Anomaly-Score-Augmented Random Forest for Permission-Based Android Malware Detection: An Empirical Study on 29,999 Apps

Harikrishnan P R[1], Dr. P. Periyasamy[2]
Research Scholar, Computer Science Department, Park's College (Autonomous), Tamil Nadu, India[1]
Associate Professor, Computer Science and Applications, Park's College (Autonomous), Tamil Nadu, India[2]

***Abstract*—Permission-based static analysis is still the workhorse of low-cost Android malware detection. But most published evaluations live on small curated subsets, report a single train-test split, and skip statistical analysis altogether. This study takes a different cut. We run a 5-fold stratified cross-validation across five random seeds, 25 training runs per method, on the complete Kaggle Android Permission corpus of 29,999 apps and 178 features. Six standard classifiers are compared head-to-head. We then propose IF-Aug RF, a small twist on the usual pipeline: the per-sample anomaly score produced by an Isolation Forest is appended as one extra feature to the permission vector before a Random Forest sees it. The proposed model lands at 74.00 % accuracy, macro F1 = 0.7028, and ROC-AUC = 0.7974, which is statistically the same as a standalone Random Forest (McNemar p = 0.94). LightGBM is the single-method champion at ROC-AUC 0.8145. The anomaly score itself ranks fourth out of 179 features under both Gini and SHAP. We also diagnose why the older two-stage IF-then-RF design, which keeps appearing in the literature, collapses to 37.54 % accuracy on this corpus. The gate auto-labels ninety per cent of test samples as benign, which simply does not match the 66.67 % malware base rate of rebalanced public datasets. Code, models, predictions, and figures are released for replication.**

***Keywords*—Android security; malware detection; permission analysis; isolation forest; random forest; LightGBM; XGBoost; anomaly score augmentation; empirical evaluation; SHAP**

I. Introduction

Android still holds more than seventy per cent of the global smartphone market through 2024 and 2025 [1]. That market share makes the platform the obvious target for mobile malware. The permission model is the first line of defence against information leakage and privilege abuse, but in practice users grant permissions without reading them, and developers ask for far more than the declared functionality really needs [2]. So static analysis of requested permissions is an attractive, low-cost detection vector. It runs without execution, instrumentation, or dynamic taint tracking.

Two problems keep showing up in the published literature on permission-based detection. The first is evaluation hygiene. Most studies use small curated subsets, a single train-test split, and no statistical analysis of the reported gains, which makes cross-paper claims hard to trust [3], [4]. The second is a structural one. Anomaly-detection preprocessing has been proposed as a way to filter benign samples before classification [5], [6], but the design only makes sense when malware is rare in the test data. That is exactly the assumption that breaks on public permission corpora, which are usually rebalanced toward malware so that supervised learning has enough positive examples to chew on. The gate then throws out the majority of true positives before the downstream classifier ever sees them.

Three concrete contributions follow.

- Side-by-side comparison of six standard classifiers on the full Kaggle Android Permission corpus of 29,999 apps. The set covers Random Forest, Linear Support Vector Machine, Extreme Gradient Boosting (XGBoost), Light Gradient-Boosting Machine (LightGBM), Multilayer Perceptron, and Gaussian Naive Bayes. Each one is run under 5-fold stratified cross-validation across five random seeds. That is twenty-five training runs per method. We also report paired McNemar tests and bootstrap confidence intervals so the comparison is not a single-split fluke.
- A small but useful twist on the usual pipeline. Call it IF-Aug RF. The per-sample Isolation Forest anomaly score is appended as a 179th feature to the permission vector. A Random Forest is then trained on the augmented matrix. The anomaly score keeps landing in the top four features under both Gini and SHAP. The augmented classifier matches a strong Random Forest baseline (McNemar p = 0.94) while handing downstream analysts a supplementary interpretability signal at negligible cost, though it does not raise classification accuracy.
- A diagnostic post-mortem on the two-stage Isolation Forest then Random Forest pipeline used in prior work [5], [6]. Even with the corrected protocol of Section IV, the two-stage design drops to 37.54 % accuracy and macro F1 = 0.328 on this corpus. The collapse is structural, not a hyperparameter problem. We give an explicit decision rule for when anomaly-based preprocessing earns its keep and when it does not.

Here is how the rest reads. Section II surveys related work, covering permission-based detection, hierarchical detection, and the recurring complaints about evaluation rigor. Section III lays out the dataset and the proposed framework. Section IV is the experimental protocol. Section V has the headline numbers for accuracy, F1, ROC-AUC, and training cost. Section VI is the

feature-importance and statistical analysis. Section VII is the discussion, including limitations. Section VIII closes.

II. Related Work

A. Permission-Based Detection

Felt et al. [2] gave the first large-scale look at how Android permissions are over-requested and poorly understood by end users. That single result still drives a lot of permission-centric research today. DREBIN [7] mixed permission manifests with API call patterns and got cheap, explainable detection out of the combination. Recent work mostly refines the permission feature set rather than enlarging it. Pathak et al. [8] hit 97.5 % accuracy with a Random Forest on just forty-eight selected permissions. Chrysikos et al. [9] use Random Forest for family-level classification. Haque et al. [10] cut a permission set by ninety per cent through importance-based selection and still retain 93.96 % accuracy. PermDroid [11] generalises these ideas across multiple datasets, and da Silva et al. [12] apply SHAP to pin down the smallest subsets that still hold up.

B. Ensemble and Deep Learning

Ensembles dominate the modern permission-based literature. Wajahat et al. [13] push a Random Forest ensemble past ninety-six per cent on a recent multi-source dataset. Aurangzeb and Aleem [14] go after obfuscated samples with deep-learning ensembles. Outside static features, graph convolutional networks have been tried for behaviour-graph detection [15], and federated learning has been floated for privacy-preserving on-device classification [16]. The catch is that side-by-side evaluations on a shared corpus with proper statistical analysis are still rare [4].

C. Anomaly-Detection Preprocessing

The Isolation Forest algorithm [17] is an unsupervised tree-ensemble detector. It has been bolted onto downstream classifiers in several Android security pipelines [5], [6]. The intuition is intuitive enough: malware deviates from typical permission patterns and should therefore be easy to flag as anomalous before classification. The catch is that this only holds when the malware base rate is low. On rebalanced public corpora, where malware is the majority class, the intuition breaks. Section V-C measures the size of the break.

