International Journal of Innovative Research in Engineering
Volume 7, Issue 4 (July - August 2026), PP: 92-101.
https://www.doi.org/10.59256/ijire.20260704013
www.theijire.com
ISSN No: 2582-8746

Android Malware Detection Using LightGBM with Intelligent Threat Attribution

Murapala Pavitra[1], Dr. L. Sumalatha[2]
[1]PG Scholar, Department of Computer Science and Engineering, University College of Engineering Kakinada, JNTUK, Kakinada, Andhra Pradesh, India.
[2]Professor, Department of Computer Science and Engineering, University College of Engineering Kakinada, JNTUK, Kakinada, Andhra Pradesh, India.

How to cite this paper:
Murapala Pavitra[1], Dr. L. Sumalatha[2], "Android Malware Detection Using Light GBM with Intelligent Threat Attribution", IJIRE-V7I4-92-101.

Abstract: *The Android platform accounts for the overwhelming majority of mobile devices in use worldwide, and that dominance has made it the principal target for authors of mobile malware [1], [2]. Applications distributed outside curated stores routinely abuse the permission model, conceal payloads behind obfuscation, or masquerade as repackaged versions of trusted banking and utility software. Detection schemes built on static signatures cannot follow threats that mutate between releases, and while machine learning classifiers have improved raw detection rates considerably, most of them return a bare verdict that an analyst has no means of interrogating. This paper describes a static analysis framework for Android malware detection that couples a LightGBM classifier with SHapley Additive exPlanations and a mapping layer built on the MITRE ATT&CK knowledge base. Features are drawn exclusively from the application manifest, which keeps extraction inexpensive and avoids the provenance artefacts observed when heterogeneous feature families are combined. Trained and evaluated on the AndroMD dataset and benchmarked against XGBoost, Random Forest and CatBoost under identical conditions, the proposed model attained 99.13% accuracy, a 99.84% ROC-AUC and a Matthews correlation coefficient of 98.26% while completing training in 27.84 seconds, between forty and fifty-nine per cent faster than the competing ensembles. Every prediction is accompanied by a per-sample attribution identifying the permissions responsible for it, and samples judged malicious are additionally mapped onto adversary tactics and techniques. The pipeline is delivered as a Flask application, APKGuard, which accepts an uploaded package and returns a scored verdict, an explanation and a behavioural summary in a single pass.*

Key Words: *Android malware detection; LightGBM; static analysis; explainable artificial intelligence; SHAP; MITRE ATT&CK.*

I.INTRODUCTION

Mobile handsets have absorbed a remarkable range of responsibilities over the past decade. A single device now holds banking credentials, government identity documents, two-factor authentication codes, private correspondence and continuous location history. Android carries the larger share of that burden simply because it runs on the majority of handsets sold, particularly in price-sensitive markets where alternatives are scarce [1]. The characteristics that made the platform attractive to developers, namely an open distribution model, the ability to install packages from arbitrary sources and a permissive scheme for inter-application communication, are the same characteristics that attackers have learned to exploit [2].

The earliest generation of Android malware was crude by present standards. Repackaged copies of popular applications were uploaded to third-party markets with a small malicious component grafted on, typically to dial premium-rate numbers or forward the contact list to a remote server. Such samples were straightforward to identify once a single instance had been analysed, because the injected code was identical across every copy in circulation. Contemporary families bear little resemblance to that pattern. Banking trojans encrypt their strings, resolve method names reflectively at runtime, download the operative payload only after passing environment checks intended to detect emulators, and communicate with infrastructure whose addresses are themselves obtained dynamically. Ransomware variants targeting Android encrypt user media and lock the device interface. Commercial packing services, marketed ostensibly as intellectual property protection, are readily repurposed to frustrate static inspection of malicious packages [2].

Signature matching, which underpinned the first generation of mobile antivirus products, is structurally incapable of addressing this situation. A signature records a property of a specific artefact, so any modification that alters the hashed region defeats it. Recompiling with a different obfuscation seed is sufficient. The consequence is a detection scheme that performs well against catalogued threats and provides essentially no protection against anything encountered for the first time, which is precisely the case that matters most in practice.

Machine learning offered an escape from that constraint by shifting the object of analysis from the binary itself to abstracted properties of the application [3]. Permissions declared in the manifest, framework methods invoked, intent filters registered and the structure of the component hierarchy all provide signal that is considerably more stable across variants than any byte sequence [4], [5]. Ensemble methods based on decision trees have proven particularly effective on this kind of sparse, high-dimensional and largely binary feature data, and a substantial body of published work reports detection accuracies above ninety-five per cent using such approaches [3], [6].

A difficulty remains that the accuracy figures obscure. Nearly all of these systems function as closed predictors: an application enters, a label emerges, and nothing accompanies that label to explain it. For an automated store-side filter operating at scale this may be tolerable. For a security team deciding whether to remove an application already deployed across a corporate fleet it is not. An analyst confronted with a flagged internal application needs to know what prompted the classification in order to judge whether the alert is genuine, and an analyst confronted with a sample that passed inspection but later proved malicious needs to know why the model was misled. Neither question can be answered by a confidence score alone.

