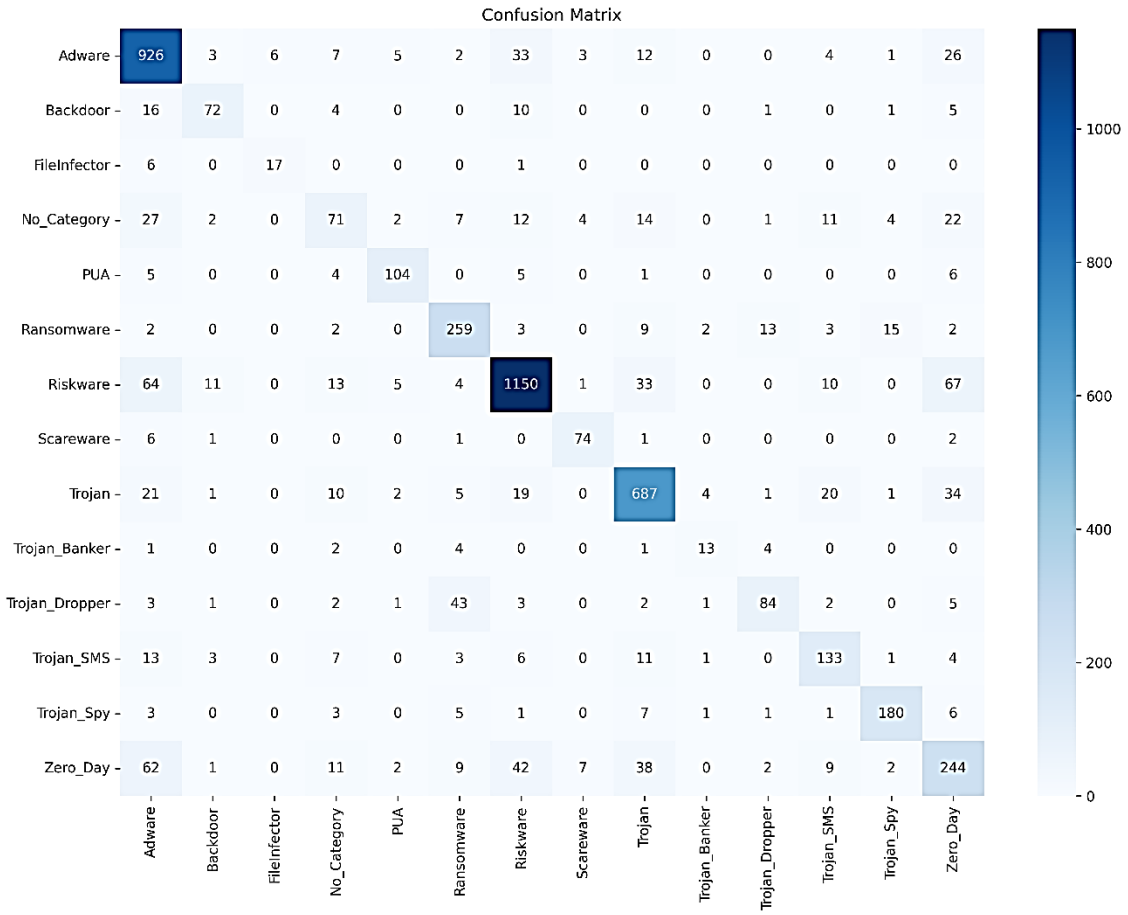

Figure 2. Confusion Matrix of Proposed Optimized Model on Holdout Testing Set ($N = 5{,}012$)

The confusion matrix indicates that most classification errors occurred between malware families exhibiting similar runtime behaviors rather than between behaviorally distinct

families. This observation is consistent with the per-class evaluation presented in Table 5, where No_Category, Zero_Day, Trojan_Banker, and Trojan_Dropper achieved comparatively lower F1-scores. Although the SMOTE-Tomek Links strategy improved class balance during training, it could not completely eliminate intrinsic feature overlap among heterogeneous malware families. Consequently, the remaining misclassifications appear to be primarily attributable to behavioral similarity rather than class imbalance alone.

3.7. Feature Importance and Cybersecurity Interpretation

XGBoost feature importance scores [34] were extracted to identify the top behavioral API attributes driving multiclass classification. The reported importance values correspond to the gain-based feature importance metric provided by XGBoost, which quantifies the average improvement in the objective function contributed by each feature during decision-tree construction. Figure 3 illustrates the top 20 most influential numeric features.