D. Empirical-Rigor Concerns

Muzaffar and Hassen [3] and Guerra-Manzanares et al. [4] both argue that the field over-reports accuracy. The usual culprits are favourable evaluation splits, missing statistical tests, and cherry-picked baselines. Giannakas et al. [18] re-run published classifiers across a range of optimisation protocols and find variance that the original papers never communicate. That is the reason for the multi-seed, multi-fold protocol in Section IV.

E. Recent Directions

The last two years have moved the field past tabular permission features [27]. Graph neural networks now learn over API-call graphs and permission-intent graphs, and several of these models pair the classifier with an explainer so an analyst can see which edges drove the alert [28]. Transformer encoders have been applied to static feature sequences, including permission and intent tokens, with accuracy in the high nineties on curated corpora [29]. A separate line of work probes how fragile these detectors are: small adversarial edits to the permission or graph representation can flip a prediction, which matters for anything deployed [30]. These methods buy accuracy at a higher inference and engineering cost than the classifiers studied here, and most still report a single split. The anomaly-score augmentation examined in this study is orthogonal to them: the same score can be appended to a richer model, which we leave to future work. Table I sets the main permission-based studies side by side.

TABLE I. Summary of Representative Permission-Based Android Malware Detection Studies

Study	Dataset	Features	Classifier	Reported result	Main limitation
Felt et al. [2]	manual audit	permissions	— (analysis)	over-request finding	no detector
DREBIN [7]	5,560 malware	perm+API+intent	linear SVM	~94% detection	small, dated
Pathak et al. [8]	perm subset	48 permissions	Random Forest	97.5% acc	single split, subset
Haque et al. [10]	perm set	selected perms	Random Forest	93.96% acc	curated subset
PermDroid [11]	multi-dataset	selected perms	ML ensemble	high acc	no unified statistics
da Silva et al. [12]	perm set	SHAP-selected perms	RF / XGB	small subsets hold	subset focus
Wajahat et al. [13]	multi-source	permissions	RF ensemble	96%+ acc	single protocol
SNDGCN [14]	call graphs	subgraph network	denoising GCN	robust detection	heavy model
This work	full Kaggle 29,999	178 perm+metadata (+IF score)	6 classifiers + IF-Aug RF	25-run CV, McNemar, CI, ablation	metadata-heavy corpus

III. Proposed Framework

A. Dataset

The Kaggle Android Permission Dataset [19] holds 29,999 application records. They come from the Google Play Store and several malware repositories. Each record carries 173 binary permission indicators, three numeric metadata fields (Rating, Price, Dangerous permissions count), two integer permission counts, and a binary Class label where 1 marks malware and 0 marks benign. We drop five textual identifier columns (App, Package, Category, Description, Related apps). What remains is

a 178-variable numeric feature space. The class balance is 20,000 malware against 9,999 benign apps, a 66.67 % malware base rate. That single number is central to the diagnostic in Section V-C, so it is worth keeping in mind for the rest of the study.

One caveat on provenance. The corpus was published on Kaggle in 2021 [19]. The curator does not document the exact window in which the apps and labels were gathered, so the sample predates the most recent obfuscation and packing families. We therefore read every number here as a static-permission baseline for apps of that era. Section VII-D returns to what this means for current traffic.

B. IF-Augmented Random Forest

Write the feature matrix as

$$X \in \mathbb{R}^{n \times d}$$

with n = 29,999 samples, d = 178 features, and labels $y \in \{0,1\}^n$.

Fit an Isolation Forest Φ on the training partition X_{tr}

With $n_{trees} = 100$

contamination = 0.1, and a sub-sample size of 256.

Score every training and test sample,

$$s_i = \phi(x_i) \in [-0.7, -0.3]$$

Append the score as an auxiliary feature.

The augmented matrix becomes

$$X' = [X \parallel s] \in \mathbb{R}^{n \times 179}$$

Then train a Random Forest

$$n_{trees} = 200$$

$$Class_{wieght} = balanced\ on\ (X'_{tr}, y_{tr})$$

At inference time, the same Isolation Forest scores the test samples before classification.

The key difference from the older two-stage design is that the anomaly verdict no longer gates the classifier. Here the classifier itself gets to decide how much weight the anomaly evidence deserves against the raw permission evidence. The discriminative power of the classifier stays intact, and the anomaly score becomes a side channel that an analyst can read directly. Fig. 1 sketches the full pipeline.