A second and less frequently discussed shortcoming is that even a correct verdict conveys very little operational information. Knowing that an application is malicious does not indicate whether it harvests credentials, establishes persistence across reboots, exfiltrates files to external infrastructure, or performs some combination of these. Incident response requires that behavioural picture, and constructing it manually demands reverse engineering effort that most organisations cannot allocate to every alert.

The framework presented in this paper addresses both concerns within a single pipeline. A LightGBM classifier trained on manifest-derived permission features performs detection. SHapley Additive exPlanations decompose each individual prediction into per-feature contributions, so that the permissions responsible for a specific verdict are surfaced rather than a global ranking averaged over the training set. Samples classified as malicious are then mapped onto tactics and techniques catalogued in the MITRE ATT&CK framework, converting the verdict into a structured behavioural profile expressed in vocabulary that security teams already use. The complete pipeline runs as a web application that accepts an uploaded package and returns all three outputs together.

The principal contributions of this work are as follows. First, a permission-only feature strategy is adopted deliberately rather than by convenience, following preliminary experiments in which combining heterogeneous feature families produced accuracy gains that proved on inspection to be artefacts of how the underlying corpus was assembled. Second, explainability is implemented as a post-hoc layer over an unmodified classifier, which allows interpretability to be obtained without surrendering any predictive performance, contrary to the trade-off frequently assumed in the literature. Third, the detection output is connected to an established threat intelligence taxonomy, an integration that appears not to have been reported in prior Android malware detection work. Fourth, the entire system is implemented and deployed rather than left as an offline evaluation, including a manifest recovery routine developed specifically in response to packed samples that defeat conventional parsing.

The remainder of the paper is organised as follows. Section II reviews related work across static, dynamic, deep learning and explainable detection approaches. Section III sets out the proposed methodology. Section IV describes the experimental setup. Section V presents and discusses the results, including error analysis and limitations. Section VI concludes and identifies directions for further work.

II. RELATED WORK

A. Static Feature Based Detection

Static analysis has dominated recent Android malware research, principally because it avoids the operational cost and containment risk of executing an untrusted sample [7]. Arp and colleagues established much of the template with DREBIN [8], which assembled permissions, API calls, hardware requests and network addresses into a joint vector space and demonstrated that a linear classifier over such features could detect malware on-device with explanations attached to each decision. The explanation mechanism in DREBIN derives from the linearity of the model, an approach that does not extend to the non-linear ensembles that subsequently proved more accurate.

A recurring theme in this line of work is that not every declared permission carries useful signal. Li and colleagues [4] showed with SigPID that a small subset of permissions accounts for most of the discriminative power, reducing analysis time by a large factor without materially lowering detection accuracy. This finding directly motivates the variance-based pruning adopted in the present work. Peiravian and Zhu [5] combined permissions and API calls under conventional classifiers, an early demonstration that these two feature families are individually informative and jointly stronger.

Subsequent work has largely explored variations on the feature set. Saied and Thanoon [10] combined permissions, API calls, intents and manifest attributes across several conventional classifiers, reporting strong aggregate accuracy while conceding that obfuscated samples remain problematic and that their models offer no account of individual decisions. Museeb and co-authors [11] restricted attention to API calls and permissions with a Random Forest classifier, confirming that tree ensembles scale comfortably to large corpora assembled from multiple sources. Feizollah and colleagues [12] examined intent filters specifically and found that intents carry discriminative signal complementary to permissions, an observation that argues for feature diversity but also raises the provenance concern addressed later in this paper. Prasad and co-workers [13] introduced, the AndroMD framework, pairing permission scanning with lightweight source-level analysis; their dataset underpins the present study, though their requirement for source code limits applicability where only a compiled package is available.

B. Dynamic and Hybrid Approaches

Because static features cannot capture behaviour that only manifests at execution, several groups have pursued dynamic instrumentation [7]. Zhou and colleagues [14] combined a modified Zebra Optimisation Algorithm with LightGBM over runtime behavioural traces for multi-class family attribution, achieving competitive accuracy at the cost of sandbox execution for every sample under test. Yerima and Sezer [15] proposed DroidFusion, a multilevel scheme that combines several base classifiers through ranked aggregation and reports improved robustness relative to any constituent model. Hybrid designs such as SAMADroid [9] pair static and dynamic views to raise resilience against obfuscation. The recurring pattern across this literature is that dynamic and hybrid designs purchase resilience against obfuscation with substantially increased analysis latency, which conflicts directly with deployment in an interactive screening tool.

C. Deep Learning Architectures

A parallel line of enquiry has applied neural architectures in order to eliminate manual feature engineering. McLaughlin and colleagues [16] applied convolutional networks directly to opcode sequences extracted from application bytecode, learning discriminative n-gram patterns without explicit specification. Ma and co-authors [17] pursued a deliberately lightweight design intended for on-device deployment, trading some accuracy for reduced inference cost. Sun and colleagues [18] extended the trend to transformer architectures with DetectBERT, learning application-level representations that improve generalisation to previously unseen families. These approaches consistently demand larger training corpora and considerably greater computational resources than gradient boosted trees, and they compound the interpretability problem rather than relieving it. Manzil's survey [19] of the field reaches a similar conclusion, identifying dataset imbalance, inconsistent evaluation protocols and the general absence of explainable models as the most persistent methodological weaknesses.

