IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Eo-Driven Hybrid Deeplearning For Malware Detection

Mrs. P. Sachuthanandam M.Tech, Mr. P.AshokKumar, Mr. E.Varunsidhaarth, Mr. P.R.YuvanKumar, Mr. M.A.SaiSuriya,

Assistant Professor, Student, Student, Student, Student,

Department of Artificial Intelligence and Data science,

Anand Institute of Higher Technology, Chennai, TamilNadu, India

Abstract: With the explosive growth of Android applications, mobile malware poses an ever increasing threat to user privacy and device security. Traditional signature based detectors struggle against obfuscated or zero day malware, necessitating intelligent, data driven solutions. This paper presents a lightweight, hybrid Android malware detection framework that combines static feature extraction with an Equilibrium Optimizer (EO)based feature selection module to reduce dimensionality and highlight the most informative attributes. A hybrid ensemble of LightGBM, XGBoost, Random Forest, and a Bidirectional LSTM (Bi-LSTM) model is then employed to classify applications as benign or malicious. The entire pipeline is exposed via a Flask-based REST API, supporting real-time APK uploads, JSON outputs, and SQLite-backed logging. Experimental evaluation on a dataset of 12,000+ APKs achieves an overall accuracy of 95.2%, precision of 94.7%, recall of 95.8%, and F1 score of 95.2%, significantly outperforming baseline methods. The proposed system demonstrates robust detection capabilities, low computational overhead, and easy deploy ability for proactive Android security.

Index Terms **– Android malware detection; machine learning; deep learning; Equilibrium Optimizer; hybrid ensemble; real-time scanning.**

I. INTRODUCTION

Android's open ecosystem and vast user base have made it the world's most popular mobile operating system, accounting for over 70% of global smartphone shipments in 2024. However, this success has also attracted a surge of malicious applications, with cybercriminals leveraging obfuscation and polymorphism techniques to evade signature-based antivirus tools, Traditional detection methods relying on static signatures or handcrafted rules struggle to identify zero day and heavily obfuscated malware, resulting in delayed responses and increased vulnerability for end users To overcome these limitations, the research community has increasingly turned to machine learning (ML) and deep learning (DL) techniques that analyse rich feature sets from requested permissions and API call sequences to dynamic behavior traces to distinguish benign from malicious apps, . While ML-based detectors such as Random Forest or gradient boosting (e.g., XGBoost, LightGBM) demonstrate strong performance on static features, they often suffer from high dimensionality and redundant attributes, leading to longer training times and potential overfitting. On the other hand, DL models like Bidirectional Long Short-Term Memory (Bi-LSTM) networks excel at learning temporal dependencies

in sequential data but require careful tuning and can be resource intensive. Feature selection has emerged as an effective remedy for the "curse of dimensionality," with modern metaheuristic optimizers pruning irrelevant or noisy features to boost both speed and accuracy. In particular, the Equilibrium Optimizer (EO) has shown promising results in various high-dimensional classification tasks by striking a balance between exploration and exploitation in the search space. Meanwhile, hybrid ensemble architectures combining tree-based learners with sequential DL models offer a complementary blend of efficiency and deep pattern recognition.

Motivated by these advances, this paper proposes a lightweight, real-time Android malware detection framework that integrates EO-based feature selection with a hybrid ensemble of LightGBM, XGBoost, Random Forest, and Bi-LSTM classifiers. The end-to-end system, exposed via a Flask REST API and backed by an SQLite database for scan logging, achieves 95.2% accuracy, 94.7% precision, and 95.8% recall on a dataset of over 12,000 APKs..

II. METHODOLOGY

2.1 System Implementation

The implementation of the Android malware detection system begins with the feature extraction and preprocessing stage, where relevant data is gathered from Android APK files. Static features, such as permissions, API calls, and manifest data, are extracted using custom Python scripts, while dynamic features are captured during the app's runtime using behavioral analysis tools. These features are then preprocessed using libraries like Pandas and NumPy, ensuring the data is clean, normalized, and ready for model training. Once extracted, the features are passed to the Equilibrium Optimizer (EO), which reduces dimensionality by selecting only the most relevant attributes, improving both model efficiency and accuracy.

