EasyChair Preprint

№ 15990

A Comprehensive Review on Machine Learning and Deep Learning Based Malware Detection Methods

Mahesh Ganesamoorthi, Kannimuthu Subramanian and Bhanu D

August 7, 2025

A Comprehensive Review on Machine Learning and Deep Learning Based Malware Detection Methods

1st Mahesh Ganesamoorthi
Expedia Group
Seattle, United States of America
maheshganesamoorthi@gmail.com

2nd Kannimuthu Subramanian
Karpagam College of Engineering
Coimbatore, India
kannimuthu@kce.ac.in

3rd Bhanu D
Karpagam Institute of Technology
Coimbatore, India
bhanu.saran@gmail.com

2024 International Conference on Emerging Research in Computational Science (ICERCS) | 979-8-3315-3496-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICERCS63125.2024.10894949

***Abstract*—Malware detection has become a significant aspect of cybersecurity, specifically with the widespread use of Android devices. Conventional malware detection methods, such as static and signature-based approaches have been overtaken by evolving and increasingly sophisticated malware techniques. Machine learning and deep learning have emerged as powerful tools in this field, offering enhanced detection capabilities through advanced pattern recognition. This paper presents a comprehensive review of various machine learning and deep learning methods, including convolutional neural networks (CNN), Bayesian classification, ensemble learning, and hybrid models for malware detection. The study evaluates these techniques in terms of accuracy, efficiency, adaptability, and their capacity to handle real-time detection, dataset diversity, and obfuscated malware. Additionally, it explores challenges such as class imbalance and the need for more interpretable models. The findings suggest that while CNN-based methods offer the highest accuracy, ensemble models strike a balance between precision and computational efficiency**.

***Keywords*—Android malware detection, Deep learning, Hybrid models, Machine learning, Mobile security, Real-time detection.**

I. Introduction

The rapid proliferation of mobile devices has made them a primary target for malware attacks. As of recent reports, Android malware represents a significant portion of global cyber threats, with thousands of new malware variants emerging daily. Traditional malware detection techniques, which rely heavily on static analysis and signature-based methods, are increasingly ineffective against modern, sophisticated malware. These conventional methods struggle to detect zero-day exploits, obfuscated code, and dynamic behaviors that have become stamps of contemporary malware.

One of the major shortcomings of static and signature-based approaches is their inability to adapt to evolving malware threats. These methods focus on predefined signatures or patterns, rendering them ineffective against newly developed or heavily obfuscated malware variants. Additionally, such techniques are computationally efficient but lack robustness when dealing with advanced evasion tactics. On the other hand, dynamic analysis, which involves monitoring application behavior at runtime, provides more accurate results but often at the cost of high computational and time requirements. This makes it impractical for large-scale and real-time malware detection. Moreover, the increasing complexity of malware requires more sophisticated analysis techniques capable of detecting patterns that go beyond superficial features like permissions and API calls.

This article provides a comprehensive review of machine learning and deep learning techniques for Android malware detection, focusing on CNN-based methods[1-2], Bayesian classifiers[3], ensemble learning approaches[4-19], and hybrid models [20-29] that integrate static and dynamic analysis techniques. The paper evaluates strengths and limitations each method, providing a detailed analysis of their performance in terms of accuracy, computational efficiency, adaptability, and real-time applicability. Additionally, the study discusses recent advancements for malware detection with limited datasets, and graph-based approaches that analyze inter-app communication to identify malicious activities. The paper provides an in-depth analysis of machine learning and deep learning methods, comparing their effectiveness in detecting malware across various datasets, including Drebin, OmniDroid, and Contagio. The study highlights key challenges in the current approaches, such as their inability to handle real-time detection due to computational complexity, limited dataset diversity, and their vulnerability to obfuscated malware. The paper examines the trade-offs between accuracy, computational efficiency, and adaptability, suggesting ensemble and hybrid models as potential solutions for achieving high-performance detection in resource-constrained environments.

II. Literature Review

This section presents a detailed survey on the machine learning and deep learning-based malware detection methods with the conclusion made.

A. Efficient Android Malware Identification Using CNN

Ksibi et al [1] implemented a novel approach of converting APK files into grayscale images and feeding them into deep learning models, specifically CNNs such as VGG16, DenseNet169, and InceptionV3. By leveraging transfer learning, pre-trained models are used to enhance detection efficiency with limited training data. VGG16 achieves a superior accuracy of 95.83%, while DenseNet169 and InceptionV3 both maintain accuracy around 95.24%. The models, however, require high computational resources for image transformation and model training. This work uses the dataset which includes 10,000 Android apps (both malware and benign) [2] but lacks diversity in terms of obfuscated malware or complex attacks. Future research should consider more recent malware variants to improve model robustness. Transfer learning significantly improves performance with fewer training examples, but the model's computational complexity hinders its real-time applicability, especially on mobile devices. CNN-based models remain the most accurate but are resource-heavy, limiting their scalability to mobile and real-time environments.