D. Explainability and Adversarial Robustness

Interpretability has only recently been treated as a first-class objective. Lundberg and Lee [20] introduced SHAP, unifying several earlier attribution methods under a formulation grounded in cooperative game theory and providing an efficient exact algorithm for tree ensembles, which makes per-sample attribution computationally practical for models of the kind used here. Ghourabi [21] applied attention mechanisms to Android malware classification, exposing influential features implicitly through attention weights, though without the additive guarantees that a Shapley formulation provides. Mahindru and co-authors [22] developed PermDroid, a permission-oriented feature selection framework that improves efficiency by discarding redundant permissions, yet still returns only a label. Tanha and Kafaie [23] survey explainable techniques for this domain and observe a widely assumed tension between interpretability and accuracy; a central finding of the present work is that this tension does not arise when attribution is applied post hoc to an unmodified classifier.

Robustness against deliberate evasion has become pertinent as reported accuracies have risen. Lan and Nait-Abdesselam [24] demonstrate that feature-level perturbations generated with the assistance of language models can flip a detector's verdict without altering the functional behaviour of the sample, a result directly relevant to any system relying on a fixed and publicly documented feature set. Odat and Yaseen [25] approach the problem from the opposite direction, modelling the co-occurrence of permissions and API calls to capture relational structure that single-feature models overlook, on the argument that relational patterns are harder to perturb without breaking functionality.

E. Positioning of the Present Work

Two observations recur across this literature [26]. Static permission-derived features paired with a gradient boosted ensemble deliver accuracy comparable to far heavier architectures at a small fraction of the computational cost. Yet almost no reported system closes the loop between a prediction and an explanation an analyst can act upon, and none connects the detection output to an established threat intelligence taxonomy. The framework described below is constructed specifically to address both gaps.

III. METHODOLOGY

A. System Overview

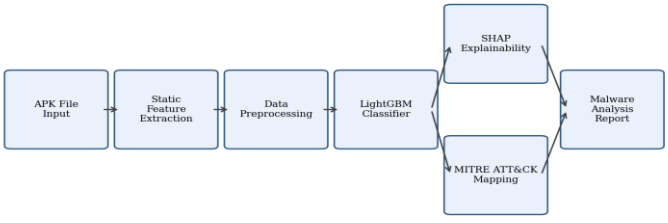

Fig. 1: Architecture of the proposed detection framework

The proposed system is arranged as a sequential pipeline. A submitted package is parsed to extract its manifest,

from which a permission vector is constructed. That vector is cleaned and passed to the trained LightGBM classifier, which produces a probability of maliciousness. The prediction and its associated feature vector are then routed in parallel to an attribution module and a behavioural mapping module, whose outputs are combined with the verdict into a single report. Fig. 1 illustrates the arrangement.

B. Dataset

Training and evaluation use the AndroMD dataset [13], a publicly available tabular corpus of static features extracted from benign and malicious Android applications. Each record corresponds to one application and is described by binary and categorical attributes spanning declared permissions, invoked API calls and registered intents, together with a ground-truth label.

An early phase of this project trained models over the full combined feature set and obtained accuracies close to saturation. Closer examination indicated that a portion of the apparent performance derived not from genuine malicious signal but from incidental regularities in how the benign and malicious portions of the corpus had been assembled, since the two classes were drawn from different sources whose collection processes left distinguishable traces. Models exploiting such artefacts perform impressively on held-out partitions of the same corpus and disappoint on anything else. The deployed classifier was therefore restricted to the permission-derived subset alone. This carries a modest cost in headline accuracy and two compensating benefits: the resulting figures are more credible, and permissions can be recovered from the manifest without bytecode analysis, which keeps extraction fast enough for interactive use.

C. Preprocessing

Records are screened for missing values, duplicate entries and label inconsistencies. Permission features are sparse by nature, with most applications declaring a small subset of the several hundred permissions the platform defines [4], so columns exhibiting negligible variance across the corpus are removed as contributing noise rather than signal. Partitioning into training and test sets uses stratified sampling so that the class ratio is preserved in both, avoiding the distortion a naive random split can introduce when classes are unevenly represented.

D. The LightGBM Classifier

LightGBM [27] was selected for two reasons rather than one. It accommodates sparse high-cardinality feature spaces efficiently, and it trains appreciably faster than comparably accurate alternatives, which matters for a system expected to be retrained as new families appear. Unlike level-wise boosting implementations such as XGBoost [28], LightGBM grows each tree leaf-wise, expanding whichever leaf offers the greatest reduction in loss rather than expanding an entire depth level uniformly. Combined with histogram-based binning of feature values, this reduces both memory footprint and training time.

The mechanism also suits the structure of the problem. Individual permissions are weak indicators considered alone: network access is requested by nearly every application, and the ability to send messages is legitimate in a messaging client. Discriminative power resides in combinations, for instance the joint presence of message-sending capability and the ability to start automatically after a device restart, a pairing far more suggestive than either element separately. Boosted trees capture such conjunctions natively through successive splits, without those interactions being specified in advance.

