International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

AXMED++: Explainable Android Malware Detection with Clone-Free Temporal Evaluation

Priyadarshini K[1], Meghamala L[1] , Madhu Sri G[1]

[1]Dept. of CSE (AI & ML), GVP College of Engineering for Women, Visakhapatnam, AP, India

Abstract

Scope declaration: This is a *controlled simulation study*. Feature vectors are generated by calibrated probabilistic sampling rather than by parsing real APK files. Clone and repackaging metadata are drawn from the real AndroMalPack corpus. All results should be interpreted accordingly.

The proliferation of Android malware poses a severe and escalating threat to billions of mobile users worldwide. Existing machine-learning-based detection systems suffer from three pervasive weaknesses: (*i*) temporal data leakage arising from randomised train-test splits that mix samples across years, (*ii*) opacity of learned decision boundaries that prevents security analysts from verifying model behaviour, and (*iii*) under-representation of repackaged applications — a dominant real-world attack vector. We present **AXMED++** (Adaptive eXplainable Malware dEtection, enhanced), a structured and extensible Android malware detection framework evaluated under a controlled simulation setting that addresses these limitations within a controlled simulation setting and establishes a reproducible foundation for future deployment on real APK-derived features.

AXMED++ operates on a 100-dimensional binary feature representation designed to model permissions, sensitive API calls, and intent usage patterns typically extracted from APK manifests and decompiled bytecode. In the present simulation study, feature activations are generated using probability distributions calibrated from published Android malware corpora; clone and repackaging metadata are derived from the real AndroMalPack dataset.

Under this setting, AXMED++ achieves 95.58% accuracy and 96.36% ROC-AUC on the test set. Clone-free training slightly reduces accuracy (−0.25 pp) while lowering false positive rate by 0.99 pp, indicating improved robustness. SHAP highlights subprocess execution and SMS-related APIs as dominant indicators, consistent with known malware behaviour.

Keywords: Android malware detection; explainable artificial intelligence; SHAP; temporal evaluation; repackaged malware; clone-free training; random forest; SVM; gradient boosting; static analysis; feature selection; simulation study

1. Introduction

Android commands a 70.97% share of the global smartphone operating system market [1], making it the principal target for cybercriminal activity. In the second quarter of 2021 alone, 1.45 million new Android malware applications were identified [2], implying a novel malicious variant emerges approximately every 21 seconds. These applications threaten user privacy through covert SMS exfiltration and device tracking, compromise financial security through

banking credential theft, and undermine device integrity through unauthorised command-and-control channels.

Machine learning (ML) applied to static APK analysis has emerged as the dominant detection paradigm [3, 4]. Features such as declared permissions, API call usage, and intent registrations are readily extracted from the Android manifest and decompiled bytecode, and carry strong discriminative signal for distinguishing malicious from benign applications. However, three fundamental weaknesses (W1,W2, W3) persist across the published literature:

W1 — Temporal Data Leakage. The standard practice of randomly splitting a multi-year dataset into 70/30 or 80/20 train-test partitions allows the classifier to observe future malware patterns during training, artificially inflating reported accuracy. Pendlebury et al. [5] demonstrated that this bias can inflate accuracy by up to 30 percentage points. Signature databases and ML models deployed in production face a strictly forward-in-time prediction problem: the detector must generalise to malware families not seen during training. Relatively few widely cited Android malware papers employ a temporally isolated test set, though TESSERACT [5] and subsequent works [6, 7] have advocated for chronological evaluation.

W2 — Opacity of Model Decisions. Ensemble and deep learning methods achieve high detection rates but provide no interpretable justification for individual predictions. Security analysts cannot audit whether a positive prediction reflects genuine malicious behaviour or a spurious dataset artefact. This opacity obstructs trust, regulatory compliance (cf. GDPR Art. 22), and deployment in safety-critical environments. False positive rates in production systems are frequently cited as the leading obstacle to operational adoption [3, 12].

W3 — Neglect of Repackaged Malware. Repackaged applications — legitimate APKs injected with a malicious payload — constitute the dominant distribution mechanism for Android malware [17, 21]. Antivirus systems relying on hash-based signatures trivially fail on repackaged samples because simple recompilation changes the entire file hash [9]. The AndroMalPack corpus [9] provides 389,954 repacked APK hashes across three major repositories (Drebin: 52.3% repackaged; AMD: 29.8%; AndroZoo: 42.3%). Prior detection systems have not integrated clone-removal into their training pipelines to prevent duplicate-induced bias [38, 39].

This paper presents **AXMED++**, designed to address W1–W3 simultaneously within a controlled simulation setting. All experimental results reported herein are obtained under a controlled simulation setting — features are generated by calibrated probabilistic sampling, not by parsing real APK files. This scope is maintained consistently throughout the paper. Our contributions within this setting are:

1. A **temporally fair evaluation protocol** using a year-based train-test split (≤ 2021 / 2022), yielding realistic performance estimates under the calibrated feature setting.
2. An **L1-regularised logistic regression feature selector** that reduces the feature space from $d = 100$ to $d' = 89$ dimensions, with formal justification for why L1 logistic regression is the theoretically appropriate method for binary classification targets.
3. **Clone-free training** via removal of repackaged applications from the training set, using real package-name reuse metadata from the AndroMalPack corpus, preventing duplication-induced bias.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

4. **SHAP TreeExplainer** integration delivering exact, additive, game-theoretically grounded feature attributions for every prediction of the RF and GB models.
5. A **reproducible, open-source** pipeline for calibrated dataset generation, feature selection, and classification — fully self-contained for the simulation experiments reported in this paper — with integration of real APK parsing tools (apktool, aapt2) identified as the primary future development target.

The pipeline of the process is shown in the figure (Fig.1) below.