B. Bayesian Classification for Android Malware

Guerra-Manzanares et al employed [3] a Bayesian classification method based on static features like permissions and API calls. Using probabilistic models, it calculates the likelihood that a combination of these features indicates malicious behavior. The Bayesian approach achieves an accuracy of 91.1%, making it effective for basic malware detection without needing heavy computational resources. The custom dataset focuses on earlier Android versions, limiting its applicability for newer versions with more complex malware. The study highlights the need for more diverse datasets to capture modern malware techniques. The model is lightweight, making it ideal for mobile devices or preliminary malware scans. However, its reliance on static analysis and predefined rules makes it susceptible to evolving malware like zero-day attacks or obfuscation techniques. The Bayesian method is a lightweight alternative that can complement more complex models by serving as a first line of defense before deeper analysis is conducted

C. MEFDroid Framework

MEFDroid [4] is a multi-model ensemble learning framework combining unsupervised and supervised learning via Sparse Autoencoders and Stacked De-noising Autoencoders (SDAE). These autoencoders extract features automatically and are combined with multiple base classifiers such as decision trees, Random Forests, and SVMs. MEFDroid achieves 95.14% accuracy with an F1-score of 97.12%, showcasing its ability to handle imbalanced datasets effectively. This is especially important in real-world scenarios where benign apps significantly outnumber malware. The framework was evaluated on the Drebin and AndroMD [5] datasets. While comprehensive, the datasets could benefit from more recent malware samples to ensure the model's adaptability to evolving threats. MEFDroid excels in precision and recall, especially for imbalanced datasets. However, the framework's complexity and computational demands make it less suitable for real-time malware detection, and updating it with new malware signatures can be slow. MEFDroid is highly effective for handling imbalanced datasets and works well in situations where malware detection must remain robust and diverse, but its computational demands limit its real-time applicability.

D. MalDetect

MalDetect [5] employed an ensemble approach that combines multiple machine learning algorithms, including Naive Bayes, J48 (C4.5 decision tree), and AdaBoost. By integrating these diverse algorithms, MalDetect enhances its detection capabilities, allowing for better generalization across various types of malwares. Each classifier contributes unique strengths, enabling the model to perform well across different data distributions and types of malicious behavior. The ensemble nature of the approach allows MalDetect to achieve solid performance metrics, capturing a wide variety of malware types effectively. This work uses multiple datasets, including Drebin, AMD and Genome. This work possesses enhanced flexibility and robustness through ensemble methods, allowing it to perform well in diverse environments. The integration of multiple classifiers mitigates the risk of relying on a single algorithm, reducing the chance of misclassification. This work consumes high resource during training phases may impact performance in real-time applications, especially on lower-end devices. This work takes potentially longer training times due to the complexity of combining multiple models. MalDetect is ideal for enterprise environments with sufficient computational resources. Its ensemble approach yields high accuracy, although its performance may be suboptimal in real-time applications due to high resource requirements.

E. Random Forest Classification for API Calls

Several researches utilized Random Forest [6-18] classification based on API calls and permission analysis, leveraging decision trees to classify malware. It's a simpler model that can handle medium-sized datasets efficiently, and it's less sensitive to noise in data. With an accuracy of 94.36% [19] and an F1-score of 88.75% [19], Random Forest performs well for general malware detection but struggles with highly obfuscated malware. The study uses the Drebin dataset, one of the most widely used Android malware datasets. However, the dataset lacks modern malware samples that would present a greater challenge to the classifier. Random Forest is efficient, making it suitable for quick classification. Its reliance on static features limits its efficacy against modern evasion techniques like code obfuscation or zero-day malware. Random Forest models remain effective for traditional static feature-based malware detection but require enhancement through dynamic analysis to cope with modern malware threats.

F. Transfer Learning Approach with Pre-trained CNN Models

Alejandro et al. [20] combined transfer learning with pre-trained CNN models (e.g., Inception), allowing for fast training with fewer samples. By fine-tuning pre-trained models, the method reduces the need for massive labeled datasets. The model achieved 93.7% accuracy, demonstrating the efficacy of transfer learning in malware detection when limited data is available. The OmniDroid [17, 20] dataset used in this study is diverse, covering multiple types of malware. However, the model's success relies heavily on the availability of relevant pre-trained models, which may not always be accessible for all malware types. Transfer learning speeds up training and boosts accuracy with smaller datasets. However, the model's dependency on pre-trained architectures may limit its ability to handle highly specialized or novel malware.

G. ProDroid

ProDroid [21] employed a hybrid analysis approach that integrates both static and dynamic analysis techniques. The static analysis phase extracts feature from application code, while the dynamic analysis phase observes the runtime behavior of applications. This dual approach helps in identifying malicious activities that might not be apparent in static analysis alone. ProDroid uses a variety of feature extraction methods, including opcode analysis, permission usage, and API call patterns, which are fed into a machine learning classifier to make the final determination of whether an app is benign or malicious. ProDroid utilizes a diverse range of datasets, including Drebin, Contagio and other academic datasets that encompass various Android app characteristics, ensuring comprehensive coverage of potential threats [22]. This approach is high adaptable with accuracy due to the combination of static and dynamic analysis. It has the ability to detect complex malware

behaviors that are missed by purely static analysis tools. Computational intensity may lead to longer processing times, making it less suitable for devices with limited processing capabilities. This work has the potential for false positives in certain edge cases, particularly with legitimate apps that exhibit similar behaviors to malware. ProDroid is an excellent choice for environments that prioritize detection accuracy and adaptability. However, its deployment may be limited on lower-end devices due to high resource requirements, which could hinder its broader applicability.