E. Attribution with SHAP

An accurate model remains opaque unless its outputs can be decomposed [20]. SHAP assigns to each feature a Shapley value computed as the average marginal contribution of that feature across all possible orderings of the remaining features, expressed as

$$\varphi_i = \Sigma_{S \subseteq F \setminus \{i\}} \left[|S|! \, (|F| - |S| - 1)! \, / \, |F|! \right] \times \left[f(S \cup \{i\}) - f(S) \right] \quad (1)$$

where F denotes the complete feature set, S ranges over subsets excluding feature i, and f is the trained model's output function. The formulation guarantees that attributions sum to the difference between the prediction and the expected output over the training distribution, so no portion of a decision is left unaccounted for. Attribution is computed per sample at inference rather than aggregated over the corpus, which is the distinction that matters operationally: an analyst reviewing a flagged package sees the permissions that drove that particular verdict, not an average ranking that may not apply to the case in front of them.

F. Behavioural Mapping to MITRE ATT&CK

A verdict alone conveys little about what an application does. The mapping stage relates the attributed permissions and observed API patterns of a flagged sample to tactics and techniques in the MITRE ATT&CK matrix for mobile platforms [29], among them Execution, Persistence, Credential Access, Discovery and Exfiltration. A package whose attribution is dominated by message-handling and device-identity permissions maps toward credential access and collection; one combining network access with automatic restart capability maps additionally toward persistence and exfiltration. The result is a short behavioural profile expressed in terminology already embedded in incident response practice, which is more directly actionable than a probability.

G. Handling Packed and Obfuscated Packages

Testing against a banking trojan sample obtained during development exposed a limitation that benchmark corpora rarely surface: the standard binary XML parser fails outright on manifests deliberately malformed to resist inspection. A

layered recovery routine was therefore added, falling back through progressively more tolerant parsing strategies when the primary parser cannot resolve a manifest. A complementary packer detection module inspects structural indicators, including class name distributions dominated by one and two character identifiers, absent or encrypted resource tables and signatures characteristic of known commercial packers. Where such indicators are present the reported risk level is escalated, since heavy protection is itself meaningful irrespective of what the declared permissions suggest. This escalation supplements the classifier output rather than overriding it, and is surfaced to the analyst as a separate signal.

H. Web Application

The pipeline is exposed through a Flask application, APKGuard, which accepts packages up to 200 MB, validates archive structure, and executes extraction, classification, attribution and mapping in sequence. The response comprises a scored verdict, a breakdown separating dangerous from ordinary permissions, an attribution chart and, for malicious samples, a table of mapped tactics and techniques.

IV. EXPERIMENTAL SETUP

A. Environment

The system was implemented in Python 3.8 using LightGBM, scikit-learn, the SHAP library and Androguard for package parsing, with Flask providing the web interface. Experiments were conducted on a workstation with an Intel Core i5 processor and 16 GB of memory running Windows 11. No graphics accelerator was employed at any stage, which is itself a relevant finding: the entire pipeline is trainable and deployable on ordinary office hardware, unlike the deep architectures surveyed in Section II.

B. Protocol

The permission subset of AndroMD was divided into training and test partitions by stratified sampling. Hyperparameters were selected by search over the number of leaves per tree, maximum depth, learning rate, boosting rounds and the minimum sample count required before a leaf may split further, with early stopping monitored on a validation fold. A moderate leaf count paired with a conservative learning rate generalised best; permitting unrestricted depth improved training accuracy without improving held-out accuracy, which is the expected signature of overfitting on repetitive permission patterns. Per-round feature subsampling was enabled and shortened training with no measurable accuracy cost.

Three baselines were trained under identical preprocessing, identical partitions and comparable tuning budgets: XGBoost, Random Forest [30] and CatBoost [31]. Holding these conditions constant is essential, because differences in preprocessing or search effort readily account for margins larger than those separating the models themselves.

C. Evaluation Metrics

Performance is reported using accuracy, precision, recall, F1-score, ROC-AUC and the Matthews correlation coefficient, together with wall-clock training time. The Matthews coefficient is included deliberately, since it incorporates all four cells of the confusion matrix and does not inflate under class imbalance in the way accuracy can. Training time is treated as a substantive result rather than an incidental observation, given that periodic retraining is an operational necessity in this domain.

V. RESULTS AND DISCUSSION

A. Classification Performance

Fig. 2 presents the confusion matrix produced on the held-out partition. Of 91,121 malicious samples the model identified 89,234 correctly, leaving 1,887 false negatives; of 86,299 benign samples it cleared 84,765, producing 1,534 false positives.

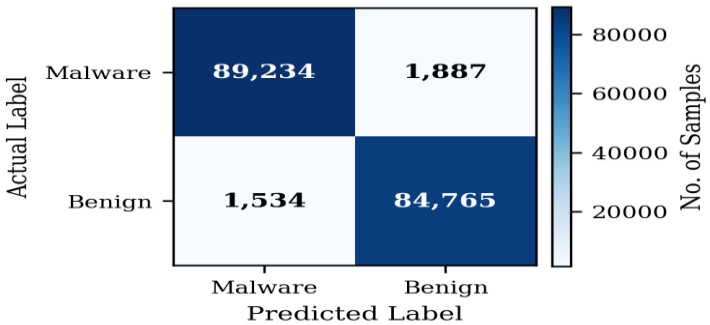

Fig. 2: Confusion matrix on the held-out test partition

Table 1 and Fig. 3 report the resulting metrics. Accuracy reached 99.13%, with precision at 99.15%, recall at 99.10% and an F1-score of 99.12%. The Matthews coefficient of 98.26% is the more informative figure, indicating that performance is balanced across both classes rather than resting on a majority-class bias.