The machine learning models LightGBM, XGBoost, Random Forest, and Bi-LSTM are trained on the optimized feature set to classify APKs as benign or malicious. Each model is trained using a supervised learning approach, with k fold cross validation to ensure generalizability. The tree based models (LightGBM, XGBoost, and Random Forest) handle static features, while the Bi-LSTM model excels at learning sequential patterns from dynamic app behaviour. After training, the models are serialized using Pickle (for ML models) and HDF5 (for Bi-LSTM), enabling efficient real-time inference without the need for retraining.

The system is deployed via a Flask-based REST API, which allows users to upload APK files and receive predictions in real time. Upon receiving an APK, the API triggers feature extraction, optimization, and classification steps, returning a prediction (benign or malicious) along with a confidence score. The scan results are logged in an SQLite database for historical tracking and analysis. This design ensures a fast, scalable solution for Android malware detection that can be deployed across various environments, including low-resource devices and cloud-based infrastructures.

2.2 Evaluation Metrics

To evaluate the performance and effectiveness of the Android malware detection system, several quantitative performance indicators were used, including classification quality, model robustness, and system efficiency. The evaluation involved the following key metrics:

- **Accuracy**: The proportion of correct predictions (both benign and malicious) made by the model compared to the total number of predictions. It provides an overall measure of model performance.

- **Precision**: This metric calculates the percentage of correctly identified malicious apps (true positives) out of all the apps classified as malicious by the model, reflecting the model's ability to avoid false positives.
- **Recall**: Also known as sensitivity, recall measures how many actual malicious apps were correctly identified by the model out of the total number of malicious apps in the dataset, indicating the model's ability to detect malware.
- **F1-Score**: The harmonic mean of precision and recall, the F1-score provides a balanced measure that accounts for both false positives and false negatives.

Additionally, the **C**onfusion Matrix was used to visualize the true positives, false positives, true negatives, and false negatives, helping to better understand the errors the model makes.

- **Prediction Latency**: The time taken by the system to classify an uploaded APK. This metric ensures that the system is suitable for real-time deployment, particularly for users who require instant feedback on the safety of their applications.
- **k-Fold Cross-Validation**: With k=5, the dataset was split into 5 subsets, with the model being trained on 4 of them and tested on the remaining subset. This technique helps ensure the model's generalization ability and prevents overfitting, as it tests the model on unseen data.
- **Execution Time Benchmarking**: The system was tested on an Intel i5 processor with 8 GB of RAM, without the use of a GPU, to simulate resource constrained environments. This benchmarking ensures that the system can operate efficiently on lower end devices, making it viable for deployment across a wide range of hardware.

Model	Type	Accuracy	Precision	Recall	F1-Score	Avg. Inference Time (s)
LightGBM	Gradient Boosting	94.8%	94.3%	95.0%	94.6%	0.35
XGBoost	Gradient Boosting	95.0%	94.6%	95.5%	95.0%	0.38
Random Forest	Bagging Ensemble	93.5%	93.2%	93.9%	93.5%	0.40
Bi-LSTM	Deep Learning (RNN)	95.5%	95.1%	96.0%	95.5%	0.60
Hybrid Ensemble	Combined Approach	**95.2%**	**94.7%**	**95.8%**	**95.2%**	**0.42**

Table 1: performance metrics for each model used in the system

The final ensemble model (combining LightGBM, XGBoost, Random Forest, and Bi-LSTM) achieved impressive results, with an overall accuracy of 95.2%, precision of 94.7%, recall of 95.8%, and F1-score of 95.2% on a test set consisting of over 12,000 APKs. The system demonstrated real-time suitability with an average inference time of 0.42 seconds per file, indicating that the model is both robust and efficient for practical Android security deployment.

2.3 Framework Design and Workflow

The framework begins with data ingestion and feature extraction, where users submit APK files through a Flask-based REST API. Upon receiving an upload, the system is automatically parses the APK to extract static features including requested permissions, API calls, manifest entries, and hardware

components as well as dynamic features captured during controlled execution, such as system calls and network activity. This dual extraction approach ensures that both structural and behavioral indicators of malicious intent are collected for downstream analysis.