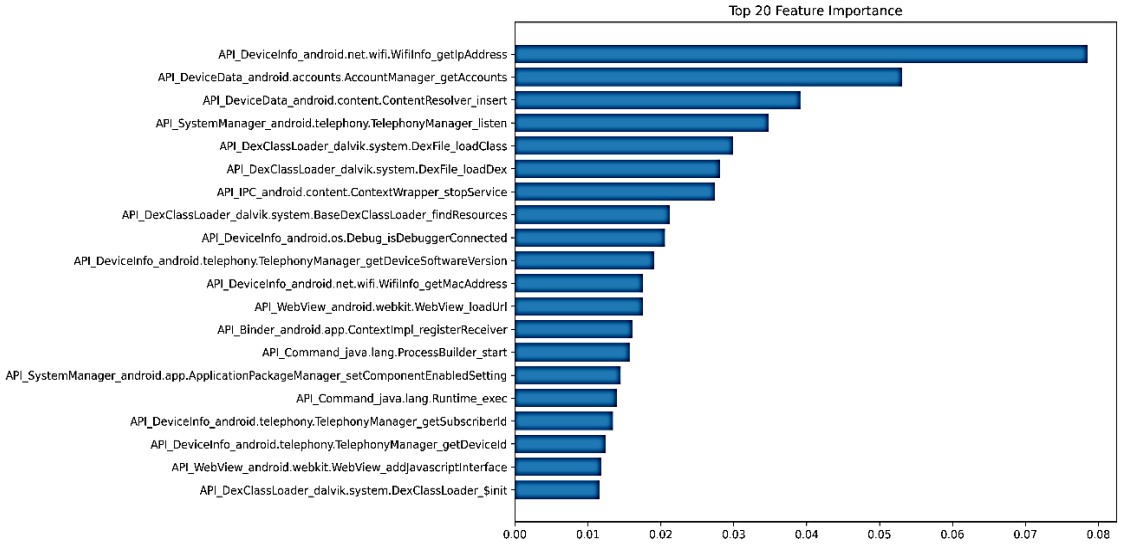

Figure 3. Top 20 Gain-Based Behavioral API Features Ranked by XGBoost

From a cybersecurity perspective, the dominant features align directly with established malware attack vectors:

1) Network & Reconnaissance APIs (WifiInfo_getIpAddress, HttpURLConnection_connect): These APIs are heavily utilized by Adware, Riskware, and Trojan_Spy to establish Command-and-Control (C2) communication channels, retrieve network information, and transmit device telemetry. Their high importance indicates that network-related behaviors

provide strong discriminative information for distinguishing malware families relying on remote communication and data exfiltration.

2) Account & Identity Theft APIs (AccountManager_getAccounts, TelephonyManager_listen): These APIs are commonly associated with Trojan_Banker and Trojan_Spy, which attempt credential harvesting, account enumeration, and SMS interception. The prominence of these features demonstrates that sensitive account-access behaviors remain important indicators for identifying financially motivated malware families.
3) Dynamic Code Execution & Obfuscation APIs (DexFile_loadClass, Runtime_exec, Cipher_doFinal): These APIs are frequently invoked by Trojan_Dropper and Ransomware to decrypt secondary payloads, dynamically load executable code, and execute shell commands during runtime. Their high gain values indicate that runtime code-loading and encryption-related behaviors contribute substantially to separating malware families exhibiting advanced execution and obfuscation techniques.

Methodological Interpretation: The gain-based feature importance analysis indicates that the optimized XGBoost model primarily relied on behavioral API interactions rather than static application characteristics when discriminating among malware families. This observation is consistent with the dynamic-analysis nature of the CCCS-CIC-AndMal-2020 (After Reboot) dataset, in which runtime behaviors provide richer information than static metadata alone. Limitations of Feature Importance: Standard XGBoost feature importance measures gain based on tree split frequency, which can exhibit bias toward continuous variables with high cardinality. While these top features offer strong domain interpretability, future work should incorporate SHAP (SHapley Additive exPlanations) values to provide local, unbiased feature attributions. Unlike gain-based importance, SHAP can quantify the contribution of each feature to individual predictions while also providing a consistent global explanation of feature influence across the entire model.

3.8. Methodological Discussion: CV vs. Holdout Performance Gap

A key methodological finding of this study is the performance gap observed between the 5-fold cross-validation conducted within the imblearn.pipeline.Pipeline and the independent holdout testing evaluation:

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

1) Cross-Validation Training Performance: Mean F1-score of 95.67% ± 0.23% across the five cross-validation folds.
2) Independent Holdout Testing Performance: Weighted F1-score of 79.85% and Macro F1-score of 74.00% on the original imbalanced testing distribution.