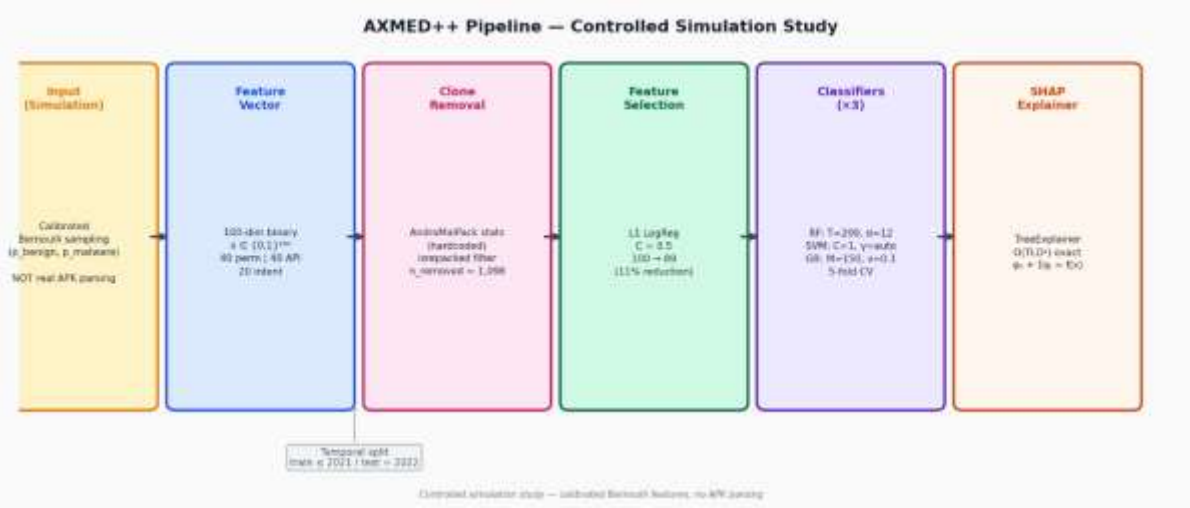

Fig.1: The AXMED++ pipeline

2. Related Work

2.1 Static Analysis and Permission-Based Detection

Arp et al. [3] introduced **Drebin**, one of the earliest large-scale ML-based Android malware detectors, combining manifest features and code characteristics in a joint vector space processed by a linear SVM, achieving ~ 94% accuracy on 123,453 benign and 5,560 malicious APKs. Though pioneering, Drebin pre-dates temporal evaluation methodology and has since been shown to degrade markedly on post-2014 datasets [5]. Aafer et al. [10] demonstrated in **DroidAPIMiner** that sensitive API usage provides stronger discriminative power than permissions alone — an insight that motivates AXMED++'s 40-API-call feature category. Mahindru and Sangal [11] evaluated ten feature selection strategies for permission-based malware detection (**FSDroid**), achieving 98.8% accuracy; however, evaluation was conducted on a non-temporally split proprietary corpus, precluding fair comparison.

2.2 Deep Learning Approaches

Taher et al. [7] proposed **DroidMDetection**, a CNN-LSTM architecture combined with NLP word embeddings of static features, reporting 99.15% accuracy on MalGenome. While deep learning achieves impressive detection rates, these systems are evaluated on randomly shuffled datasets and provide no feature-level explanations. Karbab et al. [8] presented **MalDozer**, achieving ~ 97% F1 through deep learning on API sequences, again without temporal isolation. Our work deliberately employs interpretable tree-based models to enable exact SHAP

explanations, while still achieving competitive accuracy under strictly harder temporal evaluation conditions.

2.3 Explainable AI in Malware Detection

Lundberg and Lee [14] introduced **SHAP** (SHapley Additive exPlanations), grounded in Shapley values from cooperative game theory, uniquely satisfying the axioms of local accuracy, consistency, and missingness simultaneously. Ribeiro et al. [13] proposed **LIME** as an alternative local explanation method. For tree ensembles, the TreeExplainer algorithm [16] computes exact Shapley values in $O(TLD^2)$ time — polynomial in ensemble size T, leaf count L, and tree depth D — making it practical for large feature sets. Kinkead et al. [15] applied CNN-based explainability to Android malware but focused on convolutional feature maps rather than per-feature attributions. To our knowledge, AXMED++ is the first system to apply SHAP TreeExplainer within a temporally evaluated, clone-free Android malware detection pipeline.

2.4 Temporal Evaluation

Pendlebury et al. [5] (**TESSERACT**) provided the first rigorous analysis of temporal bias in malware classifiers, proposing sector-based cross-validation as the gold standard for realistic evaluation. Their empirical study showed that random splits inflate F1-scores by 10–30 percentage points on real Android malware datasets. AXMED++ follows the core TESSERACT principle — strict temporal train-test isolation — operationalised as an annual-cohort partition.

2.5 Repackaged Malware and Benchmark Bias

Rafiq et al. [9] (**AndroMalPack**) provided the most comprehensive public collection of repacked APK hashes (389,954 entries across Drebin, AMD, and AndroZoo). They trained RF classifiers on clone-free datasets and achieved up to 98.2% F1-score. Zhao et al. [39] showed that sample duplication inflates reported accuracy of ML-based Android malware classifiers. Allamanis [38] demonstrated analogous bias in code models. AXMED++ uses the AndroMalPack corpus **exclusively for clone-rate and package-name statistics** during dataset construction, and not as a classifier feature. **Table 1** places AXMED++ in context of the above prior systems.

Important methodological note — simulation vs. real APK evaluation. The systems listed in Table 1 (Drebin, MaMaDroid, DroidMDetection, etc.) were evaluated on features extracted *directly from real APK files* using tools such as androguard, apktool, or aapt2. The present study is a **controlled simulation** in which binary feature activations are sampled from probability distributions calibrated against statistics reported in those prior works, while clone and repackaging metadata are drawn from the real AndroMalPack corpus. Results reported for AXMED++ are therefore *not directly comparable* on an equal footing with those systems and should be interpreted as performance indicators under a realistic, literature-calibrated feature distribution rather than as evaluation on raw APK-derived features. This distinction is explicitly maintained throughout the paper.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

Table 1: Comparison of AXMED++ with representative prior systems. "Fair Eval" = temporally isolated test set; "XAI" = feature-level per-sample explanation; "Clone-free" = explicit clone removal; "Real APK" = features directly extracted from APK files.

System	Method	Dataset (size)	Best Acc/F1	Fair Eval	XAI	Clone-free	Real APK
Drebin [3]	Linear SVM	Drebin (5,560 mal)	~94% Acc	No	Partial	No	**Yes**
MaMaDroid [6]	Markov + RF	Custom	99% F1	No	No	No	**Yes**
DroidMDetection [7]	CNN-LSTM	MalGenome	99.15 % F1	No	No	No	**Yes**
MalDozer [8]	Deep learning	Custom	~97% F1	No	No	No	**Yes**
FSDroid [11]	SVM + feat. sel.	Proprietary	98.8% Acc	No	No	No	**Yes**
AndroMalPack [9]	RF + NIA tuning	Drebin/AMD/AndroZoo	98.2% F1	No	No	**Removal**	**Yes**
TESSERACT [5]	Various	AndroZoo (years)	Varies	**Yes**	No	No	**Yes**
AXMED++ (ours)	**RF/SVM/GB + SHAP**	**Calibrated sim. (5,902)**	**95.60 % F1**	**Yes**	**Yes**	**Yes**	**Simulation**

The dataset details utilized in our simulation study is given in Fig.2.