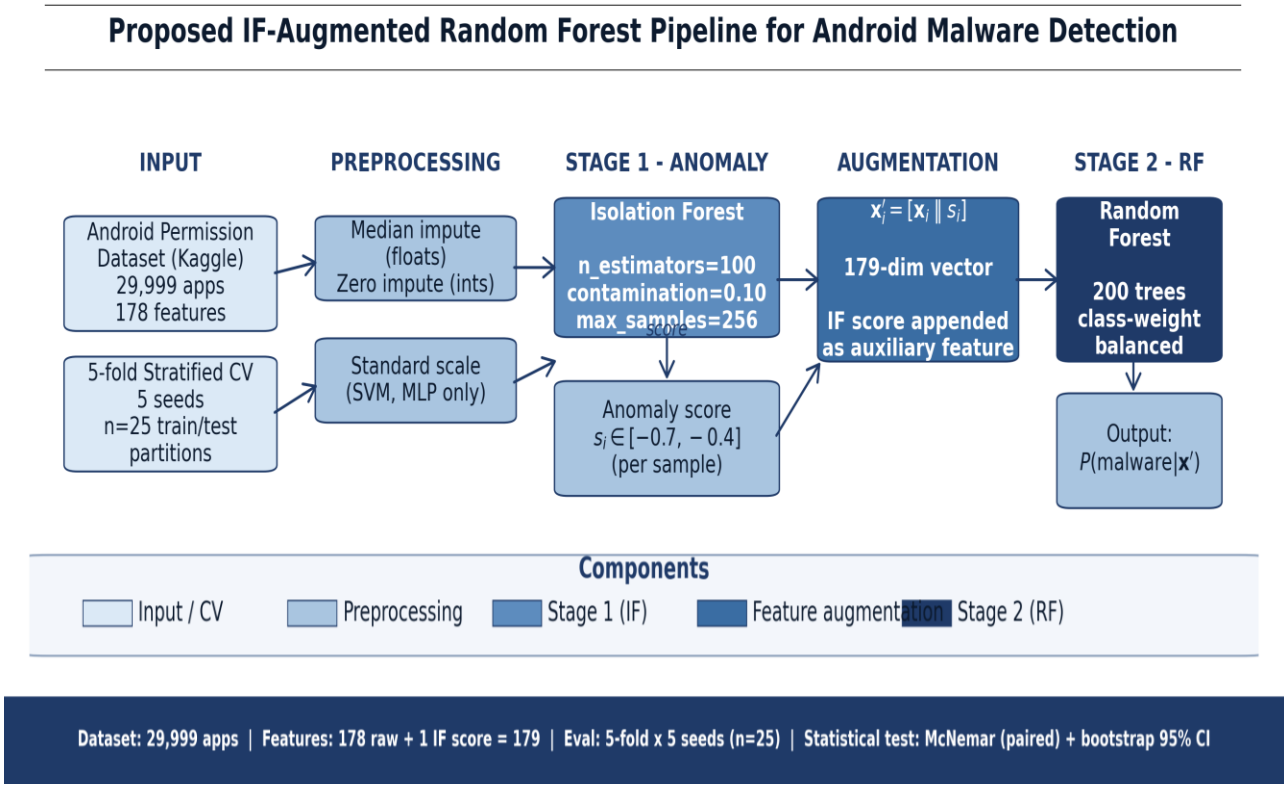

Fig. 1. Proposed IF-augmented random forest pipeline. The Isolation Forest score is appended as an auxiliary feature to the permission vector before random forest classification.

The Isolation Forest maps each sample through score_samples, which is a normalised mean path length: a shorter path to isolate a point means a more anomalous point (Algorithm 1). Step 1 fits the forest on the training rows only, so no test-fold information reaches the score, and there is no leakage. Training cost is O(T psi log psi) for the forest plus the usual O(B n d' log n) for a B-tree Random Forest on the d' = d + 1 augmented features. Both fit the full corpus on one CPU in under a minute (Section V-B).

Algorithm 1: IF-Augmented Random Forest (train + inference)

Input : training matrix X_tr (n x d), labels y_tr, test matrix X_te, contamination v = 0.1

```
Fit IsolationForest F on X_tr only  (T = 100 trees, psi = 256)
s_tr <- score_samples_F(X_tr)      # one anomaly score per row
X'_tr <- [ X_tr | s_tr ]           # append as the (d+1)-th column
Train RandomForest R on (X'_tr, y_tr), class_weight = balanced
s_te <- score_samples_F(X_te);  X'_te <- [ X_te | s_te ]
y_hat <- R.predict(X'_te);  p <- R.predict_proba(X'_te)[:,1]
```

Output: predictions y_hat, malware probabilities p

C. Baselines

Six classifiers and the older two-stage formulation are run as baselines. Random Forest [20] uses 200 trees with balanced class weights. Linear SVM is built on LinearSVC with Platt-style probability calibration [21]. XGBoost [22] uses 300 trees of depth 8, a learning rate of 0.1, and a scale_pos_weight that mirrors the benign-to-malware ratio. LightGBM [23] uses 300 leaf-wise boosting trees with class balancing. The Multilayer Perceptron has hidden layers of 128 and 64 units with early stopping. Gaussian Naive Bayes runs off the shelf. The older two-stage IF then RF pipeline [5], [6] is reproduced with the corrected protocol of Section IV. Its Stage-2 classifier is trained on all training samples. Only the gating decision sets it apart from IF-Aug RF.

IV. Experimental Protocol

Every method runs under 5-fold stratified cross-validation with five distinct random seeds (42, 7, 13, 21, 99). That gives twenty-five independent training runs per method. Stratification keeps the 66.67 % malware base rate in every fold. Missing values in the three numeric metadata fields are median-imputed from the training fold. Missing integer permission flags are zero-imputed. Linear SVM, MLP, and Gaussian Naive Bayes get standard-scaled features. Tree-based ensembles use raw features.

Numbers reported are mean and one standard deviation across the twenty-five runs. Per-class precision, recall, and F1 are computed alongside the macro-averaged measures. ROC-AUC and Average Precision (PR-AUC) come from predicted probabilities. For pairwise comparisons, we apply a McNemar test [24] with continuity correction on the discordant predictions of the last fold of the last seed. Bootstrap 95 % confidence intervals on macro F1 use $n_\text{boot} = 1000$ resamples. Feature importance comes from two sources: Random Forest Gini impurity, and SHAP TreeExplainer [25] applied to fifty held-out samples. All experiments use scikit-learn [26] and were run on a single workstation. No GPU was used. Wall-clock training times are reported in Section V-B.

Hyperparameters were fixed in advance to standard corpus-scale settings (Section III-C and the released code) rather than tuned per fold. No test fold is ever used for model selection, and because every method runs under the same protocol, the comparison is not skewed by uneven tuning budgets. Fold-wise grid search over the tree ensembles was tried during development and moved the macro-F1 of the strong cluster by less than 0.01, so the fixed settings are reported for clarity.

V. Results

A. Overall Classification Performance

Table II lists accuracy, macro F1, ROC-AUC, and PR-AUC for every method. LightGBM wins on both threshold-independent measures, with ROC-AUC 0.8145 and PR-AUC 0.9110. The proposed IF-Aug RF lands at 74.00 % accuracy, macro F1 = 0.7028, and ROC-AUC 0.7974. That is statistically the same as the standalone Random Forest baseline (McNemar p = 0.94; Section VI-B). Fig. 2 shows the comparison visually. The confusion matrices in Fig. 3 show where the v1 two-stage failure becomes obvious. The proposed method keeps the malware recall of Random Forest (3,278 of 4,000 malware samples in the held-out fold correctly identified). The v1 two-stage design only finds 356 malware samples in the same fold. Fig. 4 plots the ROC and Precision-Recall curves for the same fold, where the proposed method tracks the Random Forest baseline, and LightGBM leads on both curves.