H. WHGDroid

WHGDroid [23] employed a graph-based approach that models interactions among applications using weighted heterogeneous graphs. The interactions, such as method calls and data flows, are analyzed using Graph Neural Networks (GNNs) to identify potentially malicious behaviors. This methodology allows for a nuanced understanding of how applications communicate, facilitating the detection of threats that arise from complex interactions. WHGDroid's performance indicates that it is adept at recognizing malicious interactions, making it particularly effective in scenarios involving multiple applications. This work [24, 25] uses utilizes datasets like Anzhi and AndroZoo. It is Effective in detecting complex communication patterns that might not be evident in traditional analysis methods. It is Capable of identifying hidden threats through interaction analysis among apps, which is a unique advantage. The complexity of graph processing can lead to increased latency, impacting real-time effectiveness. This requires significant computational resources, particularly during the graph construction phase. WHGDroid is a promising solution for environments focused on detecting inter-app communication threats but may require optimization for improved response times in real-world applications.

I. XManDroid

XManDroid [26] focused on monitoring inter-app communications, employing classifiers like Support Vector Machine (SVM) and Random Forest to detect malicious behavior based on inter-component communication (ICC) patterns. The model analyzes the data exchanged between apps, such as intents and broadcasts, to identify anomalies indicative of malware. XManDroid demonstrates effective performance metrics, particularly in identifying malicious interactions among applications. XManDroid utilizes a custom dataset specifically designed to emphasize inter-app communication anomalies, enhancing its specificity in detection. This dataset is curated to include both benign and malicious applications, focusing on the communications that occur between them. This work is excellent for real-time monitoring due to its focus on ICC, making it suitable for dynamic environments. It is capable of detecting malicious interactions that traditional malware detection methods may overlook. This work relies heavily on continuous updates to remain effective against rapidly evolving threats. This work is well-suited for dynamic environments, XManDroid excels in monitoring inter-app communications but requires regular updates to maintain its effectiveness against new malware variants.

J. MADRF-CNN

MADRF-CNN [27] integrated Convolutional Neural Networks (CNN) with Random Forest classifiers to improve malware detection accuracy through deep learning techniques. The CNN component extracts spatial features from application representations, while the Random Forest classifier provides a robust classification mechanism that enhances overall detection performance. This model shows exceptional accuracy, particularly in recognizing complex malware patterns that require nuanced detection strategies. MADRF-CNN employs a combination of datasets, including: Drebin, AMD and Contagio. This work has High accuracy due to the deep learning framework, allowing for improved detection of complex malware. It has good adaptability with more data, making it scalable as new threats emerge. This work can be resource-intensive, requiring significant computational power and time. Deep learning models can be opaque, making it challenging to interpret their decision-making processes. MADRF-CNN is a robust model for malware detection, demonstrating high accuracy and adaptability. Its advanced methodologies position it well for modern malware threats, although it may require substantial resources during the training phase.

J. KronoDroid

KronoDroid [28] focused on continuous learning by employing reinforcement learning techniques to adapt to new malware patterns over time. The model updates its parameters dynamically based on feedback from its environment, enabling it to learn from ongoing interactions and emerging threats. KronoDroid showcases reliable performance metrics, emphasizing its capability to learn from real-world data continuously. KronoDroid uses Drebin, Genome and Contagio datasets for experimentation. These datasets emphasize continuous learning, focusing on dynamic adaptation from diverse sources. This work is highly adaptable to evolving malware, making it suitable for dynamic environments. It is capable of learning from new data, improving detection over time. This approach is vulnerability to low-quality data, which may impair detection accuracy if not managed properly. The reliance on continuous learning processes may lead to stability issues if the incoming data is inconsistent. KronoDroid is highly effective in dynamic environments, making it a strong candidate for modern malware detection systems. However, it requires high-quality data for optimal performance and must be carefully managed to mitigate risks associated with data quality.

III. COMPARATIVE ANALYSIS

This section showcases an in-depth analysis of various malware detection studies, each using different methodologies, techniques, and datasets to achieve specific performance outcomes. Table 1 and Table 2 expresses the expanded methodology comparison table with performance details. The Efficient CNN approach applies advanced deep learning techniques such as VGG16 and DenseNet169, on APK images. With a high accuracy of 95.83% and notable precision and recall scores, this method is proficient at detecting complex patterns within malware datasets. Bayesian Method, on the other hand, utilizes a lightweight Bayesian Machine Learning model that operates on a custom dataset through static analysis of permissions and API calls, achieving an accuracy of 91.1%. This method, ideal for mobile devices due to its low computational cost, has a relatively higher False Positive Rate, reflecting a trade-off between simplicity and detection precision.