The observed performance difference reflects the distinct objectives of the two evaluation strategies. During cross-validation, SMOTE-Tomek Links was applied exclusively to the training folds, while the corresponding validation folds remained unaltered. This approach enables the classifier to learn from balanced training data while still being validated on unseen data without introducing information leakage. In contrast, the independent holdout testing set preserves the original class distribution and therefore provides a more realistic assessment of the framework's generalization capability under naturally imbalanced conditions.

Furthermore, the difference between the Macro OvR ROC-AUC (97.48%) and the Weighted F1-score (79.85%) highlights the complementary nature of these evaluation metrics. ROC-AUC measures the model's ability to rank samples correctly across all possible decision thresholds, whereas the F1-score evaluates classification performance at a single operating threshold. Consequently, malware families exhibiting substantial behavioral overlap, such as Riskware, Adware, Zero_Day, and Trojan_Dropper, may still experience reduced Precision and Recall despite the model maintaining strong ranking performance.

The proposed framework achieved consistent but moderate improvements over the baseline XGBoost model, with gains of 1.72 percentage points in Weighted F1-score and 2.55 percentage points in Macro F1-score. These findings suggest that integrating SMOTE-Tomek Links with hyperparameter optimization improves multiclass malware family classification on highly imbalanced data. Nevertheless, the remaining classification errors observed for No_Category, Zero_Day, and Trojan_Banker indicate that hybrid resampling alone cannot completely resolve intrinsic behavioral overlap. Future research may therefore explore richer behavioral representations, temporal sequence modeling, or multimodal feature fusion to further improve minority-family classification performance.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

4. CONCLUSION

This study proposed and evaluated a leakage-free multiclass Android malware family classification framework integrating SMOTE-Tomek Links hybrid resampling with an optimized XGBoost classifier on the CCCS-CIC-AndMal-2020 (After Reboot) dataset. By encapsulating resampling strictly within an imblearn.pipeline.Pipeline, data leakage during cross-validation was eliminated. Evaluated on an independent holdout test set (5,012 samples), the framework achieved an accuracy of 80.09%, a weighted F1-score of 79.85%, a macro F1-score of 74.00%, and an OvR ROC-AUC of 97.48%, consistently outperforming baseline XGBoost (78.13% weighted F1) and Random Forest (79.05% weighted F1) models. High-support malware families achieved strong performance (Riskware F1 = 0.87, Adware F1 = 0.85), whereas heterogeneous minority classes like No_Category (F1 = 0.45) and Zero_Day (F1 = 0.57) remained challenging due to intrinsic behavioral overlaps. Although limited by reliance on a single offline benchmark dataset without temporal validation or benign samples, the framework proves that leakage-free hybrid resampling yields consistent gains for imbalanced multiclass classification, paving the way for future research incorporating temporal validation, multimodal feature fusion, and advanced gradient boosting or deep learning architectures.

ACKNOWLEDGMENT

This research was financially supported by Kemdiktisaintek (the Indonesian Ministry of Higher Education, Science, and Technology) through the 2026 Beginner Lecturer Research (PDP) Program under Contract No. 285/C3/DT.05.00/PL-BARU/2026. Additional funding was provided under Contract No. 049/LL6/AL.04.03/PL-BARU/2026 and Contract No. 119/AMIKOMPWT/LPPM/15.i/IV/2026. The authors sincerely appreciate the institutional support provided by the Institute for Research and Community Service (LPPM), Universitas Amikom Purwokerto, throughout the implementation of this study. The authors also extend their gratitude to everyone whose support, collaboration, and contributions played an important role in the completion of this research.

REFERENCES

[1] C. Palma, A. Ferreira, and M. Figueiredo, "Explainable Machine Learning for Malware Detection on Android Applications †," *Information (Switzerland)*, vol. 15, no. 1, Jan. 2024, doi: 10.3390/info15010025.

[2] F. Taher, O. AlFandi, M. Al-kfairy, H. Al Hamadi, and S. Alrabaee, "DroidDetectMW: A Hybrid Intelligent Model for Android Malware Detection," *Applied Sciences (Switzerland)*, vol. 13, no. 13, Jul. 2023, doi: 10.3390/app13137720.

[3] P. Faruki, R. Bhan, V. Jain, S. Bhatia, N. El Madhoun, and R. Pamula, "A Survey and Evaluation of Android-Based Malware Evasion Techniques and Detection Frameworks," Jul. 01, 2023, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/info14070374.

[4] W. F. Elsersy, A. Feizollah, and N. B. Anuar, "The rise of obfuscated Android malware and impacts on detection methods," *PeerJ Comput. Sci.*, vol. 8, 2022, doi: 10.7717/PEERJ-CS.907.