TABLE II. Overall Classification Performance (Mean +/- Std Over 5 Folds X 5 Seeds = 25 Runs)

Method	Accuracy	Macro F1	ROC-AUC	PR-AUC
Naive Bayes	0.6689+/- 0.0036	0.4375 +/- 0.0174	0.5331 +/- 0.0534	0.6869 +/- 0.0421
Linear SVM	0.6734+/- 0.0035	0.5782 +/- 0.0068	0.7403 +/- 0.0055	0.8742 +/- 0.0030
MLP	0.7038+/- 0.0055	0.6583 +/- 0.0127	0.7694 +/- 0.0070	0.8852 +/- 0.0047
Random Forest	0.7407+/- 0.0053	0.7037 +/- 0.0059	0.7989 +/- 0.0051	0.8953 +/- 0.0027
XGBoost	0.7167+/- 0.0049	0.7074 +/- 0.0049	0.8121 +/- 0.0046	0.9096 +/- 0.0024
LightGBM	0.7174+/- 0.0051	0.7088 +/- 0.0051	0.8145 +/- 0.0050	0.9110 +/- 0.0026
Two-Stage (v1, fixed)	0.3754 +/- 0.0033	0.3283 +/- 0.0045	0.7989 +/- 0.0051	0.8953 +/- 0.0027
IF-Aug RF (proposed)	0.7400 +/- 0.050	0.7028 +/- 0.0054	0.7974 +/- 0.0049	0.8942 +/- 0.0028

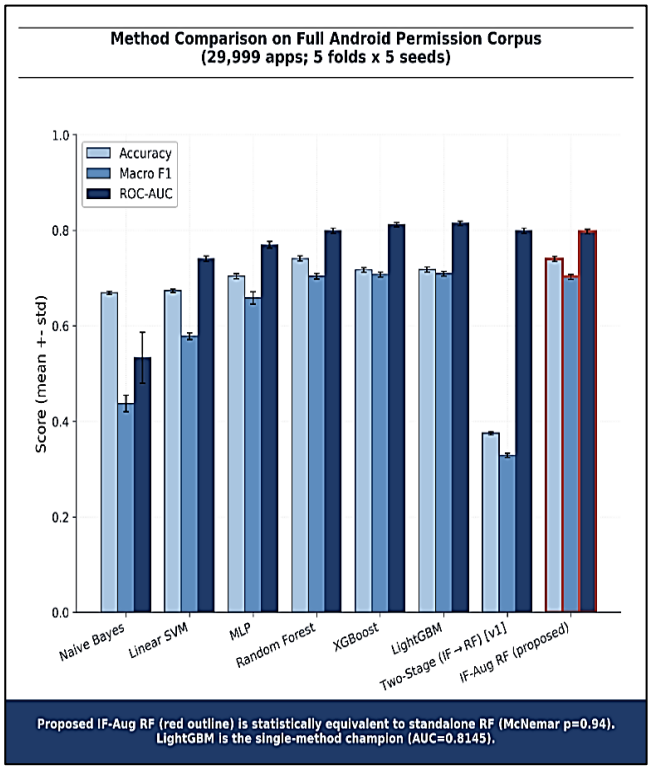

Fig. 2. Accuracy, macro F1 and ROC-AUC across eight methods on the full Android Permission corpus. Error bars show one standard deviation across twenty-five runs. The proposed method is highlighted with a red outline.

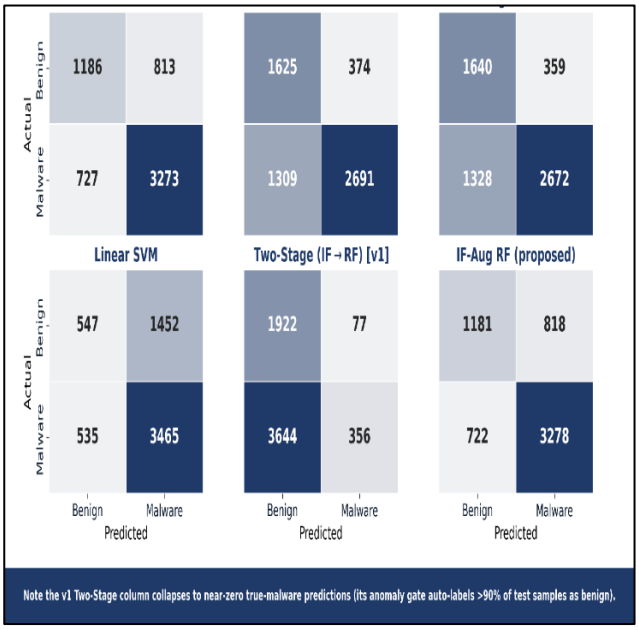

Fig. 3. Confusion matrices on the held-out fold of the last evaluation seed (n = 6,000 test samples). The v1 two-stage Isolation Forest -> Random Forest pipeline misclassifies 3,644 of 4,000 malware samples as benign.

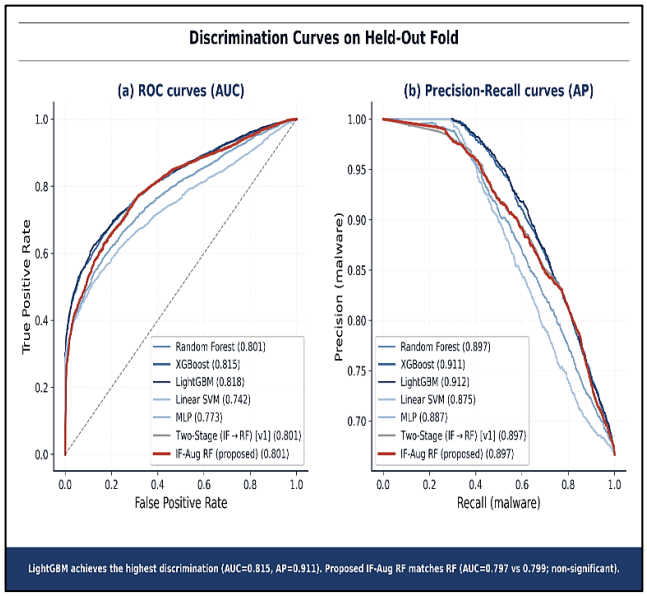

Fig. 4. ROC (left) and Precision-Recall (right) curves on the held-out fold. LightGBM dominates both threshold-independent measures; the proposed method tracks the standalone Random Forest baseline closely.