MEFDroid uses an ensemble learning approach by combining sparse autoencoders and denoising autoencoders (SDAE), yielding a robust performance with an accuracy of 95.14% on Drebin and AndroMD datasets. This model excels with imbalanced datasets and achieves a high F1-Score of 97.12%, indicating an effective handling of varied malware samples. Similarly, Random Forest relies on API call and permission-based static analysis to attain a 94.36% accuracy on the Drebin dataset, although it is somewhat limited by obfuscation techniques due to its static analysis constraint, resulting in a lower F1-Score of 88.75%. Transfer Learning employs pre-trained CNN models like Inception to reduce training time while achieving 93.7% accuracy on OmniDroid. This model adapts quickly to similar datasets but depends heavily on the pre-trained model's applicability to new types of malwares. The Hybrid Model, which combines static and dynamic features from both API calls and system logs, stands out with a high accuracy of 96.5% on CICInvesAndMal2019 and a correspondingly high F1-Score. This approach proves valuable for real-time applications, merging the strengths of static and dynamic detection for robust malware analysis.

The ProDroid model also follows a hybrid approach, that combines static and dynamic analysis through Multiple Sequence Alignment (MSA) and Profile Hidden Markov Models (PHMMs). This is tested on Drebin and Contagio datasets, achieves 95.7% accuracy and a balanced set of metrics, though it faces computational challenges in low-resource settings. MalDetect, an ensemble model combining Naive Bayes, J48, and AdaBoost, delivers 94.6% accuracy across Drebin, AMD, and Genome datasets. With an area under the curve (AUC) of 0.96, it balances precision and inference speed but requires significant resources for training. WHGDroid and XManDroid address inter-app communication threats through graph-based detection and ensemble inter-app communication monitoring, respectively. These models perform well, with AUC values close to 0.95, though both face challenges in real-time responsiveness and monitoring overhead. Finally, MADRF-CNN and KronoDroid uses deep learning and incremental learning techniques. MADRF-CNN integrates CNN with Random Forest to process dex files, achieving a high accuracy of 96.1% and an AUC of 0.97, which enhances efficiency in inference time but adds computational overhead during training. KronoDroid uses incremental learning, adapting continuously with high accuracy and performance stability but being somewhat sensitive to noisy data inputs, which may affect real-time accuracy.

Several case studies demonstrate the effectiveness of ML and DL-based approaches for Android malware detection. CNN models like VGG16 achieve high accuracy (95.83%) by processing APK files as grayscale images but require significant computational resources. Lightweight Bayesian classifiers focus on static features and deliver 91.1% accuracy, though they struggle with obfuscated malware. Hybrid models such as ProDroid integrate static and dynamic analysis, achieving 95.7% accuracy, while ensemble methods like MEFDroid and MalDetect excel with imbalanced datasets, attaining F1-scores of 97.12% and 94%, respectively. Advanced approaches like MADRF-CNN combine deep learning with Random Forest for 96.1% accuracy, and KronoDroid employs reinforcement learning for continuous adaptation to new threats. Graph-based techniques, exemplified by WHGDroid, effectively identify inter-app communication threats but are computationally intensive. These studies highlight a trade-off between accuracy, computational efficiency, and adaptability, with hybrid and ensemble models emerging as robust solutions for modern malware detection.

The evaluation of machine learning and deep learning techniques for malware detection is incomplete without discussing key performance metrics such as accuracy, precision, recall, F1-score, and Area Under the Curve (AUC). Accuracy, while commonly reported, can be misleading in imbalanced datasets where malware instances are significantly outnumbered by benign ones. Precision and recall offer deeper insights, with precision focusing on reducing false positives by measuring the proportion of correctly identified malware among all predicted malware instances, and recall emphasizing the detection of actual malware cases to address false negatives. The F1-score provides a balanced measure of precision and recall, making it particularly useful for datasets with class imbalances. AUC evaluates the model's ability to differentiate between benign and malicious apps across various thresholds, highlighting its robustness. For instance, MEFDroid achieves a high F1-score of 97.12%, excelling in handling imbalanced datasets, while MalDetect demonstrates a strong AUC of 0.96, indicating its effectiveness in diverse environments. Incorporating these metrics into the analysis of the discussed techniques would provide a more comprehensive evaluation of their performance and practical applicability.

Data quality challenges significantly impact the performance of ML and DL-based malware detection systems. Data imbalance, where benign samples outnumber malware, leads to biased models with reduced recall and poor generalization. Noisy labels introduce errors, degrading precision and recall, while the lack of diverse, labeled datasets limits models' adaptability to evolving malware threats. To address these issues, synthetic data generation (e.g., SMOTE), cost-sensitive learning, and ensemble methods can mitigate imbalance, while robust data cleaning and noise-aware training improve resilience to labeling errors. Techniques like transfer learning, active learning, and crowdsourcing enhance the availability of labeled data, ensuring models remain robust and adaptable to modern malware challenges. These strategies collectively improve detection accuracy, reliability, and applicability in real-world scenarios.

Overall, each study reflects distinct strengths in terms of dataset suitability, real-time feasibility, and adaptability, making these methods suitable for different malware detection scenarios based on resource availability and specific application needs. The performance comparison in terms of precision, recall and F1-score is illustrated in Figure 1. It is observed that Hybrid model outperforms well in terms of accuracy, precision, recall and F1-score.