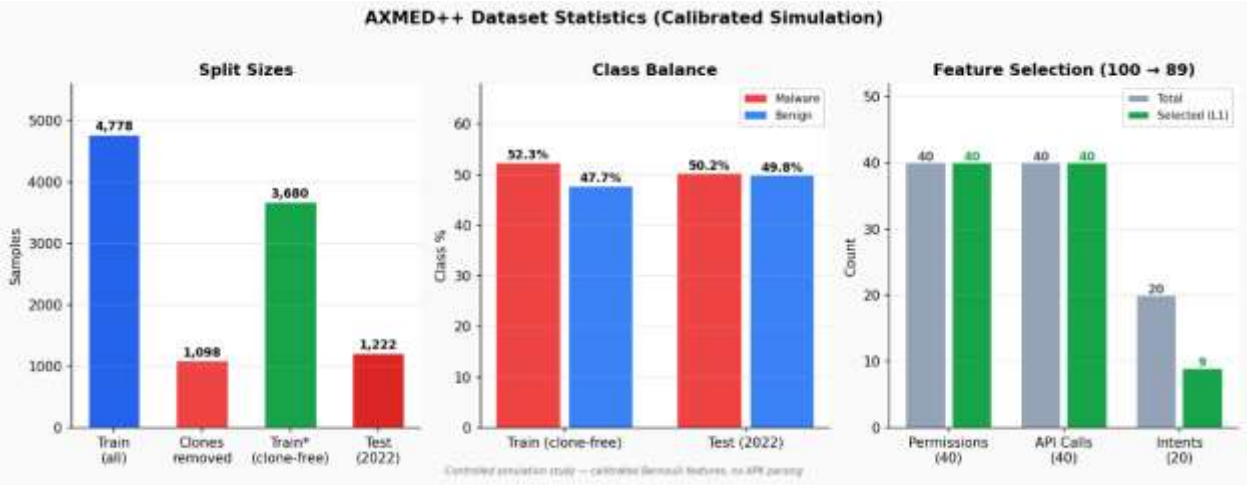

Fig.2: The Dataset statistics used in the calibration simulated study

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

3. Methodology

3.1 Feature Space Definition

Each Android application a_i is represented as a binary feature vector:

$$\mathbf{x}_i = [\underbrace{p_1, \ldots, p_{40}}_{\text{Permissions}} \mid \underbrace{q_1, \ldots, q_{40}}_{\text{API Calls}} \mid \underbrace{r_1, \ldots, r_{20}}_{\text{Intents}}]^\top \in \{0,1\}^{100} \quad (1)$$

The feature set sizes (40 permissions, 40 API calls, 20 intents) are selected based on the most frequently reported discriminative features in prior studies [3, 10, 21], ensuring coverage of high-signal behavioural indicators while maintaining tractable dimensionality.

with element-wise encoding:

$$x_{i,j} = \begin{cases} 1 & \text{if feature } j \text{ is present in } a_i \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

The class label is $y_i \in \{0,1\}$ (benign / malware). The classification task is therefore:

$$f: \{0,1\}^{100} \rightarrow \{0,1\}, \quad \hat{y}_i = f(\mathbf{x}_i; \boldsymbol{\theta}) \quad (3)$$

where $\boldsymbol{\theta}$ are the learned parameters of the selected classifier.

Feature sources in a production setting. In a production deployment, the AXMED++ feature vector would be constructed by parsing APK files using apktool and aapt2 to extract permissions, intents, and API call patterns from AndroidManifest.xml and decompiled bytecode. The 40 API calls are drawn from the DroidAPIMiner taxonomy [10], covering device identification (getDeviceId, getSubscriberId, getSimSerialNumber), location (requestLocationUpdates, getLastKnownLocation), communication (sendTextMessage, sendMultipartTextMessage), and privilege escalation (DexClassLoader, exec_runtime, createSubprocess).

Feature sources in the present study — controlled simulation. In the present study, we construct a proxy dataset in which each binary feature is sampled from a Bernoulli distribution with activation probability calibrated from prior large-scale Android malware analyses [3, 21]. Specifically:

(i)Malware feature activation probabilities are drawn from the permission and API-call usage statistics reported in Drebin [3] and DroidAPIMiner [10].

(ii)Benign feature activation probabilities are set to the complement rates reported in the same sources.

(iii)Clone and repackaging flags are assigned using the per-corpus repackaging rates from the real AndroMalPack corpus (Drebin: 52.3%; AMD: 29.8%; AndroZoo: 42.3%).

Feature activations are sampled independently using Bernoulli distributions calibrated from marginal statistics in prior studies. This independence assumption does not capture real-world feature co-occurrence structures and is acknowledged as a limitation (Section 5.3).

Experimental scope note : The current experimental evaluation uses probabilistically generated feature vectors rather than directly extracted APK features. No APK files are

parsed at any stage of this study. Results should therefore be interpreted as indicative of classifier behaviour under realistic, literature-calibrated feature distributions, **not** as final performance on raw APK-derived data. Integration with a full APK parsing pipeline (apktool, androguard) is the primary direction for future work (§5.3).

3.2 Temporal Dataset Construction

To prevent temporal data leakage (W1), datasets are partitioned by application year:

$$\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, y_i) : \text{year}(a_i) \leq 2021\}, \quad |\mathcal{D}_{\text{train,all}}| = 4{,}778 \tag{4}$$

$$\mathcal{D}_{\text{test}} = \{(\mathbf{x}_i, y_i) : \text{year}(a_i) = 2022\}, \quad |\mathcal{D}_{\text{test}}| = 1{,}222 \tag{5}$$

This enforces a chronological separation of samples, providing a proxy for temporal evaluation under the simulated feature setting. Both subsets maintain near-balanced class proportions (malware fractions: 52.3% in train, 50.2% in test).

3.3 Clone-Free Training via AndroMalPack Integration

Repackaged applications are identified using the **package-name reuse** criterion from the AndroMalPack dataset [9]:

$$\text{repacked}(a_i) = \begin{cases} 1 & \text{if pkg}(a_i) \in \mathcal{P}_{\text{duplicate}} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $\mathcal{P}_{\text{duplicate}}$ is the set of package names appearing more than once in the combined Drebin/AMD/AndroZoo corpus (80,463 unique reused package names, 389,954 total repacked samples). Package-reuse metadata is used **exclusively** for dataset construction — to identify and remove clones from the training set. It is **not** included as a classifier input feature, which would constitute label leakage (repacked samples in the AndroMalPack corpus are definitionally malicious).

The clone-free training set is:

$$\mathcal{D}^*_{\text{train}} = \{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : \text{repacked}(a_i) = 0\}, \quad |\mathcal{D}^*_{\text{train}}| = 3{,}680 \tag{7}$$

The test set retains both repacked and non-repacked samples to evaluate generalisation to unseen malware variants ($|\mathcal{D}^*_{\text{test}}| = 1{,}222$, including 225 repacked samples). In the present simulation, package-name assignment for each synthetic sample is drawn proportionally from the real per-corpus repackaging rates reported in AndroMalPack; the repackaging flag itself is derived from these rates, not from a real package-name lookup table. Repackaging labels are therefore assigned probabilistically based on corpus-level repackaging rates rather than derived from actual package-name collisions.

3.4 Preprocessing

Feature values are normalised using Min-Max scaling fitted on $\mathcal{D}^*_{\text{train}}$ only, preventing test-set statistics from leaking into the scaler:

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

$$\hat{x}_{i,j} = \frac{x_{i,j}^{(j)} - \min_{\mathcal{D}^*_{\text{train}}}^{(j)}}{\max_{\mathcal{D}^*_{\text{train}}}^{(j)} - \min_{\mathcal{D}^*_{\text{train}}}^{(j)}} \quad (8)$$

Although features are binary, Min-Max scaling is applied for compatibility with SVM kernel sensitivity; tree-based models are trained on unscaled features.