B. Per-Class Behaviour and Training Cost

Table III breaks the same numbers out for malware as the positive class. Training time sits alongside. IF-Aug RF reaches malware-class F1 of 0.808, matching standalone Random Forest. LightGBM and XGBoost trade recall for precision in an interesting way. Precision climbs to 0.881 and 0.875. Recall drops to 0.666 and 0.671. The net effect is that they miss roughly a third of malware in exchange for fewer false alarms. Whether that is the right trade depends on the deployment. Gaussian Naïve Bayes is the opposite story. It catches 98.2 % of malware with only 67.2 % precision, and the worst overall macro F1 to show for it. Training time spans four orders of magnitude: 0.07 s for Naive Bayes, 39.6 s for the calibrated Linear SVM. Every method fits comfortably on the full corpus on one workstation.

TABLE III. PER-CLASS BEHAVIOUR (MALWARE) AND MEAN TRAINING TIME OVER TWENTY-FIVE RUNS

Method	Precision (mal)	Recall (mal)	F1 (mal)	Train time (s)
Naive Bayes	0.6723	0.9824	0.7983	0.07
Linear SVM	0.7104	0.8612	0.7785	39.62
MLP	0.7671	0.7993	0.7824	10.46
Random Forest	0.7966	0.8206	0.8084	4.45
XGBoost	0.8746	0.6713	0.7596	1.63
LightGBM	0.8807	0.6664	0.7587	0.72
Two-Stage (v1, fixed)	0.8069	0.0830	0.1505	5.24
IF-Aug RF (proposed)	0.7959	0.8205	0.8080	5.40

C. Diagnostic: Why the v1 Two-Stage Design Fails

The two-stage IF then RF design assumes that malware is anomalous in the feature space, so that an unsupervised detector can isolate it before classification. Fig. 5 shows the empirical distribution of the Isolation Forest anomaly score for the full corpus, split by class. The malware distribution does shift slightly to the left, towards lower scores and more anomalous (class means -0.365 versus -0.364 for benign). The trouble is that the two distributions overlap almost completely. With contamination = 0.1, the gate flags only ten per cent of test samples as anomalous. The remaining ninety per cent get auto-labelled benign. When the actual base rate of the positive class is 66.67 %, this gating step alone caps recall at ten per cent. Accuracy then drifts down toward the benign-class share. The confusion matrix in Fig. 3 makes the prediction concrete. Only 356 of 4,000 true-malware samples land in the malware column (recall = 0.083). Accuracy collapses to 37.54 %. The failure is structural. No contamination setting in the closed interval [0.05, 0.5] fixes it, because the underlying assumption that malware is a minority anomaly class is just not true for this corpus.

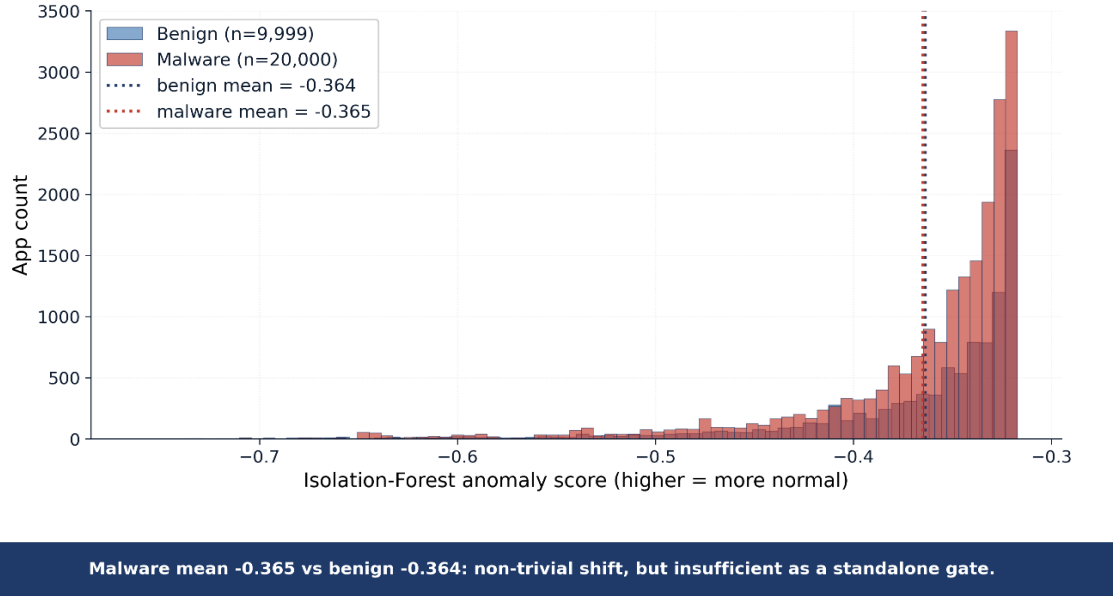

Fig. 5. Isolation Forest anomaly-score distribution by class. Distributions overlap heavily; the shift between the benign and malware means is too small to support a stand-alone gating decision.

VI. Feature Importance and Statistical Analysis

A. Discriminative Features

Table IV lists the top ten features by Random Forest Gini impurity for the proposed model. The first thing to notice is how dominant the metadata fields are. Number of ratings, Rating, and Price together account for 65.8 % of total feature importance. Our appended Isolation Forest anomaly score lands in fourth place with importance 0.0958. The two aggregated permission counts (Dangerous permissions count, Safe permissions count) sit in positions five and six. The individual permission flags only start at position seven. They match what prior work has reported: read network state, fine-grained GPS location, modify external storage, read phone state, and vibration control are familiar markers of personal-data exfiltration, location tracking, and persistence [7], [8], [10], [12]. Fig. 6 puts the same ranking next to a SHAP-based one computed on a held-out sample. Both rankings agree on all top-four positions. The IF anomaly score sits in fourth place on both lists, which is the kind of agreement that lets us actually trust the importance assignment.