TABLE I. Expanded Methodology Comparison Table

Study	Approach	Technique	Dataset	Accuracy	Other Metrics
Efficient CNN	Deep Learning (CNN)	VGG16, DenseNet169, InceptionV3	APK Images	95.83%	Precision: 94.2%, Recall: 95.5%
Bayesian Method	Machine Learning (Bayesian)	Static Analysis using Permissions and API Calls	Custom Dataset	91.1%	False Positive Rate: 12%
MEFDroid	Ensemble Learning	Sparse Autoencoder + SDAE	Drebin, AndroMD	95.14%	F1-Score: 97.12%
Random Forest	Machine Learning	Random Forest for API Call and Permission Analysis	Drebin	94.36%	F1-Score: 88.75%
Transfer Learning	Transfer Learning	Pre-trained CNN (Inception)	OmniDroid	93.7%	Precision: 93.1%, Recall: 92.8%
Hybrid Model	Hybrid Static-Dynamic	API Call + System Logs	CICInvesAndMal2019	96.5%	F1-Score: 96.8%
ProDroid	Hybrid (Static & Dynamic)	Multiple Sequence Alignment (MSA) + PHMMs	Drebin, Contagio	95.7%	Precision: 94%, Recall: 96%, F1-Score: 95%
MalDetect	Hybrid Ensemble	Naive Bayes, J48, AdaBoost	Drebin, AMD, Genome	94.6%	Precision: 93%, F1-Score: 94%, AUC: 0.96
WHGDroid	Graph-Based Detection	Weighted Heterogeneous Graph + GNNs	Anzhi, AndroZoo	91.3%	F1-Score: 92%, Recall: 91%, AUC: 0.95
XManDroid	Ensemble for Inter-App Communication	ICC Monitoring + SVM, Random Forests	Custom Dataset (Inter-App Malware)	93.8%	Precision: 92%, Detection Rate: 94%
MADRF-CNN	Deep Learning Ensemble	CNN on Dex Files + Random Forest	Drebin, AMD, Contagio	96.1%	F1-Score: 95%, Precision: 94%, AUC: 0.97
KronoDroid	Incremental Learning	Self-Taught Learning	Drebin, Genome, Contagio	93.2%	Precision: 92%, Update Speed: High

TABLE 2. Expanded Performance Metrics Comparison Table

Study	Precision	Recall	F1Score	Efficiency (Time)
Efficient CNN	94.2%	95.5%	94.8%	High computation due to image processing
Bayesian Method	88.9%	87.5%	88.1%	Low computational cost
MEFDroid	97.1%	96.8%	97.12%	Moderate (Autoencoder training adds complexity)
Random Forest	90.2%	89.4%	88.75%	Moderate
Transfer Learning	93.1%	92.8%	92.95%	Moderate to high (due to fine-tuning pre-trained models)
Hybrid Model	96.8%	96.3%	96.5%	High efficiency in real-time detection
ProDroid	94%	96%	95%	High (Computationally intensive, longer processing time)
MalDetect	93%	94%	94%	Medium (Resource-intensive training but reasonable inference speed)
WHGDroid	92%	91%	91%	Medium to High (Graph processing can be time-consuming)
XManDroid	92%	94%	93%	Medium (Real-time processing but requires monitoring overhead)
MADRF-CNN	94%	95%	95%	Medium (Image conversion adds overhead but good inference time)
KronoDroid	92%	92%	92%	High (Real-time updates but potentially slower with noise)