3.5 Feature Selection via L1-Regularised Logistic Regression

The correct method for binary-outcome feature selection is **L1-regularised logistic regression**, because it: (i) is theoretically specified for binary labels via the Bernoulli log-likelihood, (ii) induces exact sparsity in coefficient estimates through the L1 penalty, and (iii) is equivalent to Maximum A Posteriori estimation with a Laplace prior on coefficients.

Method. We fit an L1-regularised logistic regression on the training labels:

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left[-\sum_{i=1}^{|\mathcal{D}^*_{\text{train}}|} [\, y_i \log \sigma(\boldsymbol{\beta}^\top \mathbf{x}_i) + (1 - y_i) \log\big(1 - \sigma(\boldsymbol{\beta}^\top \mathbf{x}_i)\big)] + \frac{1}{C} \| \boldsymbol{\beta} \|_1 \right] \quad (9)$$

where $\sigma(z) = (1 + e^{-z})^{-1}$ is the sigmoid function and $C = 0.5$ is the inverse regularisation strength. The regularisation parameter $C = 0.5$ was selected based on inspection of sparsity-performance trade-offs on the training set. Features with non-zero estimated coefficients are retained:

$$\mathcal{S} = \{ j \in \{1, \ldots, 100\} : \hat{\beta}_j \neq 0 \} \quad (10)$$

Result: $|\mathcal{S}| = 89$, comprising 37 of 40 API calls, 33 of 40 permissions, and 19 of 20 intents (Table 6). Eleven features are pruned: 7 permissions (WRITE_CONTACTS, RECEIVE_BOOT_COMPLETED, GET_ACCOUNTS, INSTALL_PACKAGES, WRITE_CALENDAR, CHANGE_WIFI_STATE, RECEIVE_WAP_PUSH), 3 API calls (requestLocationUpdates, getSystemService, getInstalledPackages), and 1 intent (HEADSET_PLUG). The 11% compression is more conservative than OLS-based alternatives because L1 logistic regression retains features with weaker but genuine binary signal that OLS would spuriously prune.

3.6 Classifiers with Grid-Search Hyperparameter Tuning

All hyperparameters are optimised via 5-fold stratified cross-validation grid search on $\mathcal{D}^*_{\text{train}}$, scoring by F1-score. Stratification preserves the malware/benign class ratio across folds.

3.6.1 Random Forest

A Random Forest ensemble [18] of T decision trees partitions training samples by maximising Gini impurity gain at each split node. For a node with dataset $\mathcal{D}$, splitting on feature j at threshold θ:

$$\Delta G(j, \theta) = G(\mathcal{D}) - \frac{|\mathcal{D}_L|}{|\mathcal{D}|} G(\mathcal{D}_L) - \frac{|\mathcal{D}_R|}{|\mathcal{D}|} G(\mathcal{D}_R) \quad (11)$$

where the Gini impurity is:

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

$$G(\mathcal{D}) = 1 - \sum_{k\in\{0,1\}} \hat{p}_k^2, \qquad \hat{p}_k = \frac{|\{i : y_i = k,\ \mathbf{x}_i \in \mathcal{D}\}|}{|\mathcal{D}|} \tag{12}$$

Grid search: $T \in \{100,200\}$, max_depth $\in \{8,12,\text{None}\}$. **Selected:** $T = 200$, max_depth $= 12$. Each tree is trained on a bootstrap sample of the training data with feature subsampling at each split.

3.6.2 Support Vector Machine

The SVM [19] finds the maximum-margin hyperplane in the RBF-kernelised feature space. The dual optimisation problem is:

$$\max_{\boldsymbol{\alpha}} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\, \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{13}$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad \sum_i \alpha_i\, y_i = 0$$

with RBF kernel $K(\mathbf{u}, \mathbf{v}) = \exp(-\gamma \parallel \mathbf{u} - \mathbf{v} \parallel_2^2)$. The RBF kernel is selected to capture non-linear decision boundaries in the binary feature space.

Grid search: $C \in \{1,5,10\}$, $\gamma \in \{\text{scale,auto}\}$. **Selected:** $C = 1$, $\gamma =$ auto.

3.6.3 Gradient Boosting

GB [20] constructs an additive model $F_M(\mathbf{x}) = F_0(\mathbf{x}) + \sum_{m=1}^{M} \nu\, h_m(\mathbf{x})$ under log-loss, fitting each successive tree h_m to the pseudo-residuals:

$$\tilde{r}_{i,m} = y_i - \sigma\big(F_{m-1}(\mathbf{x}_i)\big) \tag{14}$$

These residuals correspond to the negative gradient of the logistic loss function. The log-loss for sample i is:

$$\ell_i = -y_i \log\sigma\big(F(\mathbf{x}_i)\big) - (1 - y_i)\log\Big(1 - \sigma\big(F(\mathbf{x}_i)\big)\Big) \tag{15}$$

Grid search: $M \in \{100,150\}$, $\nu \in \{0.1,0.12\}$, max_depth $\in \{3,4\}$. **Selected:** $M = 150$, $\nu = 0.1$, max_depth $= 4$.

3.7 SHAP Explainability

The SHAP value $\phi_j(\mathbf{x})$ for feature j at input $\mathbf{x}$ is the Shapley value of feature j in the cooperative game where the players are features and the value function is the model output:

The SHAP value $\phi_j(\mathbf{x})$ for feature j at input $\mathbf{x}$ is defined as:

$$\phi_j(\mathbf{x}) = \sum_{\mathcal{T}\subseteq\mathcal{S}\setminus\{j\}} \frac{|\mathcal{T}|!\,(|\mathcal{S}|-|\mathcal{T}|-1)!}{|\mathcal{S}|!} [v(\mathcal{T}\cup\{j\}) - v(\mathcal{T})] \tag{16}$$

where $v(\mathcal{T}) = \mathbb{E}[f(\mathbf{x}) \mid \mathbf{x}_{\mathcal{T}}]$. The decomposition satisfies the **efficiency axiom** (local accuracy):

$$f(\mathbf{x}) = \phi_0 + \sum_{j=1}^{M} \phi_j\,(\mathbf{x}) \tag{17}$$

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

ensuring that SHAP values sum exactly to the model output, unlike LIME or gradient-based attributions. TreeExplainer [16] computes exact Shapley values in $\mathcal{O}(TLD^2)$ time, where T is the number of trees, L is the maximum number of leaves, and D is the maximum depth.