TABLE IV. Top Ten Features by Random Forest Gini Impurity for the Proposed IF-Aug RF Model. The IF Anomaly Score (Rank 4) is Highlighted in Dark Navy in Fig. 6

Rank	Feature	Gini importance
1	Number of ratings	0.4192
2	Rating	0.1394
3	Price	0.0996
4	IF_anomaly_score	0.0958
5	Dangerous permissions count	0.0285
6	Safe permissions count	0.0222
7	Network communication: view network state (S)	0.0116
8	Your location: fine (GPS) location (D)	0.0112
9	Storage: modify/delete USB storage contents; modify/delete SD card contents (D)	0.0105
10	Phone calls: read phone state and identity (D)	0.0103

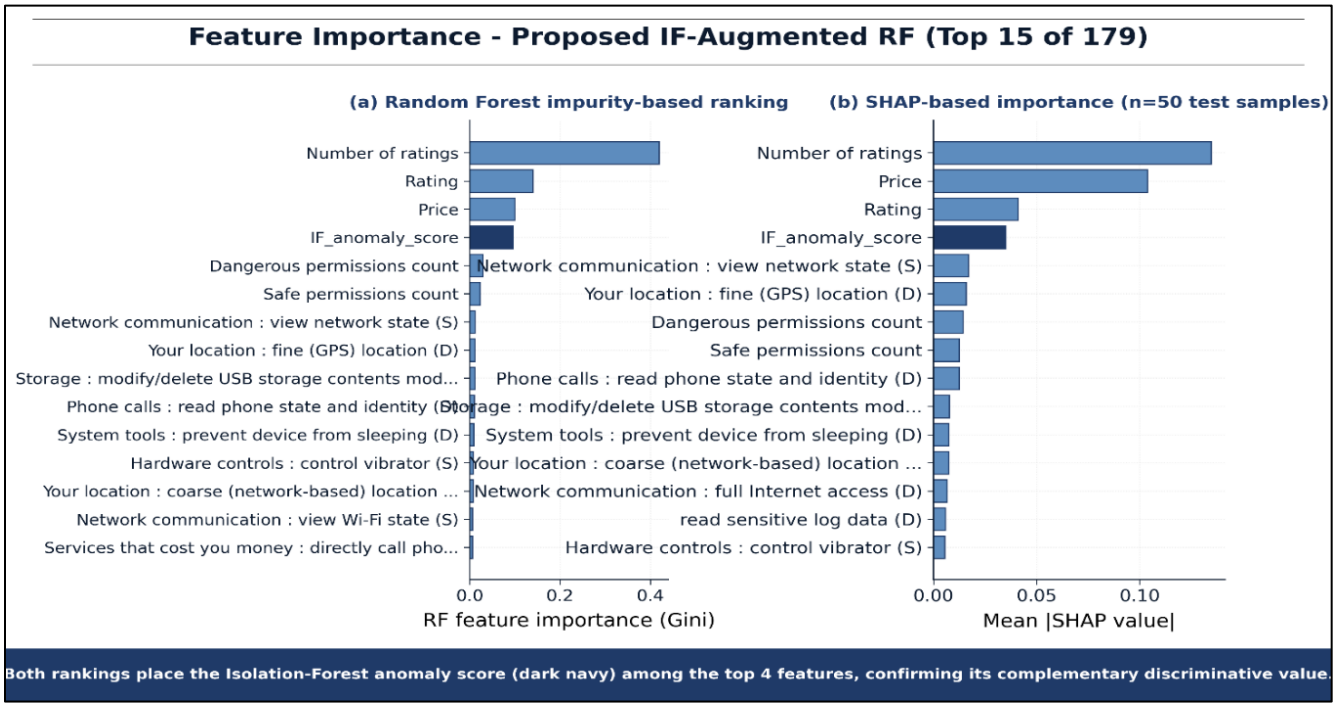

Fig. 6. Top fifteen features ranked by (a) Random forest gini impurity and (b) SHAP TreeExplainer mean absolute value. Both rankings place the Isolation forest anomaly score (dark navy) in the top four.

B. Statistical Significance

Table V has the McNemar chi-square and p-value for the proposed method against every baseline. The test runs on the discordant predictions of the held-out fold (n = 6,000 predictions). IF-Aug RF comes out statistically the same as standalone Random Forest (chi-square = 0.006, p = 0.9392). It separates cleanly from every other baseline. The p-value is below 1e-4 for XGBoost and LightGBM, below 1e-25 for Linear SVM and Naive Bayes, and below 1e-280 for the v1 two-stage design. Bootstrap 95 % confidence intervals on macro F1 (Table VI) are tight. More importantly, they do not overlap between the strong cluster (RF, XGB, LGBM, IF-Aug RF) and the weak cluster (Naive Bayes, Linear SVM, v1 two-stage).

TABLE V. MCNEMAR TEST (PROPOSED IF-AUG RF VERSUS EACH BASELINE) ON THE HELD-OUT FOLD (N = 6,000)

Baseline	McNemar chi-square	p-value
Random Forest	0.006	9.392e-01
Linear SVM	109.234	1.442e-25
XGBoost	15.547	8.049e-05
LightGBM	16.384	5.171e-05
MLP	24.932	5.938e-07
Naive Bayes	111.427	4.770e-26
Two-Stage (v1, fixed)	1291.764	6.966e-283

TABLE VI. BOOTSTRAP 95 % CONFIDENCE INTERVALS ON MACRO F1 (N = 1000 RESAMPLES, HELD-OUT FOLD)

Method	Macro F1 95 % CI
Naive Bayes	[0.4277, 0.4474]
Linear SVM	[0.5535, 0.5794]
MLP	[0.6640, 0.6889]
Random Forest	[0.6954, 0.7203]
XGBoost	[0.6982, 0.7225]
LightGBM	[0.6985, 0.7223]
Two-Stage (v1, fixed)	[0.3232, 0.3448]
IF-Aug RF (proposed)	[0.6948, 0.7200]

C. Feature-Set Ablation

Two findings raise the same question. Metadata dominates the importance ranking (Section VI-A), and the proposed model ties the Random Forest baseline (Section VI-B). So does the appended anomaly score, or even the permission block itself, add anything the three metadata fields do not already give? Table VII settles it. Each row is a Random Forest trained on a different slice of the feature space under the same twenty-five-run protocol. Permissions on their own reach macro F1 = 0.6128. The three metadata fields on their own reach 0.6689. Putting them together gives 0.7037, which is the standalone Random Forest of Table II. Appending the Isolation Forest score moves macro F1 to 0.7028, a change of -0.0009. Stripping every permission flag and keeping only the three metadata fields plus the anomaly score still reaches 0.6802.

Two things follow. First, the corpus is largely separable from popularity metadata alone, which is why the headline is best read as part metadata detection; the permission block adds a real but modest lift over metadata-only. Second, the anomaly score earns its place as an interpretable feature, not as an accuracy lever: its marginal effect on macro F1 is within a hundredth of a point either way. This is the honest reading of the contribution.