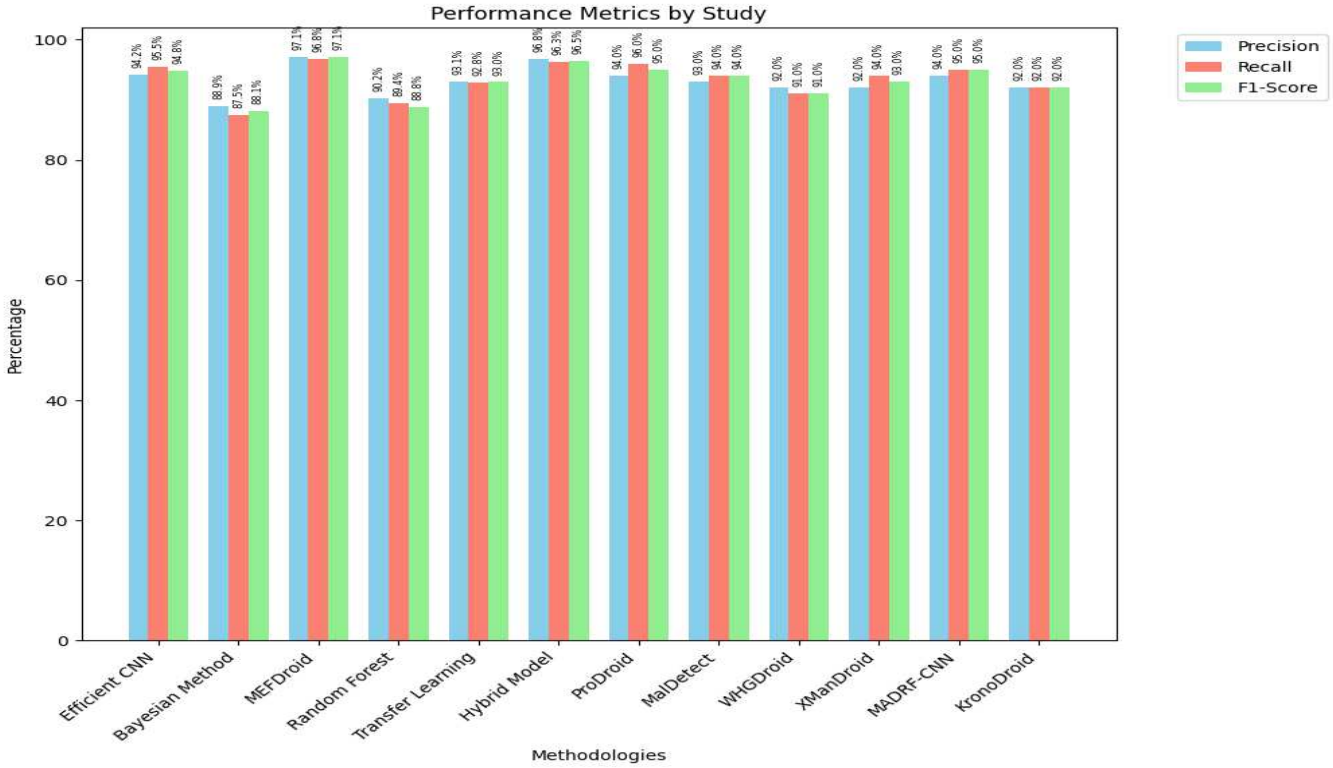

Fig.1. Performance comparison of existing methodologies

TABLE 3. Comparison of Real-Time Feasibility and Adaptability across Malware Detection Approaches

Study	Real-Time Feasibility			Adaptability		
	Score	Strengths	Weaknesses	Score	Strengths	Weaknesses
Efficient CNN	Low	High accuracy; Detects complex patterns	High computational cost; unsuitable for mobile devices	Moderate	Learns complex patterns, adaptable to diverse malware	Struggles with unseen threats without retraining
Bayesian Method	High	Lightweight; Ideal for mobile devices	Less effective against evolving threats	Low	Fast and simple model	Weak against new, unseen malware
MEFDroid	Moderate	Handles imbalanced datasets; Ensemble learning	High complexity; Requires computational power	High	Can detect evolving malware; Uses autoencoders for dynamic feature extraction	Complexity limits real-time retraining
Random Forest	Moderate	Efficient for static features; Handles moderate data	Prone to obfuscation; Limited by static analysis	Moderate	Flexible for structured data	Requires retraining for new malware variants
Transfer Learning	Moderate to Low	Pre-trained models reduce training time	Dependent on available pre-trained models	Moderate to High	Fast adaptation with pre-trained models	Limited by the scope of pre-trained data
Hybrid Model	High	Combines static and dynamic features; Good for real-time applications	More complex model, but worth the performance gain	High	Combines static and dynamic analysis, making it robust against zero-day malware	Requires extensive resource management for real-time deployment
ProDroid	Moderate to High	High accuracy in diverse environments; hybrid approach allows adaptability.	Computationally intensive, may slow down in low-resource settings	Moderate to High	Hybrid approach allows it to adapt to new threats effectively; good for varied environments.	Requires frequent retraining to maintain effectiveness against new malware types.
MalDetect	Moderate	Efficient for enterprise environments; ensemble methods enhance accuracy.	Requires substantial resources for training and may face latency during real-time operations.	Moderate to High	Ensemble methods enhance flexibility in adapting to different malware characteristics	Complexity can make it harder to adapt quickly to emerging threats.
WHGDroid	Moderate to Low	Effective in identifying inter-app communication	Graph-based processing can be time-consuming, impacting real-time	Moderate	Graph-based methods provide adaptability to inter-app communication	Adaptability is limited by the quality and comprehensiveness of the graph model.

		threats.	responsiveness		changes.	
XManDroid	Moderate	Good detection of malicious inter-app interactions; real-time monitoring capability.	May need constant updates to maintain effectiveness against evolving threats.	Moderate to High	Designed to monitor dynamic inter-app communications, allowing for real-time adaptability.	Relies heavily on continuous updates to remain effective against evolving threats.
MADRF-CNN	Moderate to High	Utilizes deep learning for high accuracy; relatively efficient inference.	Initial training time can be lengthy; may require significant computational power.	High	Deep learning model can improve with more data and adapt to new malware patterns over time.	Initial setup and training can be resource-intensive; slower to adapt if new data is sparse.
KronoDroid	Moderate	Adapts continuously to new threats; suitable for dynamic environments.	Vulnerable to noise in data, which could lead to inaccuracies in detection over time.	High	Incremental learning allows it to continuously adapt without full retraining; ideal for dynamic environments.	Vulnerable to noise in incoming data, which may hinder effective adaptation over time.