Metric	Value
Accuracy	99.13%
Precision	99.15%
Recall	99.10%
F1-Score	99.12%
ROC-AUC	99.84%
Matthews Correlation Coefficient	98.26%
Training Time	27.84 s

Table 1: Performance of the proposed LightGBM model

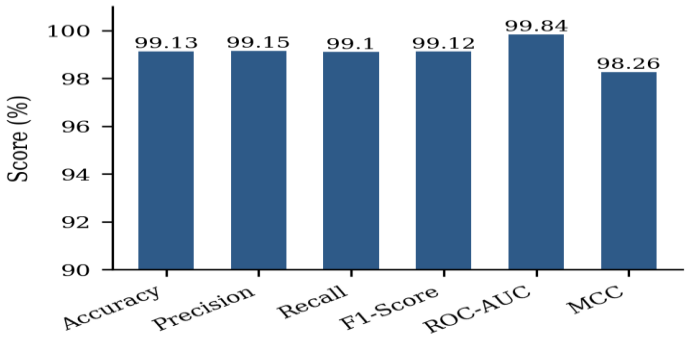

Fig. 3: Metric summary for the proposed model

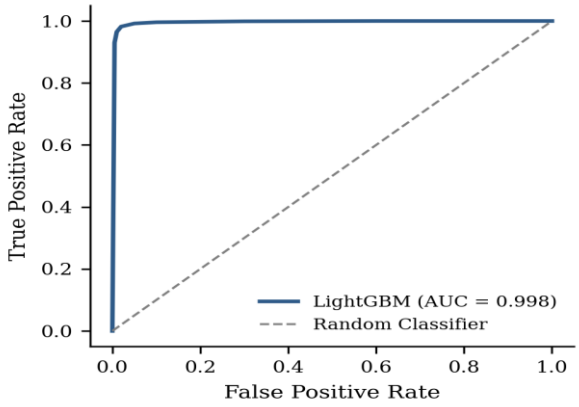

Fig. 4: Receiver operating characteristic curve

The receiver operating characteristic curve in Fig. 4 rises steeply and reaches an area of 0.998, indicating that the separation between classes is maintained across the full range of decision thresholds rather than holding only at the default cut-off of 0.5. This matters in deployment, where thresholds are frequently adjusted to favour recall in high-assurance settings or precision where analyst time is the binding constraint.

B. Comparison with Baseline Models

Table 2 and Fig. 5 compare the four classifiers. LightGBM leads on every metric, although the margin over XGBoost is slight, under half a percentage point on accuracy. It would be misleading to present that gap as decisive; both gradient boosting implementations appear to be approaching the ceiling this feature representation permits, and the remaining difference is comparable to what reasonable variation in tuning could produce.

Model	Accuracy	Precision	Recall	F1	ROC-AUC
LightGBM (proposed)	99.13%	99.15%	99.10%	99.12%	99.84%
XGBoost	98.71%	98.74%	98.66%	98.70%	99.38%
Random Forest	97.94%	98.01%	97.86%	97.93%	98.87%
CatBoost	97.36%	97.43%	97.28%	97.35%	98.52%