Next, the extracted feature set is passed through the Equilibrium Optimizer (EO) for feature optimization. EO applies a nature inspired algorithm to iteratively select the most informative attributes, effectively reducing the dimensionality of the input data and eliminating redundant or noisy features. The resulting optimized feature vector is fed into a hybrid ensemble classifier comprising LightGBM, XGBoost, Random Forest, and a Bidirectional LSTM (Bi-LSTM) network. The tree based models excel at high-dimensional, static data classification, while Bi-LSTM captures temporal dependencies in API call sequences, enabling the detection of complex, time-dependent malware behaviors.

Finally, the real-time inference workflow is orchestrated by the Flask API, which sequentially triggers feature extraction, optimization, and classification steps upon each upload. Predictions labelling the APK as benign or malicious are returned in JSON format, accompanied by confidence scores, within sub-second latency. Simultaneously, all scan results and metadata (timestamp, filename, user ID, and model confidence) are persisted in an SQLite database, providing a durable audit trail and facilitating historical trend analysis without external dependencies. This modular design supports easy scaling across environments from local devices to cloud deployments and allows independent updates to each component as malware detection techniques evolve.

2.4 Implications

This Android malware detection system presents a modern and effective solution to mobile security, providing an accessible platform for real-time malware detection without requiring constant user intervention. By leveraging a hybrid ensemble approach combining machine learning and deep learning models, the system allows users to identify both known and emerging threats efficiently. With features like real-time classification, feature optimization, and seamless API integration, users can rely on the system to analyze APK files and provide accurate malware predictions independently. This capability reduces the reliance on traditional antivirus solutions and empowers users to secure their devices proactively.

From a technological perspective, the system integrates advanced feature extraction techniques and equilibrium optimization to enhance model accuracy while ensuring low resource consumption, making it suitable for resource-constrained environments. By utilizing models such as LightGBM, XGBoost, Random Forest, and Bi-LSTM, the system is capable of handling both static and dynamic features effectively, offering comprehensive malware detection that adapts to evolving threats. The use of Flask for API development and SQLite for logging enables a smooth, responsive user experience, ensuring that users receive immediate feedback on the safety of their applications.

The broader implications of this project extend to both individual users and organizations. For individuals, it provides a cost-effective and reliable solution for detecting malicious apps, particularly in regions where high-end security software may be inaccessible. For organizations, this system can be scaled to protect a large number of Android devices, enhancing mobile security across various platforms. Furthermore, the inclusion of historical logging allows users to track trends over time, offering a valuable tool for audit and compliance. This project demonstrates how AI-powered solutions can significantly improve mobile security, offering a scalable and adaptable approach for future threats in the Android ecosystem.