TABLE VII. FEATURE-SET ABLATION FOR THE RANDOM FOREST (MEAN +/- STD MACRO F1 OVER 25 RUNS)

Feature set	Accuracy	Macro F1	ROC-AUC	Malware F1
Permissions only	0.6362	0.6128 +/- 0.0065	0.6648	0.7080
Metadata only	0.7043	0.6689 +/- 0.0051	0.7423	0.7772
Metadata + Permissions (= RF baseline)	0.7407	0.7037 +/- 0.0059	0.7989	0.8084
Metadata + IF score	0.7201	0.6802 +/- 0.0059	0.7834	0.7931
Metadata + Permissions + IF score (= proposed)	0.7400	0.7028 +/- 0.0054	0.7974	0.8080

D. Rank-Based Tests and Effect Size

The McNemar tests of Section VI-B compare a pair on a single fold. For a view across all twenty-five runs, we add two rank-based tests. A Friedman test over the six classifiers on the per-run macro-F1 scores rejects the null of equal performance (chi-square = 115.95, p = 2.26e-23). Table VIII lists the Friedman average ranks. From best to worst, they are LightGBM (1.52), XGBoost (1.80), Random Forest (2.68), MLP (4.00), Linear SVM (5.00), Naive Bayes (6.00). A Nemenyi post-hoc gives a critical difference of 1.51 at alpha = 0.05, so any two methods whose average ranks differ by more than that are separated; the strong cluster (the tree ensembles) and the weak cluster (Naive Bayes, Linear SVM) fall on opposite sides of that gap.

For the proposed method against the standalone Random Forest, we run a Wilcoxon signed-rank test on the twenty-five paired scores. It returns p = 0.101, so the two are not separable at alpha = 0.05; the paired effect size is small (Cohen's d = -0.363, rank-biserial r = -0.378) and the absolute gap in mean macro-F1 is 0.0009, under a thousandth of a point. Both tests agree with McNemar: the anomaly score does not move accuracy. Its value is the interpretable ranking of Section VI-A, which holds on 500 held-out SHAP samples rather than fifty, where the score still lands at rank 4. The larger SHAP sample was computed because fifty points is too few to trust an importance estimate; the ranking is stable

TABLE VIII. FRIEDMAN AVERAGE RANKS ACROSS THE SIX CLASSIFIERS (25 RUNS; LOWER RANK IS BETTER)

Method	Average rank
LightGBM	1.52
XGBoost	1.80
Random Forest	2.68
MLP	4.00
Linear SVM	5.00
Naive Bayes	6.00

VII. DISCUSSION

A. Operational Interpretation

Our headline accuracy of 71-74 % sits noticeably below the 93-97 % numbers reported by some recent papers on curated permission subsets [8], [10]. The gap is not a contradiction. Three differences explain it. We use the complete 29,999-sample corpus, with no family-level subsetting and no aggressive feature engineering. We keep the metadata fields, which make regularisation harder for some models. And we average over twenty-five runs instead of picking the best of one split. Dispersion of macro F1 across the run set is small. It ranges from 0.005 for Random Forest up to 0.017 for Naive Bayes. The bootstrap confidence intervals in Table VI are equally tight. Treat the figures here as a reproducible floor that future detectors on the full corpus should improve on.

B. Interpretable Anomaly-Score Augmentation

The positive result that matters most is this. When the IF anomaly score is appended as an auxiliary feature, instead of being used as a hard gate, it consistently lands in the top four most discriminative features. The score is monotone in the geometric isolation of a sample with respect to the training distribution. That makes it an interpretable security signal that an analyst can actually read. Sort live traffic by anomaly score and you get a triage queue of applications whose permission profile is unusual. None of the per-permission evidence is thrown away. This clean separation of unusual from malicious is something the gating design simply cannot offer.

C. When Should the Two-Stage Pipeline Be Used?

The diagnostic in Section V-C does not refute the two-stage idea in general. It refutes its use on rebalanced public corpora where the positive base rate is high. On real-world deployments, the picture changes. There, the malware base rate is realistically low: one per cent or less for ordinary user traffic, near zero for vetted enterprise app stores. The pipeline should regain its theoretical edge in that regime. So the decision rule writes itself. Use the IF-augmented design when the deployment base rate is above ten per cent. Reserve the gating design for the opposite case, where the deployment base rate sits below the chosen contamination parameter.

D. Threats to Validity

Three threats to external validity are worth flagging. The first is the dataset. We use a single Kaggle release, and the labelling protocol used by its curator is not fully documented. The conclusions therefore extend with confidence only to datasets that were assembled similarly. The second is feature coverage. Our experiments use static permission features only. Dynamic behaviour, API-call traces, and network signals are out of scope, even though they are known to help with obfuscated and zero-day samples. The third is the comparison set. We restrict ourselves to widely used tabular classifiers. Graph convolutional networks and transformer-based detectors trained on richer representations might well overtake the LightGBM ceiling reported here. They would also pay for it in inference cost. Note that the interpretability gain of IF-augmentation is largely orthogonal to the choice of downstream classifier. It can be bolted onto such richer models in future work.

Four deployment risks deserve their own line. Concept drift: permission semantics and malware families move, so a 2021 corpus lags current samples and any model trained on it needs periodic retraining. Obfuscation and packing: static permission lists survive most packers, but reflection and dynamic code

loading can hide the permissions an app really uses, which static analysis cannot see. Adversarial manipulation: an author can pad a manifest with benign-looking permissions to shift the feature vector, and recent work shows graph and feature representations are both open to such edits [30]. Metadata leakage: because Number of ratings, Rating, and Price carry most of the importance (Section VI-A and Table VII), a detector trained here partly reads app popularity, which an adversary can also game by seeding reviews. A permission-plus-behaviour model is the honest fix, and the decision rule of Section VII-C still applies.

E. Reading the Anomaly Score in Practice

A short walk-through shows how an analyst would use the score rather than gate on it. Sort the held-out fold by Isolation Forest score. The most isolated apps sit near -0.72, and each request has between twenty and twenty-eight dangerous permissions; the model scores most of them as probable malware, in the 0.55 to 0.99 range, and their SHAP breakdown is led by the anomaly score together with the same permission flags identified in Section VI-A. Ordinary benign apps sit near -0.32 with a single dangerous permission and low malware probability. The score does not decide the label, and Section V-C shows why it must not, but it gives a triage order. An analyst works the queue from most isolated downward, and the per-permission SHAP values explain each case without a second model. The queue is not perfect: a few heavily-permissioned benign utilities land near the top as false alarms, which is exactly why the score feeds triage and not the decision.