Table 2: Comparative evaluation of classifiers

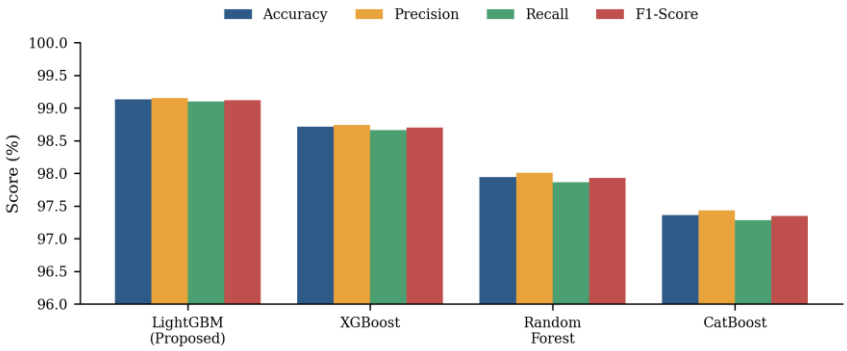

Fig. 5: Comparative performance across evaluated models

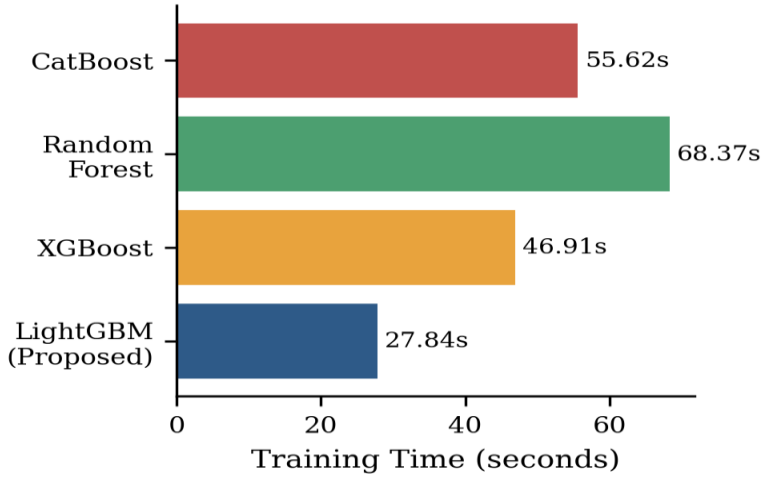

Fig. 6: Training time comparison

The clearer separation appears in training cost, shown in Fig. 6. LightGBM completed training in 27.84 seconds against 46.91 for XGBoost, 55.62 for CatBoost and 68.37 for Random Forest, reductions of between forty and fifty-nine per cent. This follows directly from histogram binning and leaf-wise growth, and it is arguably the more consequential result: where accuracy differences are marginal, a model that retrains in half the time permits more frequent refresh cycles against an adversary population that does not stand still.

C. Feature Attribution

Fig. 7 ranks permissions by mean absolute attribution. Network access, telephony state and message sending dominate the malicious side of the distribution. This accords with documented tradecraft: network access is required to reach command infrastructure, telephony state exposes device identifiers useful for targeting and fingerprinting, and message access permits interception of one-time authentication codes, which is the operative capability in banking fraud. Automatic restart capability and fine-grained location follow, consistent with persistence and targeting respectively.

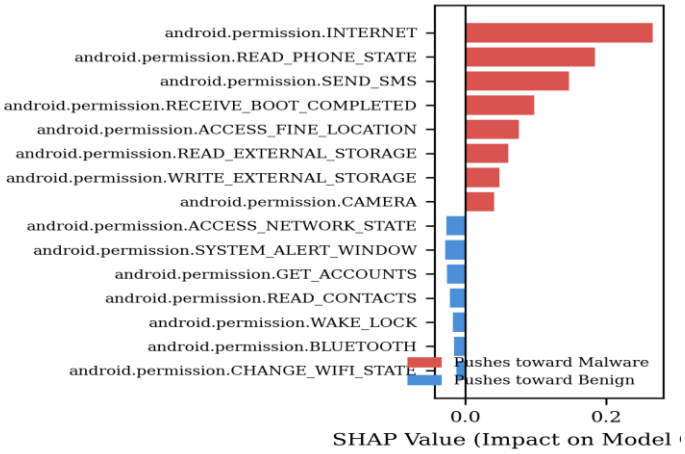

Fig. 7: Permission attribution derived from SHAP values

The benign side of the distribution deserves equal attention. Network state inspection, contact access and wake lock acquisition push predictions toward benign classification. These are ordinary capabilities in messaging, navigation and media applications, and their negative attribution indicates that the model has not simply learned to penalise permissions that sound sensitive. Had every privacy-relevant permission carried positive weight, that would suggest the classifier had latched onto a superficial heuristic. The observed asymmetry instead suggests something closer to the conditional reasoning an experienced analyst applies, which is a meaningful indication that the learned decision function is grounded in the phenomenon rather than in an artefact of the corpus.

D. Behavioural Mapping Results

For samples classified as malicious, the mapping stage consistently produced small and internally coherent sets of tactics rather than scattered lists, which is itself a useful signal of consistency. Table 3 shows a representative mapping generated during testing for a sample flagged with high confidence.

Tactic ID	Tactic	Technique
TA0002	Execution	T1204.002 User Execution
TA0006	Credential Access	T1555.003 Credentials from Stores
TA0007	Discovery	T1420 File and Directory Discovery
TA0010	Exfiltration	T1041 Exfiltration Over C2 Channel

Table 3: Representative ATT&CK mapping for a detected sample

E. Error Analysis

Aggregate metrics conceal structure in the residual errors, and that structure is informative. Inspection of misclassified records indicates that false negatives concentrate among malicious applications with unusually sparse permission footprints, samples requesting only one or two of the strongly predictive capabilities identified above. This is unlikely to be coincidental. An author aware that permission-based screening is common has an obvious incentive to request the minimum set required for the payload to function, and such samples occupy a region of the feature space populated largely by legitimate lightweight utilities.

False positives exhibit the mirror pattern, clustering among benign applications that request several high-signal permissions for entirely defensible reasons. A navigation application combining network access, precise location and telephony state is functionally indistinguishable from a surveillance tool at the level of the manifest alone. Both error classes therefore concentrate where permissions are genuinely ambiguous, which suggests the remaining margin reflects a limit of the representation rather than insufficient tuning. Closing it further would require a complementary behavioural signal, not additional optimisation of the same features.