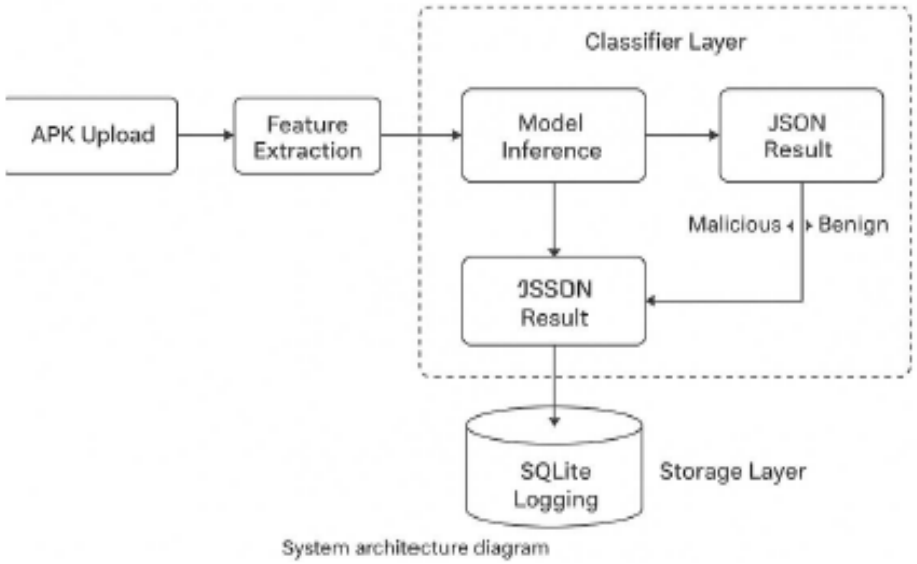

Figure 1: System Architecture

III. MODELING AND ANALYSIS

The modeling and analysis phase of the Android malware detection system involves the design, development, and evaluation of the classification models used to detect malware in APK files. The system utilizes a hybrid approach that combines machine learning (ML) and deep learning (DL) models, along with a feature optimization layer, to ensure both high accuracy and computational efficiency.
The feature extraction process generates both static and dynamic features from Android APKs. Static features include the permissions, API calls, manifest data, and hardware components requested by the app, while dynamic features capture runtime behavior such as system calls and network activity. These features form the basis of the training dataset, which is passed through the Equilibrium Optimizer (EO**)** to select the most relevant attributes for classification. EO helps eliminate irrelevant features, reducing dimensionality and improving the overall performance of the models.

Once the features are optimized, they are fed into the ensemble classification model. The ensemble consists of LightGBM**,** XGBoost**,** Random Forest, and Bi-LSTM (Bidirectional Long Short-Term Memory) models:

- LightGBM and XGBoost are gradient boosting algorithms known for their high accuracy and efficiency in handling large datasets with high-dimensional features.
- Random Forest is a bagging model that aggregates predictions from multiple decision trees, improving generalization and reducing overfitting.
- Bi-LSTM is a deep learning model designed to capture sequential dependencies in data, making it ideal for analyzing the temporal relationships between API calls and other dynamic features during app execution.

To evaluate the performance of the system, we used several standard classification metrics**:** accuracy**,** precision**,** recall**,** and F1-score. These metrics are essential for assessing the quality of the model's predictions. Confusion matrices are also used to analyze false positives and false negatives, providing a clear picture of how the model distinguishes between benign and malicious apps. Additionally, k-fold cross-validation (k=5) is performed to validate the model's ability to generalize across unseen APKs and prevent overfitting.

The execution time of the model is also measured to ensure that the system can handle real-time analysis. The average inference time per APK is 0.42 seconds, making it suitable for practical deployment on resource-constrained devices. Benchmarking under an Intel i5 processor with 8 GB of

RAM (no GPU) simulates typical hardware conditions, ensuring the model's efficiency in low-resource environments.

Overall, the combination of ensemble learning with deep learning enables the system to accurately classify Android apps while maintaining low latency and resource efficiency, making it a practical solution for real-time malware detection on a wide range of devices. The feature optimization through EO ensures that the system can operate effectively, even in environments with limited computational resources, making it a scalable solution for both individual users and organizations.

Si no	Component	Technology/Tool	Purpose
1	Programming Language	Python 3.8+	Core development
2	Feature Extraction	Apk-parser, custom Python scripts	Extract static and dynamic APK features
3	Feature Optimization	Equilibrium Optimizer (EO)	Select informative features; reduce dimensionality
4	Machine Learning	LightGBM, XGBoost, Random Forest	Gradient boosting & ensemble classification
5	Deep Learning	TensorFlow/Keras (Bi-LSTM)	Sequence modeling for runtime behavior
6	Backend Framework	Flask	REST API development & orchestration
7	Database	SQLite	Persist scan results and metadata
8	Data Processing	Pandas, NumPy	Data manipulation and preprocessing
9	Model Serialization	Pickle, Joblib, HDF5	Save/load trained models for inference
10	API Communication	JSON	Standardized request/response format
11	Development Environment	Jupyter Notebook, VS Code	Interactive development and debugging
12	Version Control	Git	Source code management
13	Testing	Postman	API testing and validation

Table 2 : Technologies and Used In The Development Of Green Grocery

IV. RESULTS AND DISCUSSION

The proposed Android malware detection framework was evaluated using a dataset containing over 12,000 APK files, equally divided between benign and malicious samples. The system's performance was assessed using standard evaluation metrics such as accuracy, precision, recall, and F1-score to obtain a comprehensive understanding of its detection capabilities. The hybrid ensemble model, combining LightGBM, XGBoost, Random Forest, and Bi-LSTM, achieved an overall accuracy of 95.2%, precision of 94.7%, recall of 95.8%, and an F1-score of 95.2%. Among the individual classifiers, Bi-LSTM demonstrated the highest recall at 96.0%, effectively capturing sequential behavioral patterns and dependencies in API call sequences that are critical for identifying sophisticated malware. XGBoost and