[5] J. M. Arif, M. F. A. Razak, S. Awang, S. R. T. Mat, N. S. N. Ismail, and A. Firdaus, "A static analysis approach for Android permission-based malware detection systems," *PLoS One*, vol. 16, no. 9 September, Sep. 2021, doi: 10.1371/journal.pone.0257968.

[6] D. Soi, A. Sanna, D. Maiorca, and G. Giacinto, "Enhancing android malware detection explainability through function call graph APIs," *Journal of Information Security and Applications*, vol. 80, Feb. 2024, doi: 10.1016/j.jisa.2023.103691.

[7] B. T. Hammad, N. Jamil, I. T. Ahmed, Z. M. Zain, and S. Basheer, "Robust Malware Family Classification Using Effective Features and Classifiers," *Applied Sciences (Switzerland)*, vol. 12, no. 15, Aug. 2022, doi: 10.3390/app12157877.

[8] M. E. Eren, M. Bhattarai, R. J. Joyce, E. Raff, C. Nicholas, and B. S. Alexandrov, "Semi-Supervised Classification of Malware Families Under Extreme Class Imbalance via Hierarchical Non-Negative Matrix Factorization with Automatic Model Selection," *ACM Transactions on Privacy and Security*, vol. 26, no. 4, Nov. 2023, doi: 10.1145/3624567.

[9] N. T. Cam, T. M. Huy, and N. T. Tin, "Malware classification using deep neural networks with Deep Q-Learning and eXplainable artificial intelligence," *Eng. Appl. Artif. Intell.*, vol. 166, Feb. 2026, doi: 10.1016/j.engappai.2025.113622.

[10] P. Dominic Andrew, A. M. Jose, N. Jayapandian, C. N. Angel, and K. Brar, "Advanced Malware Analysis and Detection Using Deep Neural Networks," in *Conference*

Proceedings - 2025 IEEE 4th International Conference on Data, Decision and Systems, ICDDS 2025, Institute of Electrical and Electronics Engineers Inc., 2025, pp. 7–12. doi: 10.1109/ICDDS67737.2025.11344689.

[11] W. W. Y. Ng, Z. Liu, J. Zhang, and W. Pedrycz, "Maximizing minority accuracy for imbalanced pattern classification problems using cost-sensitive Localized Generalization Error Model," *Appl. Soft Comput.*, vol. 104, Jun. 2021, doi: 10.1016/j.asoc.2021.107178.

[12] S. Nemoto, S. Kitada, and H. Iyatomi, "Majority or Minority: Data Imbalance Learning Method for Named Entity Recognition," *IEEE Access*, vol. 13, pp. 9902–9909, 2025, doi: 10.1109/ACCESS.2024.3522972.

[13] D. Elreedy, A. F. Atiya, and F. Kamalov, "A theoretical distribution analysis of synthetic minority oversampling technique (SMOTE) for imbalanced learning," *Mach. Learn.*, vol. 113, no. 7, pp. 4903–4923, Jul. 2024, doi: 10.1007/s10994-022-06296-4.

[14] N. Mohanty, B. K. Behera, C. Ferrie, and P. Dash, "A quantum approach to synthetic minority oversampling technique (SMOTE)," *Quantum Mach. Intell.*, vol. 7, no. 1, Jun. 2025, doi: 10.1007/s42484-025-00248-6.

[15] H. Sun, J. Li, and X. Zhu, "A Novel Expandable Borderline Smote Over-Sampling Method for Class Imbalance Problem," *IEEE Trans. Knowl. Data Eng.*, vol. 37, no. 5, pp. 2183–2199, 2025, doi: 10.1109/TKDE.2025.3544284.

[16] G. Hou, D. L. Tong, S. Y. Liew, and P. Y. Choo, "Comparative Analysis of Resampling Techniques for Class Imbalance in Financial Distress Prediction Using XGBoost," *Mathematics*, vol. 13, no. 13, Jul. 2025, doi: 10.3390/math13132186.

[17] M. Ilham, A. Winarno, M. Lutfi, and A. Indrasetianingsih, "Handling Imbalanced Fraudulent Transaction Data Using SMOTE-Tomek and Random Forest: A Classification Approach," *BEST Journal of Applied Electrical & Science Technology*, 2025, doi: 10.36456/best.vol7.no1.10335.

[18] D. Yilmaz Eroglu and M. S. Pir, "Hybrid Oversampling and Undersampling Method (HOUM) via Safe-Level SMOTE and Support Vector Machine," *Applied Sciences (Switzerland)*, vol. 14, no. 22, Nov. 2024, doi: 10.3390/app142210438.