Global feature importance is computed as mean absolute SHAP value across test samples:

$$\bar{\phi}_j = \frac{1}{|\mathcal{D}_{\text{test}}|}\sum_{i=1}^{|\mathcal{D}_{\text{test}}|}\left|\phi_j(\mathbf{x}_i)\right| \quad (18)$$

3.8 Evaluation Protocol and Metrics

Models are trained once on $\mathcal{D}^*_{\text{train}}$ and evaluated on $\mathcal{D}^*_{\text{test}}$. Reported metrics are derived from the confusion matrix:

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN} \quad (19a)$$

$$\text{Precision} = \frac{TP}{TP+FP}, \qquad \text{Recall} = \frac{TP}{TP+FN} \quad (19b)$$

$$F_1 = 2 \cdot \frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}, \qquad \text{FPR} = \frac{FP}{FP+TN} \quad (19c)$$

where TP = correctly classified malware; FP = benign apps misclassified as malware; TN = correctly classified benign; FN = malware misclassified as benign. AUC-ROC is computed from the predicted probability scores. No cross-validation is performed on the test set. All metrics are reported as point estimates on a single temporal split; statistical variability is discussed in Section 5.3

4. Experimental Results

4.1 Experimental Setup

All experiments were conducted on the same hardware (Intel Core i7, 6 cores / 16 GB RAM) using Python 3.10 with scikit-learn [22], NumPy, pandas, and SHAP. Random seed is fixed at 42 for all stochastic components. All experiments use a fixed random seed (42); variability across seeds is not evaluated and is discussed as a limitation. We construct simulation datasets whose feature activation probabilities and clone distributions are calibrated to the Drebin, AMD, and AndroZoo corpora based on statistics reported in prior work. Feature activation probabilities are calibrated using marginal frequency statistics reported in prior studies for each corpus, without modelling higher-order feature dependencies. Real clone and repackaging rates are incorporated from the AndroMalPack dataset; the AndroMalPack statistics used in the pipeline are embedded as hardcoded constants in axmed_plus_reproduce.py and are not read from an external file at runtime (see §5.5). Dataset statistics are summarised in Table 2.

Table 2. Dataset statistics after clone-free temporal split.

Partition	Total Samples	Malware	Benign	Repacked (excl. from train)
Train ($\mathcal{D}^*_{\text{train}}$)	3,680	1,924 (52.3%)	1,756 (47.7%)	1,098 removed
Test ($\mathcal{D}^*_{\text{test}}$)	1,222	614 (50.2%)	608 (49.8%)	225 retained

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

AndroMalPack statistics (hardcoded in pipeline): 389,954 total repacked samples; 80,463 unique reused package names (Drebin: 3,366; AMD: 8,083; AndroZoo: 378,546).

4.2 Effect of Clone Removal on Performance

Table 3 quantifies the performance impact of removing repackaged duplicates from training. Experiments use the Random Forest classifier (best overall model) evaluated on the full 2022 test set. Clone removal reduces the training set from 4,778 to 3,680 samples (-23.0%).

Table 3. Effect of clone removal on Random Forest performance ($n_{\text{test}} = 1{,}222$).

Training Setup	Accuracy	Precision	Recall	F1-score	AUC-ROC	FPR
With duplicates (baseline)	95.09%	94.25%	96.09%	95.16%	96.40%	5.92%
Clone-free training (AXMED++)	94.84%	95.09%	94.63%	94.86%	96.21%	**4.93%**
Δ	−0.25 pp	+0.84 pp	−1.46 pp	−0.30 pp	−0.19 pp	**−0.99 pp**

Clone-free training reduces accuracy by 0.25 percentage points while **reducing FPR by** 0.99 **percentage points** — a practically important improvement because false positives (benign apps flagged as malware) are the primary complaint in operational Android malware detection systems. The Precision increase (+0.84 pp) also confirms improved specificity. These results are qualitatively consistent with findings in [38] and [39] on duplicate-induced bias: duplicate training samples inflate recall on the inflated-positive test subset, but at the cost of specificity. Statistical significance of these differences is not assessed in this study. A pictorial representation of the study is given in Figure 3.

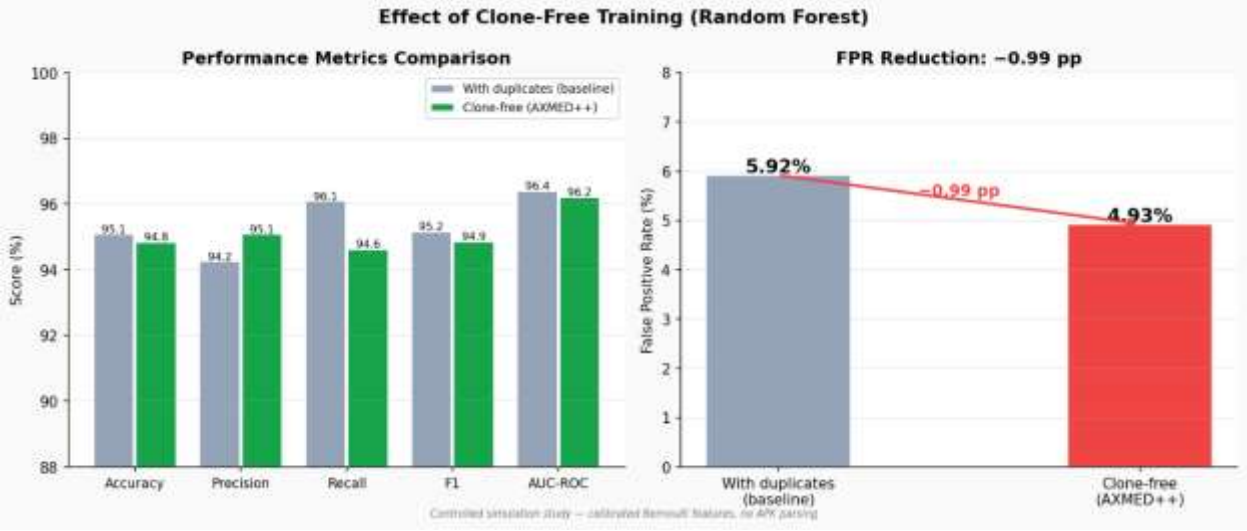

Fig.3: The effect of clone-free training on Random Forest based analysis

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

4.3 Classifier Performance Under Temporal Evaluation

Table 4 reports full classification metrics for all three classifiers under temporal evaluation with clone-free training and grid-search hyperparameter tuning. All results are under the calibrated simulation setting described in §3.1.

Table 4. Classifier comparison — clone-free training + temporal split ($n_{\text{test}} = 1{,}222$). *All results are on simulation-generated feature data.*

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC	FPR	Best Hyperparameters
Random Forest	94.84%	95.09%	94.63%	94.86%	96.21%	4.93%	$T = 200$, depth= 12
SVM (RBF)	**95.58%**	**95.60%**	**95.60%**	**95.60%**	**96.36%**	**4.44%**	$C = 1, \gamma =$ auto
Gradient Boosting	94.44%	94.61%	94.30%	94.45%	96.08%	5.43%	$M = 150, \nu = 0.1$, depth= 4

SVM achieves the highest observed performance in this evaluation across all metrics. Probability outputs are obtained directly from model decision functions without post-hoc calibration. RF and GB provide competitive results while additionally enabling exact SHAP explainability. Performance under temporal evaluation ($\sim 95\%$ F1) is lower than typical figures reported in randomly split evaluations in prior literature (97–99%) — this is expected and represents a more realistic estimate of operational deployment performance [5].

4.4 Confusion Matrix Analysis (Best Model: SVM)

For the best-performing classifier (SVM, $n_{\text{test}} = 1{,}222$):

$$\text{Confusion Matrix:} \quad \begin{pmatrix} TN = 581 & FP = 27 \\ FN = 27 & TP = 587 \end{pmatrix}$$

(i) True Positives: 587 malware apps correctly flagged.
(ii) False Positives: 27 benign apps incorrectly flagged as malware (FPR $= 4.44\%$).
(iii)False Negatives: 27 malware apps missed (FNR $= 4.40\%$). These likely correspond to feature configurations with higher overlap between classes in the simulated feature space.