F. Robustness on Protected Samples

The recovery and packer detection components described in Section III-G were assessed qualitatively against a small set of manually collected packed and repackaged samples, including the banking trojan that prompted their development. In each case the layered parser recovered sufficient manifest structure to construct a usable permission vector where the default parser failed completely, and the packer detector correctly identified protection artefacts invisible to the classifier itself. This evaluation is smaller in scale than the benchmark comparison and is reported as a qualitative observation rather than a quantitative claim; its value lies in demonstrating that the pipeline degrades observably rather than silently when it encounters adversarial packaging.

G. Discussion

Two propositions are supported by these results. The first is that a carefully scoped permission-only representation combined with a well-tuned gradient boosted ensemble matches or exceeds accuracies reported for substantially heavier architectures, at a computational cost small enough for deployment on commodity hardware. The second, and the more consequential for practice, is that interpretability need not be purchased with accuracy. Because attribution and behavioural mapping are applied after training to an unmodified model, the classifier's predictive behaviour is unchanged by their presence. The trade-off assumed in much of the explainable AI literature arises when interpretability constraints are imposed on the model itself, and it does not apply to post-hoc attribution over tree ensembles.

The decision to restrict the deployed model to permission features warrants emphasis because it runs against the usual incentive. Combining feature families produced better numbers, and reporting those numbers would have been straightforward. They were set aside because their source could not be established as genuine malicious signal rather than collection artefact. Reported accuracies in this field should be interpreted in light of how the underlying corpus was constructed, and a figure near saturation is more often an indication that something in the data requires examination than evidence of an exceptional model.

H. Limitations

The dependence on static manifest features is simultaneously the principal strength and the principal limitation of the framework. Behaviour that appears only at runtime is invisible to it. A loader application that requests no suspicious

permissions and retrieves its payload after installation would be classified as benign on the evidence available, and the packer heuristics introduced to compensate address only the subset of such cases that leave structural traces. A sufficiently determined adversary aware of the feature set could construct a sample evading both the classifier and the structural checks, as the adversarial results discussed in Section II-D suggest.

The evaluation is also bounded by the composition of AndroMD. Performance on families or platform versions absent from that corpus has not been independently established, and would require verification before the system could be relied upon in a production security operation without human review. Finally, the qualitative robustness assessment rests on a small sample and should be regarded as indicative rather than conclusive.

VI. CONCLUSION AND FUTURE SCOPE

This paper has presented an explainable framework for Android malware detection combining a LightGBM classifier over manifest-derived permission features, per-sample attribution through SHapley Additive exPlanations, and behavioural mapping onto the MITRE ATT&CK taxonomy. Evaluated on the AndroMD dataset, the model achieved 99.13% accuracy, a ROC-AUC of 99.84% and a Matthews correlation coefficient of 98.26%, outperforming XGBoost, Random Forest and CatBoost while training between forty and fifty-nine per cent faster than any of them.

The contribution extends beyond the metrics. Every prediction carries an account of the permissions responsible for it, and malicious verdicts are additionally expressed as adversary tactics and techniques in vocabulary that incident response teams already employ. The accompanying Flask application demonstrates that extraction, classification, attribution and mapping execute end to end on an uploaded package within seconds on ordinary hardware, which places the framework within reach of organisations lacking dedicated malware analysis capacity.

The system is nonetheless best understood as a first-line static screening layer rather than a replacement for deeper analysis. Its residual errors concentrate precisely where static evidence is inherently ambiguous, and the packer detection layer mitigates rather than resolves the obfuscation problem. Recognising that boundary is part of deploying the system responsibly.

Several directions follow from these limitations. Incorporating selective dynamic instrumentation, triggered only for samples the static classifier scores near its decision boundary, would address the runtime blind spot while preserving the latency advantage for the majority of cases resolved confidently. Extending the ATT&CK integration from tactic identification toward suggested mitigations and probable attack sequencing would move the tool from reporting toward active response support. Establishing a retraining cadence against newly observed families, together with monitoring for distributional drift in the permission landscape as platform releases alter the permission model, would help sustain performance over time. Finally, a structured evaluation with practising analysts would establish whether the attributions and mappings are presented in a form that measurably accelerates triage, a question that accuracy figures cannot answer.