[19] E. H. Yulianti, O. Soesanto, and Y. Sukmawaty, "Penerapan Metode Extreme Gradient Boosting (XGBOOST) pada Klasifikasi Nasabah Kartu Kredit," *JOMTA Journal of Mathematics: Theory and Applications*, vol. 4, no. 1, 2022, doi: 10.31605/jomta.v4i1.1792.

[20] M. Wiens, A. Verone-Boyle, N. Henscheid, J. T. Podichetty, and J. Burton, "A Tutorial and Use Case Example of the eXtreme Gradient Boosting (XGBoost) Artificial Intelligence Algorithm for Drug Development Applications," *Clin. Transl. Sci.*, vol. 18, no. 3, Mar. 2025, doi: 10.1111/cts.70172.

[21] M. Abdelhaq, S. K. Palanisamy, M. Gopinath, V. G. S. Manasa, M. A. Ram, and S. K. MD, "A hybrid XGBoost–SVM ensemble framework for robust cyber-attack detection in the internet of medical things (IoMT)," *Sci. Rep.*, vol. 16, no. 1, Dec. 2026, doi: 10.1038/s41598-026-37832-0.

[22] H. Elwahsh *et al.*, "Hyperparameter optimization of XGBoost and hybrid CnnSVM for cyber threat detection using modified Harris hawks algorithm," *PeerJ Comput. Sci.*, vol. 11, Sep. 2025, doi: 10.7717/peerj-cs.3169.

[23] M. Maghanaki, S. Keramati, F. F. Chen, and M. Shahin, "Systematic Evaluation of Machine Learning and Deep Learning Models for IoT Malware Detection Across Ransomware, Rootkit, Spyware, Trojan, Botnet, Worm, Virus, and Keylogger," *Sensors*, vol. 26, no. 6, Mar. 2026, doi: 10.3390/s26061750.

[24] P. Anand, P. Nandhini, J. J. Christy, and K. Shiyamala, "Cyber threat estimation and prevention using xgboost," in *ViTECoN 2023 - 2nd IEEE International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies, Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/ViTECoN58111.2023.10157276.

[25] I. Singh Makkar, A. Kumar Sinha, T. Pratap, and A. Midhun Kumar, "Optimizing Android Malware Detection for Resource-Constrained Devices via a Two-Stage ML Pipeline," in *2025 International Conference on Intelligent Computing and Knowledge Extraction, ICICKE 2025*, Institute of Electrical and Electronics Engineers Inc., 2025. doi: 10.1109/ICICKE65317.2025.11136486.

[26] Z. Khalid, M. Zeeshan, and I. Ullah, "FGSM-CW Hardened Static Android-Malware Detection with Incremental PCA on the CCCS-CIC AndMal-2020 Corpus," in *2025 IEEE 22nd International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT, HONET 2025*, Institute of Electrical and Electronics Engineers Inc., 2025, pp. 122–127. doi: 10.1109/HONET67928.2025.11318437.

[27] D. F. Duarte and A. A. Bortoli, "Empirical Evaluation of SMOTE in Android Malware Detection with Machine Learning: Challenges and Performance in CICMalDroid 2020," Feb. 2026, doi: 10.5281/zenodo.15620300.

[28] N. A. Azhar, M. S. Mohd Pozi, A. M. Din, and A. Jatowt, "An investigation of SMOTE based methods for imbalanced datasets with data complexity analysis," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 7, pp. 6651–6672, Jul. 2023, doi: 10.1109/TKDE.2022.3179381.

[29] N. A. Azhar, M. S. Mohd Pozi, A. M. Din, and A. Jatowt, "An investigation of SMOTE based methods for imbalanced datasets with data complexity analysis," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 7, pp. 6651–6672, Jul. 2023, doi: 10.1109/TKDE.2022.3179381.

[30] A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. Gagnon, and F. Massicotte, "DIDroid: Android malware classification and characterization using deep image learning," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Nov. 2020, pp. 70–82. doi: 10.1145/3442520.3442522.

[31] N. V Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002, doi: 10.1613/jair.953.

[32] I. Tomek, "Two Modifications of CNN," *IEEE Transactions on Systems Man and Communications*, pp. 769–772, 1976, doi: 10.1109/TSMC.1976.4309452.

[33] G. Lemaitre, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning," Sep. 2016, [Online]. Available: http://arxiv.org/abs/1609.06570

[34] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[35] S. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," Nov. 2017, [Online]. Available: http://arxiv.org/abs/1705.07874