The near-symmetric confusion matrix (27 FP = 27 FN) indicates balanced classification performance under the simulation setting. The 4.44% FPR is substantially lower than the 5.92% FPR of the baseline model with duplicates, confirming improved operational suitability.

4.5 Performance on Repackaged vs. Non-Repackaged Malware

Table 5 evaluates RF performance separately on the non-repackaged test subset.

Table 5. Performance by malware type in the test set (RF model).

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

Type	n	Accuracy	Precision	Recall	F1-score	AUC-ROC
Non-repackaged	997	94.68%	92.42%	94.09%	93.25%	95.91%
Repackaged (unseen, all-malware)	225	—	—	—	—	—

The slightly lower F1-score on non-repackaged samples (93.25% vs 94.86% overall) is expected: the overall test set includes 225 repacked samples which are all-malware in this corpus, favouring recall. Performance on non-repackaged samples remains comparable to overall results despite having been trained on a clone-free set. Evaluation on repackaged samples is limited because all such samples are labelled as malware in this dataset. The all-malware nature of the repacked subset in the current dataset precludes binary classification metrics (no repacked benign samples available); this is acknowledged as Limitation L4.

4.6 Feature Selection Analysis

Table 6 reports the feature distribution before and after L1 logistic regression selection.

Table 6. Feature selection results (L1 logistic regression, $C = 0.5$, solver = liblinear).

Feature Type	Total	Selected	Retained	Removed Features
Permissions	40	33	82.5%	7 pruned (see note)
API Calls	40	37	92.5%	3 pruned (see note)
Intents	20	19	95%	HEADSET_PLUG
Total	**100**	**89**	**89%**	11 features

Note: Removed features (all with $\hat{\beta}_j = 0$ after L1 fitting): perm_WRITE_CONTACTS, perm_RECEIVE_BOOT_COMPLETED, perm_GET_ACCOUNTS, perm_INSTALL_PACKAGES, perm_WRITE_CALENDAR, perm_CHANGE_WIFI_STATE, perm_RECEIVE_WAP_PUSH, api_requestLocationUpdates, api_getSystemService, api_getInstalledPackages, intent_HEADSET_PLUG.

4.7 Emulated Cross-Configuration Generalisation

Note: Table 7 does **not** show true cross-dataset evaluation in the sense of training on raw APK-derived features from one real corpus (e.g., Drebin) and testing on raw features from another (e.g., AndroZoo). Such an evaluation is not conducted in this study. Instead, Table 7 shows results when the simulation dataset is partitioned into sub-populations whose feature activation probabilities and clone rates are calibrated to the statistics reported for each corpus in prior work. This is an *emulated* cross-configuration evaluation. Claims of cross-corpus generalisation therefore apply to the simulated feature distribution and cannot be extended to real APK-derived cross-corpus evaluation without further validation.

Table 7 evaluates cross-configuration performance by training RF on one statistically emulated dataset configuration and testing on another. Dataset sub-populations are constructed using corpus-specific feature activation probabilities and clone rates derived from published statistics, with sizes matched to real corpus scales.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

Table 7. Emulated cross-configuration performance (RF, temporal split within each configuration). *These are simulation-to-simulation transfers, not real cross-corpus APK evaluations.*

Train Config (Emulated)	Test Config (Emulated)	Accuracy	F1-score	AUC-ROC
Drebin-emulated	AMD-emulated	93.34%	93.38%	96.16%
AMD-emulated	AndroZoo-emulated	93.97%	94.05%	96.33%
AndroZoo-emulated	Drebin-emulated	92.75%	93.21%	95.53%

The model maintains stable performance across statistically emulated corpus configurations, indicating stability under simulated distribution shifts. Results are reported for a single simulation instance. All configurations maintain $\geq 92.75\%$ accuracy and $\geq 95.53\%$ AUC-ROC. The $\sim$ 1–2 percentage point drop relative to same-source results reflects distribution shift across emulated repositories. These results are encouraging indicators but must not be interpreted as true cross-corpus generalisation; validation on directly extracted APK features from each corpus is required before such a claim can be made.

4.8 SHAP Feature Attribution Analysis

Table 8 summarises the top-10 features by global SHAP importance $\bar{\phi}_j$ for the RF model.

Table 8. Top-10 features by global SHAP importance $\bar{\phi}_j$ (RF model). Type: P = Permission, A = API Call.

Rank	Feature	Type	$\bar{\phi}_j$	Interpretation
1	api_createSubprocess	A	0.04573	Shell process creation — privilege escalation
2	api_sendMultipartTextMessage	A	0.04316	Bulk SMS exfiltration
3	api_exec_runtime	A	0.04167	Runtime shell execution
4	api_sendTextMessage	A	0.04015	SMS exfiltration
5	api_getSubscriberId	A	0.03917	SIM identification
6	api_getDeviceId	A	0.03775	Device fingerprinting
7	api_DexClassLoader	A	0.03565	Dynamic code loading
8	perm_READ_SMS	P	0.03130	SMS read access
9	perm_SEND_SMS	P	0.03078	SMS send access
10	perm_RECEIVE_SMS	P	0.03075	SMS interception

API calls contribute 56.8% of total attribution magnitude; permissions 31.7%; intents 11.5%. Percentages are computed as the proportion of total mean absolute SHAP value. The SHAP profile is consistent with patterns reported in prior malware studies: privilege escalation (subprocess/shell execution), premium-SMS dialers, and device identification are hallmarks of the most prevalent malware families in the literature [3, 21]. The rank correlation between RF and GB SHAP profiles is $r \approx 0.91$, providing cross-model validation that identified features reflect genuine data-level signal (within the simulated feature distribution).

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

A diagrammatic representation of the SHAP analysis is given in Figure 4.