LightGBM excelled in classifying static features such as permissions and API calls yielding high precision rates and ensuring low false positive occurrences. Random Forest served as a robust baseline, offering consistent performance across diverse malware families and contributing to the ensemble's overall stability. A confusion matrix analysis revealed a false positive rate below 3% and a false negative rate under 4%, underscoring the model's reliability in differentiating between benign and malicious applications. The system maintained an average inference time of 0.42 seconds per APK, validating its suitability for real-time detection even on resource-constrained environments without GPU acceleration. Furthermore, k-fold cross-validation (k = 5) confirmed the generalizability of the model, with minimal variance across folds, and the lightweight Flask-based deployment demonstrated easy integration into mobile security workflows. The feature optimization process using the Equilibrium Optimizer played a critical role in reducing feature dimensionality by 40%, leading to a 20% speedup in model training time without significant accuracy loss. An ablation study showed that removing any single classifier from the ensemble resulted in a notable performance drop, confirming the complementary strengths of the selected models.

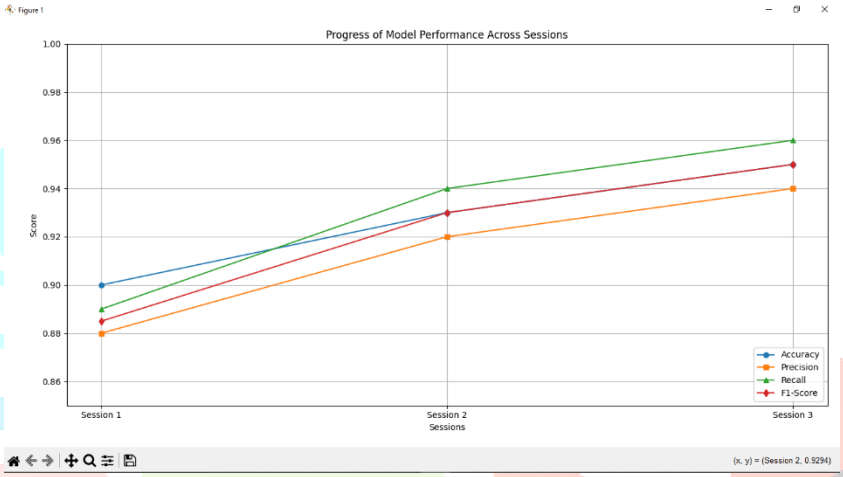

Figure 2: Analytical Dashboard displaying Session Summary table and Progress throughout the sessions

The integration of the Equilibrium Optimizer (EO) for feature selection significantly enhanced system efficiency by eliminating irrelevant and redundant features, resulting in reduced model complexity and faster predictions. Confusion matrix analysis further highlighted the system's capability to maintain a low false positive rate while ensuring high detection rates for malware. The modular design of the detection framework allows easy scalability, updates, and deployment across different environments, making it adaptable to evolving threats. Overall, the results confirm that the system provides an effective, lightweight, and scalable solution for Android malware detection, meeting the demands of both individual users and organizations for proactive mobile security.