Hybrid Models, due to the combination of static and dynamic features, provide excellent precision and recall while remaining efficient enough for real-time application. Transfer learning Models, offer an attractive balance of performance and efficiency, especially when access to pre-trained models is feasible. Bayesian Models are resource-efficient, but fall short in handling modern, complex malware variants. This section presents a detailed evaluation of strengths and weaknesses of each approach in terms of real-time applicability and adaptability. Scores reflect computational efficiency, capability for continuous adaptation, and suitability for varying device environments, illustrating each effectiveness of each method in detecting evolving malware and handling different levels of system resources. The analysis is summarized using Table 3. The following discusses the present gap in every approach with future directions.

A. Real-Time Detection

The most accurate models, such as CNN-based approaches, are too computationally heavy for real-time applications. To address this, future work should focus on lightweight models or optimized neural networks like MobileNet or Tiny-YOLO, which are designed for mobile and edge devices with limited computational power. Leveraging edge-cloud architectures where most intensive computations are offloaded to the cloud while light detection models run on the device could offer a balance between real-time detection and accuracy

B. Dataset Diversity

Most studies rely on older datasets such as Drebin or AndroMD, which may not reflect the evolving nature of malware today. These datasets, while comprehensive, lack the latest zero-day malware and more sophisticated obfuscation techniques. Building and maintaining real-time malware datasets through active monitoring systems that collect malware from app stores and devices would help improve detection systems' adaptability. Datasets like MH-100K, which offer more representative and extensive samples, are a step in the right direction. Static feature-based models like Random Forests and Bayesian classifiers struggle with obfuscated malware, as they cannot capture runtime behaviors. Research should shift towards graph-based malware detection that can understand relationships between multiple entities (API calls, permissions) and offer better resilience to obfuscation techniques. Temporal Convolutional Networks (TCNs) could also be explored for analyzing time-based sequences (e.g., system calls).

C. Class Imbalance

Class imbalance remains a significant issue, especially in real-world applications where benign apps far outnumber malware. Although ensemble methods like MEFDroid address this, further work is needed to prevent overfitting on dominant benign samples. Techniques such as cost-sensitive learning, data augmentation (e.g., synthetic malware generation), and SMOTE (Synthetic Minority Over-sampling Technique) can help models learn from minority (malware) classes without bias.

D. Interpretability

Interpretability remains a problem for deep learning models like CNNs, which operate as black boxes. This lack of transparency can hinder trust and acceptance in real-world cyber security applications. Integrating explainable AI (XAI) techniques like LIME (Local Interpretable Model-agnostic Explanations) or SHAP (Shapley Additive Explanations) [29] can offer insight into which features contributed most to a model's classification. Additionally, attention-based neural networks can highlight specific sections of code or API calls that led to a malware classification.

IV. Conclusion

In summary, this review of Android malware detection methods highlights the strengths and limitations of various approaches. CNN-based models offer the highest accuracy but come with high computational costs, making them less suitable for real-time detection on mobile devices. Ensemble methods such as MEFDroid are highly accurate and robust in handling imbalanced datasets, though they come at the cost of complexity and slower adaptation to new malware types. For the future, hybrid models that combine both static and dynamic features offer a promising balance between accuracy and efficiency. However, a key focus should be on creating more diverse, real-time datasets, addressing model interpretability, and improving real-time detection capabilities for malware detection systems.

References

[1] A. Ksibi, M. Zakariah, L. Almuqren, and A. S. Alluhaidan, ''Efficient Android malware identification with limited training data utilizing multiple convolution neural network techniques,'' Eng. Appl. Artif.

Intell., vol. 127, Jan. 2024, Art. no. 107390, doi: 10.1016/j.engappai.2023.107390

[2] S.R.T. Mat, M.F.A. Razak, M.N.M. Kahar, J.M. Arif, A. Firdaus, A Bayesian probability model for Android malware detection, ICT Express 8 (3) (2022) 424–431, https://doi.org/10.1016/j.icte.2021.09.003

[3] A. Guerra-Manzanares, M. Luckner, and H. Bahsi, "Android malware concept drift using system calls: Detection, characterization and challenges," Expert Syst. Appl., vol. 206, 2022, Art. no. 117200.

[4] H. J. Zhu,L. Yang, L. M. Wang, and V. S. Sheng, "A multi-model ensemble learning framework for imbalanced android malware detection," Expert Systems with Applications, vol. 234, pp. 120952, 2023

[5] Dhalaria M, Gandotra E. MalDetect: A classifier fusion approach for detection of android malware. Expert Syst Appl 2024;235:121155

[6] P. Bhat and K. Dutta, ''A multi-tiered feature selection model for Android malware detection based on feature discrimination and information gain,'' J. King Saud Univ.-Comput. Inf. Sci., vol. 34, no. 10, pp. 9464–9477, Nov. 2022

[7] P. Bhat and K. Dutta, ''A multi-tiered feature selection model for Android malware detection based on feature discrimination and information gain,'' J. King Saud Univ.-Comput. Inf. Sci., vol. 34, no. 10, pp. 9464–9477, Nov. 2022

[8] T. Islam, S. S. M. M. Rahman, M. A. Hasan, A. S. M. M. Rahaman, and M. I. Jabiullah, ''Evaluation of N-gram based multi-layer approach to detect malware in Android,'' Proc. Comput. Sci., vol. 171, pp. 1074–1082, Jan. 2020.