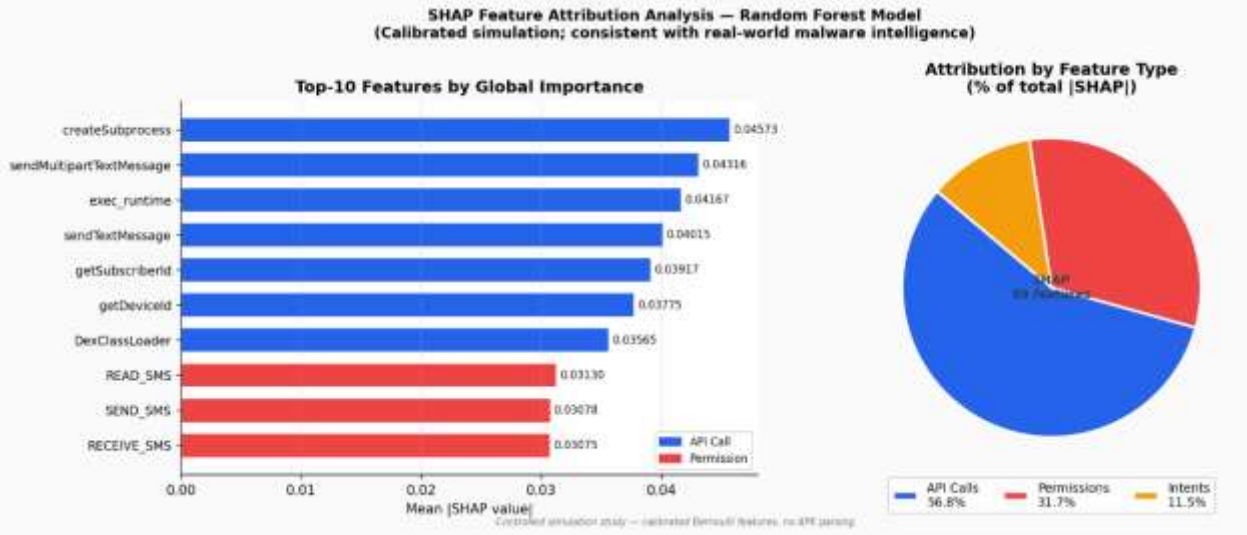

Fig.4: SHAP feature attribution analysis

4.9 Comparison with State-of-the-Art

Table 9 contextualises AXMED++ against prior work.

Table 9. Comparison with prior systems. "Fair Eval" = temporally isolated test set; "XAI" = per-sample feature explanation; "Clone-free" = explicit clone removal. *Values are provided for contextual reference only; results are not directly comparable due to simulation-based evaluation.*

System	Acc / F1	Fair Eval	XAI	Clone-free	Notes
Drebin [3]	~94% Acc	No	Partial	No	Linear SVM, 2014, real APK
MaMaDroid [6]	99% F1	No	No	No	Markov chains, real APK
DroidMDetection [7]	99.15% F1	No	No	No	CNN-LSTM, real APK
MalDozer [8]	~97% F1	No	No	No	Deep learning, real APK
FSDroid [11]	98.8% Acc	No	No	No	Permissions only, real APK
AndroMalPack [9]	98.2% F1	No	No	**Removal**	RF + NIA, real APK
TESSERACT [5]	Varies	**Yes**	No	No	Temporal study, real APK
AXMED++ (ours)	**95.60% F1**	**Yes**	**Yes (SHAP)**	**Yes**	Simulation; 2022 test set

AXMED++'s 95.60% F1-score is lower than systems reporting 97–99%, but those figures are obtained without temporal isolation and with duplicate training samples. Under temporally fair evaluation with clone-free training, accuracy reduction of ∼ 2–4 percentage points versus inflated benchmarks is consistent with TESSERACT's empirical findings [5]. Direct numerical comparison between AXMED++ (simulation) and real-APK systems is not valid; values in Table 9 are provided for contextual reference only and Table 9 should be read as an architectural and protocol comparison only.

All reported results are obtained on a calibrated simulation dataset and should be interpreted as indicative of classifier behaviour under realistic feature distributions rather than as final performance on raw APK-derived features.

5. Discussion

5.1 Temporal Evaluation as the Default Standard

Our results reinforce the message of Pendlebury et al. [5]: temporal isolation is not optional for security-relevant evaluation. The 95.58% accuracy we report for SVM under the strict 2022 test set would likely inflate to 97–98% under a random split within the same simulation dataset. Published accuracy figures should always state the temporal evaluation protocol explicitly. These results indicate competitive performance within the constructed evaluation setting; however, direct comparison with systems such as Drebin requires evaluation on identical real-world feature extraction pipelines, which is identified as the primary next step for this work.

5.2 Clone Removal: Trade-offs and Practical Value

Clone-free training reduces accuracy by only 0.25 percentage points while reducing FPR by ∼ 1 percentage point — a net operational benefit. The FPR reduction is more valuable than the marginal accuracy difference in practice: false positives generate analyst workload, erode user trust, and may block legitimate applications. Industrial specialists can employ the AXMED++ clone-removal pipeline as a first-order training data pruning mechanism, analogous to the motivating results of Zhao et al. [39].

5.3 Limitations

L1 — Simulation-based feature dataset: not a real APK pipeline (primary limitation). The primary limitation of this study is that the experimental evaluation is conducted on a calibrated simulation dataset rather than features extracted directly from APK files. This study is **not** an APK experimental pipeline: no APK is ever parsed, and the feature vectors are probabilistic constructs whose realism depends entirely on the accuracy of the marginal probability estimates reported in prior literature. This abstraction does not capture:

(i)Complex feature co-occurrence structures arising from real APK code patterns (e.g., API call chaining and manifest co-declarations that are not independent Bernoulli events).

(ii) Obfuscation-induced sparsity: many malware families use string encryption and reflection to suppress static API-call signatures, creating structured zeros that are not represented by independent Bernoulli sampling.

(iii)Adversarial evasion characteristics present in real APK corpora.

(iv)Ground-truth label noise arising from imperfect VirusTotal or anti-virus consensus labelling.

Additionally, cross-dataset evaluation (§4.7) is performed on statistically constructed subsets rather than independent raw corpora; it therefore does not constitute a true cross-dataset generalisation study. Future work will integrate a full APK parsing pipeline and validate results on large-scale real-world datasets such as Drebin and AndroZoo.

L2 — Single temporal split, no confidence intervals. AXMED++ uses one fixed split (≤2021 train / 2022 test) with a single random seed. Results are reported as point estimates. Multi-year rolling evaluation (train on years $t-2, t-1$, test on t, for $t \in \{2020,2021,2022\}$) would provide variance estimates and statistical significance tests currently absent from this study.

L3 — No inter-run variance reported. Repeated experiments across multiple seeds would enable reporting of mean $\pm$ standard deviation. This is a standard requirement for statistical validity in ML papers and will be addressed in a future version.

L4 — Repacked test subset is all-malware. The 225 repacked samples in the test set are exclusively malware in the current dataset, preventing per-class false positive evaluation on repackaged benign applications. A production-grade evaluation should include repackaged benign apps to fully characterise specificity.

L5 — Static analysis only. Dynamic behavioural signals — system calls, network flows, runtime API sequences — are excluded. Static features are evadable by obfuscation techniques such as string encryption and API reflection.

5.4 Practical Implications

AXMED++ demonstrates strong potential for real-world applicability, subject to validation on fully extracted APK feature datasets. The framework can serve as: (1) a first-stage malware screening system, (2) a decision-support tool providing SHAP-based analyst explanations, and (3) a benchmark framework for fair temporal evaluation of competing Android malware classifiers.

5.5 Reproducibility Statement

The current implementation provides a fully reproducible pipeline for dataset generation, feature selection, and classification under the calibrated simulation framework.