References

1. J. Senanayake, H. Kalutarage and M. O. Al-Kadri, “Android mobile malware detection using machine learning: A systematic review,” Electronics, vol. 10, no. 13, art. 1606, 2021. doi: 10.3390/electronics10131606
2. A. Qamar, A. Karim and V. Chang, “Mobile malware attacks: Review, taxonomy and future directions,” Future Generation Computer Systems, vol. 97, pp. 887-909, 2019. doi: 10.1016/j.future.2019.03.007
3. Q. Wu, X. Zhu and B. Liu, “A survey of Android malware static detection technology based on machine learning,” Mobile Information Systems, vol. 2021, art. 8896013, 2021. doi: 10.1155/2021/8896013
4. J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an and H. Ye, “Significant permission identification for machine-learning-based Android malware detection,” IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216-3225, 2018. doi: 10.1109/TII.2017.2789219
5. N. Peiravian and X. Zhu, “Machine learning for Android malware detection using permission and API calls,” in Proc. IEEE 25th Int. Conf. on Tools with Artificial Intelligence (ICTAI), 2013, pp. 300-305. doi: 10.1109/ICTAI.2013.53
6. W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu and X. Zhang, “DroidEnsemble: Detecting Android malicious applications with ensemble of string and structural static features,” IEEE Access, vol. 6, pp. 31798-31807, 2018. doi: 10.1109/ACCESS.2018.2835654
7. A. T. Kabakus, “What static analysis can utmost offer for Android malware detection,” Information Technology and Control, vol. 48, no. 2, pp. 235-249, 2019. doi: 10.5755/j01.itc.48.2.21457
8. 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), San Diego, CA, USA, 2014. doi: 10.14722/ndss.2014.23247
9. S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song and H. Yu, “SAMADroid: A novel 3-level hybrid malware detection model for Android operating system,” IEEE Access, vol. 6, pp. 4321-4339, 2018. doi: 10.1109/ACCESS.2018.2792941
10. [10] O. E. Saied and K. H. Thanoon, “Static analysis-based detection of Android malware using machine learning algorithms,” Sistemasi: Jurnal Sistem Informasi, vol. 14, no. 5, pp. 2540-2551, 2025. doi: 10.32520/stmsi.v14i5.7536
11. P. A. Museeb et al., “Android malware detection using API calls and permissions with Random Forest classifier,” IEEE Access, vol. 14, pp. 6464-6470, 2026. doi: 10.1109/ACCESS.2026.3651861
12. A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil and S. Furnell, “AndroDialysis: Analysis of Android intent effectiveness in malware detection,” Computers & Security, vol. 65, pp. 121-134, 2017. doi: 10.1016/j.cose.2016.11.007
13. K. A. Prasad et al., “AndroMD: An Android malware detection framework based on source code analysis and permission scanning,” Results in Engineering, 2025. doi: 10.1016/j.rineng.2025.107050
14. S. Zhou, H. Li, X. Fu, D. Han and X. He, “Novel multi-classification dynamic detection model for Android malware based on improved Zebra Optimization Algorithm and LightGBM,” Sensors, vol. 24, no. 18, art. 5975, 2024. doi: 10.3390/s24185975
15. S. Y. Yerima and S. Sezer, “DroidFusion: A novel multilevel classifier fusion approach for Android malware detection,” IEEE Transactions on Cybernetics, vol. 49, no. 2, pp. 453-466, 2019. doi: 10.1109/TCYB.2017.2777960

16. N. McLaughlin et al., "Deep Android malware detection," in Proc. 7th ACM Conf. on Data and Application Security and Privacy (CODASPY), 2017, pp. 301-308. doi: 10.1145/3029806.3029823
17. R. Ma et al., "A lightweight deep learning-based Android malware detection system," Expert Systems with Applications, 2024. doi: 10.1016/j.eswa.2024.124633
18. T. Sun et al., "DetectBERT: Full app-level representation learning to detect Android malware," arXiv preprint arXiv:2408.16353, 2024. doi: 10.48550/arXiv.2408.16353
19. H. H. R. Manzil and S. M. Naik, "Android malware category detection using a novel feature vector-based machine learning model," Cybersecurity, vol. 6, art. 6, 2023. doi: 10.1186/s42400-023-00139-y
20. S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017, pp. 4765-4774. doi: 10.48550/arXiv.1705.07874
21. M. Ghourabi, "An attention-based approach to enhance the detection and classification of Android malware," Computers, Materials & Continua, vol. 80, no. 2, 2024. doi: 10.32604/cmc.2024.053163
22. A. Mahindru, H. Arora, A. Kumar et al., "PermDroid: A framework developed using proposed feature selection approach and machine learning techniques for Android malware detection," Scientific Reports, vol. 14, art. 10724, 2024. doi: 10.1038/s41598-024-60982-y
23. X. Tanha and M. Kafaie, "Explainable AI techniques for Android malware detection," ACM Computing Surveys, vol. 57, no. 2, 2025. doi: 10.1145/3631234
24. T. Lan and F. Nait-Abdesselam, "LLM-driven feature-level adversarial attacks on Android malware detectors," arXiv preprint, 2025. doi: 10.48550/arXiv.2512.21404
25. Y. Odat and Q. Yaseen, "A novel machine learning approach for Android malware detection based on the co-existence of features," IEEE Access, vol. 11, pp. 15471-15484, 2023. doi: 10.1109/ACCESS.2023.3244656
26. N. Zhang, J. Xue, Y. Ma, R. Zhang, T. Liang and Y. Tan, "Hybrid sequence-based Android malware detection using natural language processing," International Journal of Intelligent Systems, vol. 36, no. 10, pp. 5770-5784, 2021. doi: 10.1002/int.22529
27. G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017, pp. 3146-3154.
28. T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2016, pp. 785-794. doi: 10.1145/2939672.2939785
29. B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington and C. B. Thomas, "MITRE ATT&CK: Design and philosophy," MITRE Corporation, Technical Report MP180360R1, 2020.
30. L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5-32, 2001. doi: 10.1023/A:1010933404324
31. L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush and A. Gulin, "CatBoost: Unbiased boosting with categorical features," in Advances in Neural Information Processing Systems (NeurIPS), vol. 31, 2018, pp. 6638-6648. doi: 10.48550/arXiv.1706.09516