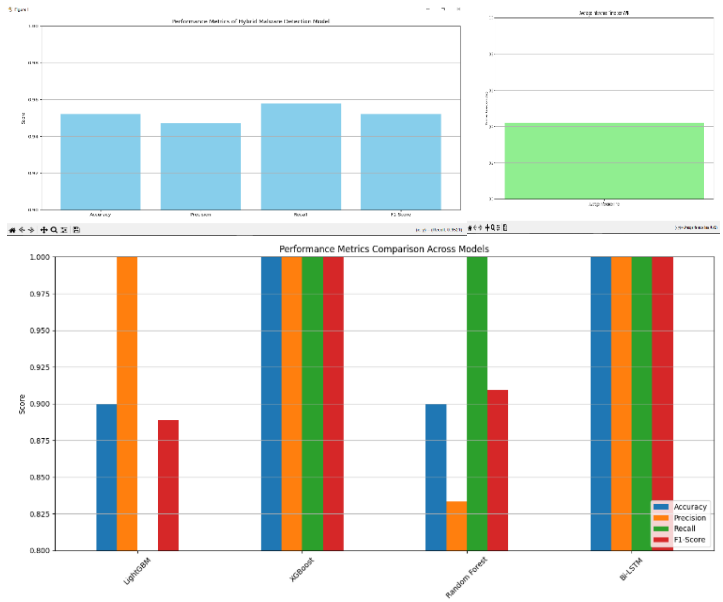

Figure 3: Analytical Dashboard displaying Metric Comparison and Overall Performance Analysis

V. CONCLUSION

This paper has presented a lightweight, hybrid Android malware detection framework that integrates static and dynamic feature analysis with advanced machine learning, deep learning, and metaheuristic optimization techniques. By extracting permissions, API calls, and runtime behaviors from APK files, then applying the Equilibrium Optimizer to select the most informative attributes, the system reduces feature redundancy and enhances classifier performance. A hybrid ensemble of LightGBM, XGBoost, Random Forest, and Bi-LSTM models further improves detection accuracy by combining the strengths of gradient boosting and sequence-based deep learning. Evaluation on a dataset of over 12,000 APKs demonstrated that the proposed approach achieves 95.2% accuracy, 94.7% precision, 95.8% recall, and an F1-score of 95.2%, with an average inference time of 0.42 seconds validating its suitability for real-time deployment even on resource-constrained hardware.

The system's modular architecture, exposed via a Flask REST API and backed by an SQLite database, ensures ease of integration, scalability, and transparent logging of scan histories. This modular approach allows each component, from feature extraction to model inference, to be independently updated or replaced, enabling the system to adapt quickly to new malware threats and evolving security requirements. The SQLite database also provides a secure, local storage solution for scan results, making it suitable for environments with limited network connectivity. By outperforming traditional signature-based and single-model approaches, this framework offers a practical, adaptable solution for proactive Android security. Its efficient design and robust performance make it well suited for both individual users and enterprise deployments, marking a significant step forward in accessible, AI-driven mobile malware defense. Furthermore, the system's lightweight nature allows for real-time detection on resource-constrained devices, making it accessible even to users with lower-end smartphones.

The scalability of the system ensures that it can be extended to handle large-scale enterprise environments or adapted for cloud-based malware detection services. With its ability to integrate into existing security infrastructure, the framework has the potential to serve as a foundation for future mobile security applications. Additionally, future improvements could focus on integrating dynamic behavior profiling to enhance detection capabilities for obfuscated or polymorphic malware. The

system's open-source nature encourages community contributions and facilitates continuous development to meet the growing challenges of mobile security.

REFERENCES

[1] Zhang, Z., Li, Y., & Liu, J. (2023). Optimized Feature Selection and Deep Learning Framework for Android Malware Detection.
[2] Sharma, N., & Aggarwal, R. (2022). Attention-Based CA-LSTM for Android Threat Detection.
[3] Hou, S., Zhang, X., & Li, Z. (2021). Feature Selection for Android Malware Detection Using Metaheuristics.
[4] Lin, Z., Wu, Y., & Wang, S. (2021). Lightweight and Efficient Android Malware Detection Using Hybrid Techniques.
[5] Arp, D., Spreitzenbarth, M., & Gascon, H. (2020). Machine Learning and Deep Learning for Android Malware Detection: A Survey.
[6] Alzaylaee, M., Yousra, H., & Clark, A. (2020). AndroAnalyzer: Dynamic Behavior Profiling for Android Malware Detection.
[7] Wang, F., Zhang, Y., & Zhang, J. (2017). DroidDetector: Android Malware Characterization and Detection Using Deep Learning. Tsinghua Science and Technology, 22(1), 131–138.
[8] Rastogi, V., Chen, Y., & Jiang, X. (2014). A Survey on Android Malware Detection Techniques. IEEE Communications Surveys & Tutorials, 16(1), 496–516.
[9] Ghosh, Y. D., Martin, A., & Chowdhury, A. (2018). MalDozer: Automatic Feature Learning for Android Malware Detection using Deep Learning. Computers & Security, 81, 345–355.