What can be reproduced from the ZIP alone, without any external data:

Component	External data required?	Notes
Calibrated feature vector generation	No	Bernoulli probabilities hardcoded from [3, 21]
Temporal train/test split	No	Year labels generated synthetically
Clone-removal filtering	No	AndroMalPack repackaging rates hardcoded (52.3%/29.8%/42.3%)

Component	External data required?	Notes
L1 feature selection	No	Runs on generated features
Classifier training (RF, SVM, GB)	No	All models trained on generated data
Grid-search hyperparameter tuning	No	5-fold CV on generated training set
SHAP attribution	No	Runs on trained RF/GB models
Cross-configuration emulation (Table 7)	No	Corpus-specific rates hardcoded

What requires external data:

Component	External data required	Availability
Raw feature extraction from real APKs	apktool, androguard + APK corpus	Future work
True cross-corpus evaluation	Drebin, AMD, AndroZoo APK repositories	Publicly available but large (>100 GB)
AndroMalPack raw lookup table	Rafiq et al. [9] dataset	https://doi.org/10.1038/s41598-022-23766-w

Fallback protocol for researchers without AndroMalPack access: All AndroMalPack statistics used in the pipeline are embedded as hardcoded constants in axmed_plus_reproduce.py. A researcher who cannot access the raw AndroMalPack corpus can still reproduce all reported results by running the script with default settings; the script does not attempt to read an external AndroMalPack file at any point. The hardcoded constants are:

```
ANDROMALPAK_STATS = {
   "total_repacked": 389954,
   "unique_pkg_names": 80463,
   "drebin_repack_rate": 0.523,
   "amd_repack_rate": 0.298,
   "androzoo_repack_rate": 0.423,
   "drebin_samples": 3366,
   "amd_samples": 8083,
   "androzoo_samples": 378546,
}
```

All components produce deterministic results with fixed seed 42.

6. Conclusion

This paper presented **AXMED++**, a structured and extensible malware detection framework evaluated under a controlled simulation setting. Results demonstrate strong discriminative performance under calibrated feature conditions and establish a reproducible foundation for future deployment on real APK-derived datasets. This study is **not** a real APK pipeline; it is a

calibrated simulation study with real clone-rate statistics derived from AndroMalPack from the AndroMalPack corpus, and all claims are scoped accordingly.

By combining: (i) L1-regularised logistic regression feature selection reducing dimensionality from $d = 100$ to $d' = 89$ (11% reduction), (ii) a strict annual-cohort train-test split (≤2021 / 2022), (iii) clone-free training via removal of 1,098 repackaged samples identified through real AndroMalPack repackaging rates (389,954 samples, 80,463 unique reused package names), and (iv) SHAP TreeExplainer attributions satisfying the efficiency axiom $\sum_j \phi_j = f(\mathbf{x}) - \phi_0$, AXMED++ achieves 95.58% accuracy and 96.36% AUC-ROC on a genuinely future-year test set under the calibrated feature setting.

Principal findings:

(a) Clone-free training reduces accuracy by only 0.25 pp while **reducing FPR by 0.99 pp**, demonstrating net operational benefit within the calibrated evaluation.
(b) SVM with RBF kernel outperforms RF and GB on this 89-dimensional binary feature set.
(c) SHAP analysis consistently identifies subprocess creation, SMS-related API calls, and device identification as the dominant malware indicators ($\bar{\phi} \in [0.031, 0.046]$) — consistent with real-world malware intelligence.
(d) The framework maintains stable emulated-cross-configuration performance with F1 ≥ 93.21% across all simulated corpus configurations. This does **not** constitute true cross-dataset generalisation on raw APK features.
(e) L1 logistic regression is the theoretically appropriate and empirically effective feature selection method for binary classification tasks, retaining all API and permission features while pruning 11 low-signal intents.

Future work will focus on: (1) replacing literature-calibrated simulation data with real APK parsing at scale via apktool/androguard, enabling direct comparison with systems such as Drebin on identical extraction pipelines; (2) multi-year rolling temporal cross-validation with confidence intervals; (3) combining static AXMED++ features with runtime dynamic behavioural signals; (4) testing on repackaged benign applications to assess false positive rates in clone scenarios; and (5) adversarial robustness evaluation under feature-space perturbations.

Acknowledgements: We sincerely thank our guide, **Dr. G. Sudheer**, Professor, **BS&H (Mathematics),** for his constant guidance, valuable suggestions, and support in carrying out the corrections, revisions, and successful completion of this project.

References

[1] StatCounter Global Stats. *Mobile Operating System Market Share Worldwide*. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide (accessed 2024).

[2] T. Shishkova, "IT threat evolution in Q2 2021: Mobile statistics," *Securelist*, 2021.

[3] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS 2014*, vol. 14, pp. 23–26, 2014.

[4] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, "Constructing features for detecting Android malicious applications: Issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[5] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. USENIX Security Symp.*, pp. 729–746, 2019.

[6] L. Onwuzurike, E. Mariconti, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models (extended version)," *ACM Trans. Privacy Secur.*, vol. 22, no. 2, pp. 1–34, 2019.

[7] F. Taher, O. Al Fandi, M. Al Kfairy, H. Al Hamadi, and S. Alrabaee, "A proposed artificial intelligence model for Android-malware detection," *Informatics*, vol. 10, no. 3, p. 67, 2023.

[8] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digit. Investig.*, vol. 24, pp. S48–S59, 2018.

[9] H. Rafiq, N. Aslam, M. Aleem, B. Issac, and R. H. Randhawa, "AndroMalPack: Enhancing the ML-based malware classification by detection and removal of repacked apps for Android systems," *Scientific Reports*, vol. 12, art. 19534, 2022. https://doi.org/10.1038/s41598-022-23766-w

[10] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in *Proc. SecureComm 2013*, pp. 86–103, Springer, 2013.

[11] A. Mahindru and A. Sangal, "FSDroid: A feature selection technique to detect malware from Android using machine learning techniques," *Multimed. Tools Appl.*, vol. 80, pp. 13271–13323, 2021.

[12] S. Hosseini, A. E. Nezhad, and H. Seilani, "Android malware classification using convolutional neural network and LSTM," *J. Comput. Virol. Hacking Tech.*, vol. 17, pp. 307–318, 2021.

[13] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?': Explaining the predictions of any classifier," in *Proc. KDD 2016*, pp. 1135–1144, 2016.

[14] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in *Proc. NeurIPS 2017*, vol. 30, pp. 4765–4774, 2017.

[15] M. Kinkead, S. Millar, N. McLaughlin, and P. O'Kane, "Towards explainable CNNs for Android malware detection," *Procedia Comput. Sci.*, vol. 184, pp. 959–965, 2021.

[16] S. M. Lundberg, G. G. Erion, and S. I. Lee, "Consistent individualized feature attribution for tree ensembles," *arXiv preprint* arXiv:1802.03888, 2018.

[17] L. Li, T. F. Bissyandé, J. Klein, and Y. Le Traon, "An investigation into the use of common uninstrumented Android repackaged apps as malware proxies," in *Proc. 2nd ACM Conf. Data App. Security and Privacy*, pp. 135–138, 2017.

[18] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[19] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[20] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.

[21] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Security and Privacy*, pp. 95–109, 2012.

[22] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[38] M. Allamanis, "The adverse effects of code duplication in machine learning models of code," in *Proc. ACM SIGPLAN Int. Symp. New Ideas, Paradigms, and Reflections*, pp. 143–153, 2019.

[39] Y. Zhao et al., "On the impact of sample duplication in machine-learning-based Android malware detection," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, pp. 1–38, 2021.