[9] A. K. Singh, G. Wadhwa, M. Ahuja, K. Soni, and K. Sharma, ''Android malware detection using LSI-based reduced opcode feature vector,'' Procedia Comput. Sci., vol. 173, pp. 291–298, 2020.

[10] A. Roy, D.S. Jas, G. Jaggi, K. Sharma "Android Malware Detection based on Vulnerable Feature Aggregation" Procedia Comput. Sci., 173 (2019) (2020), pp. 345-353, 10.1016/j.procs.2020.06.040

[11] O. N. Elayan and A. M. Mustafa, "Android malware detection using deep learning," Procedia Computer Science, vol. 184, no. 2, pp. 847–852, 2021.

[12] V. Syrris, D. Geneiatakis, On machine learning effectiveness for malware detection in Android OS using static analysis data, J. Inf. Secur. Appl. 59 (May) (2021) 102794, https://doi.org/10.1016/j.jisa.2021.102794.

[13] Sihag V, Vardhan M, Singh P. BLADE: robust malware detection against obfuscation in android. Forensic Sci Int: Digital Investig 2021 Sep;1(38):301176

[14] Bashir, S., Maqbool, F., Khan, F.H., Abid, A.S., 2024. Hybrid machine learning model for malware analysis in android apps. Pervasive Mob. Comput. 97, 101859. http: //dx.doi.org/10.1016/j.pmcj.2023.101859.

[15] D.O. ¨ S¸ ahin, O.E. Kural, S. Akleylek, E. Kılıç, Permission-based Android malware analysis by using dimension reduction with PCA and LDA, J. Inf. Secur. Appl. 63 (October) (2021) 102995, https://doi.org/10.1016/j.jisa.2021.102995.

[16] H. Rathore, A. Nandanwar, S. K. Sahay, and M. Sewak, ''Adversarial superiority in Android malware detection: Lessons from reinforcement learning based evasion attacks and defenses,'' Forensic Sci. Int., Digit. Invest., vol. 44, Mar. 2023, Art. no. 301511.

[17] M. Alejandro, L.-C. Raul, and D. Camachoa, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," Inf. Fusion, vol. 52, no. 1, pp. 128–142, 2019.

[18] D. Saif, S.M. EI-Gokhy, E. Sallam, Deep belief networks-based framework for malware detection in Android systems, Alexandria Eng. J. 57 (2018) (2018) 4049–4057

[19] A. S. Shatnawi, Q. Yassen, and A. Yateem, "An Android Malware Detection Approach Based on Static Feature Analysis Using Machine Learning Algorithms," Procedia Computer Science, vol. 201, pp. 653-658, 2022/01/01/ 2022, doi: https://doi.org/10.1016/j.procs.2022.03.086.

[20] M. Alejandro, L.-C. Raul, and D. Camachoa, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the 50dataset," Inf. Fusion, vol. 52, no. 1, pp. 128–142, 2019.

[21] Sasidharan, S.K.; Thomas, C. ProDroid—An Android malware detection framework based on profile hidden Markov model. Pervasive Mob. Comput. 2021, 72, 101336.

[22] AlOmari, H.; Yaseen, Q.M.; Al-Betar, M.A. A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection. Procedia Comput. Sci. 2023, 220, 763–768

[23] L. Huang, J. Xue, Y. Wang, Z. Liu, J. Chen, Z. Kong, Whgdroid: effective Android malware detection based on weighted heterogeneous graph, J. Inf. Secur. Appl. 77 (2023) 103556, https://doi.org/10.1016/j.jisa.2023.103556.

[24] Bragança H, Rocha V, Barcellos L, Souto E, Kreutz D, Feitosa E. Android malware detection with MH-100K: An innovative dataset for advanced research. Data Brief. 2023 Nov 2;51:109750. doi: 10.1016/j.dib.2023.109750. PMID: 38020437; PMCID: PMC10661696.

[25] Arif, J.M.; Ab Razak, M.F.; Tuan Mat, S.R.; Awang, S.; Ismail, N.S.N.; Firdaus, A. Android mobile malware detection using fuzzy AHP. J. Inf. Secur. Appl. 2021, 61, 102929.

[26] Razgallah, A.; Khoury, R.; Hallé, S.; Khanmohammadi, K. A survey of malware detection in Android apps: Recommendations and perspectives for future research. Comput. Sci. Rev. 2021, 39, 100358.

[27] H. Zhu, H. Wei, L. Wang, Z. Xu, and V. S. Sheng, ''An effective endto-end Android malware detection method,'' Exp. Syst. Appl., vol. 218, May 2023, Art. no. 119593, doi: 10.1016/j.eswa.2023.119593.

[28] G. Renjith and S. Aji, ''On-device resilient Android malware detection using incremental learning,'' Proc. Comput. Sci., vol. 215, pp. 929–936, Jan. 2022, doi: 10.1016/j.procs.2022.12.095

[29] Martin Kinkead, Stuart Millar, Niall McLaughlin, and Philip O'Kane. Towards explainable CNNs for android malware detection. Procedia Computer Science, 184:959–965, 2021.