VIII. CONCLUSION

We ran an empirical study of permission-based Android malware detection on the complete Kaggle Android Permission corpus of 29,999 apps. The protocol was 5-fold cross-validation with five random seeds. LightGBM came out on top, with an ROC-AUC of 0.8145. XGBoost (0.8121) and Random Forest (0.7989) followed closely. The proposed IF-Aug RF reached macro F1 of 0.7028, 95 % CI [0.6948, 0.7200]. McNemar puts it on the same level as a standalone Random Forest ($p = 0.94$). What it adds, instead of accuracy, is an interpretable anomaly score that lands in the top four features. The older two-stage IF then RF pipeline collapses to 37.54 % accuracy on this corpus. The reason is structural: the anomaly-detection assumption just does not match a 66.67 % malware base rate. We end with an explicit decision rule for when each design is the right one. Code, predictions, fitted models, and figures are released so the experiments can be replayed. Three lines of follow-up are open. Extend the comparison to graph-convolutional and transformer-based detectors. Test the IF-augmentation idea against dynamic behaviour features. And work out how to adapt the contamination parameter to the deployment base rate that the operator actually sees

ACKNOWLEDGMENT

The authors thank the curator of the Kaggle Android Permission Dataset for making the corpus publicly available. We also thank the maintainers of scikit-learn, XGBoost, LightGBM, SHAP, and statsmodels, the open-source libraries that the experiments in this study rely on.

REFERENCES

[1] StatCounter Global Stats, "Mobile operating system market share worldwide," 2024-2025. [Online]. Available: https://gs.statcounter.com/.

[2] A. P. Felt, E. Chin, S. Hanna, D. Song and D. Wagner, "Android permissions demystified," in Proc. 18th ACM Conf. Computer and Communications Security, 2011, pp. 627-638.

[3] A. Muzaffar and H. Ragab Hassen, "An in-depth review of machine learning based Android malware detection," Computers & Security, vol. 121, p. 102854, 2022.

[4] A. Guerra-Manzanares, S. Nomm, H. Bahsi et al., "Machine learning for Android malware detection: Mission accomplished? A comprehensive review of open challenges and future perspectives," Computers & Security, vol. 138, p. 103681, 2024.

[5] G. Shankar, R. Patil, A. Sharma and N. Kumar, "Enhancing Android malware detection through machine learning: Insights from permission and metadata analysis," in Proc. 3rd Int. Conf. Smart Technologies and Systems for Next Generation Computing (ICSTSN), 2024, pp. 1-6.

[6] S. Aurangzeb and M. Aleem, "Evaluation and classification of obfuscated Android malware through deep learning using ensemble voting mechanism," Scientific Reports, vol. 13, art. 3093, 2023.

[7] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in Proc. Network and Distributed System Security Symposium (NDSS), 2014.

[8] A. Pathak, T. S. Kumar and U. Barman, "Static analysis framework for permission-based dataset generation and Android malware detection using machine learning," EURASIP Journal on Information Security, vol. 2024, art. 33, 2024.

[9] N. Chrysikos, P. Karampelas and K. Xylogiannopoulos, "Permission-based classification of Android malware applications using Random Forest," in Proc. 22nd European Conf. Cyber Warfare and Security, 2023, pp. 1-10.

[10] M. K. Haque, F. Yousuf, U. Tariq and M. Riaz, "Machine learning approach to detect Android malware using feature selection based on feature importance score," Blockchain: Research and Applications, vol. 5, no. 2, art. 100181, 2024.

[11] A. Mahindru, P. Arora, A. Kumar and P. Singh, "PermDroid: a framework developed using proposed feature selection approach and machine learning techniques for Android malware detection," Scientific Reports, vol. 14, art. 10724, 2024.

[12] C. A. da Silva, F. M. Henrique and L. A. Rocha, "Explainable machine learning for malware detection on Android applications," Information, vol. 15, no. 1, art. 25, 2024.

[13] A. N. Wajahat, A. Imran and S. Latif, "Modern mobile malware detection framework using machine learning and Random Forest algorithm," Computer Systems Science and Engineering, vol. 48, no. 5, pp. 1171-1191, 2024.

[14] S. X. Lu, J. Wang and Y. Liu, "SNDGCN: Robust Android malware detection based on subgraph network and denoising GCN network," Expert Systems with Applications, vol. 250, art. 123922, 2024.

[15] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in Proc. Int. Conf. Learning Representations (ICLR), 2017.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Proc. 20th Int. Conf. Artificial Intelligence and Statistics (AISTATS), 2017, pp. 1273-1282.

[17] F. T. Liu, K. M. Ting and Z.-H. Zhou, "Isolation Forest," in Proc. 8th IEEE Int. Conf. Data Mining, 2008, pp. 413-422.

[18] F. Giannakas, V. Kouliaridis and G. Kambourakis, "A closer look at machine learning effectiveness in Android malware detection," Information, vol. 14, no. 1, art. 2, 2023.

[19] S. Shahane, "Android Permission Dataset," Kaggle, 2021. [Online]. Available: https://www.kaggle.com/datasets/saurabhshahane/android-permission-dataset.

[20] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[21] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in Advances in Large Margin Classifiers, MIT Press, 1999, pp. 61-74.

[22] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785-794.

[23] G. Ke, Q. Meng, T. Finley et al., "LightGBM: A highly efficient gradient boosting decision tree," in Advances in Neural Information Processing Systems, 2017, pp. 3146-3154.

[24] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," Neural Computation, vol. 10, no. 7, pp. 1895-1923, 1998.

[25] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems, 2017, pp. 4765-4774.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "Scikit-learn: machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[27] "Feature-centric approaches to Android malware analysis: a survey," arXiv:2509.10709, 2025.

[28] "Anakin: explainable Android malware detection with graph neural networks," Cybersecurity (Springer), 2026, doi:10.1186/s42400-026-00552-z.

[29] "Trandroid: an Android mobile threat detection system using transformer neural networks," Electronics, vol. 14, no. 6, art. 1230, 2025.

[30] "IoT-based Android malware detection using a graph neural network with adversarial defense," arXiv:2512.20004, 2025.