**AI-BASED ANDROID MALWARE SCANNER**

Tan Yong Chien1, a) and Ang Ee Mae1, 2, b)

1Faculty of Information Science and Technology (FIST), Multimedia University, Jalan Ayer Keroh Lama, 75450 Bukit Beruang, Melaka, Malaysia

2Centre for Intelligent Cloud Computing (CICC), Multimedia University, Jalan Ayer Keroh Lama, 75450 Bukit Beruang, Melaka, Malaysia

b) Corresponding author: eemae@mmu.edu.my

a)llamameYC@gmail.com

**Abstract.** The increasing number of Android users has made the platform a prime target for cyberattacks, necessitating more advanced malware detection solutions. Traditional signature-based methods struggle to keep up with evolving threats, highlighting the need for AI-driven approaches. An Android malware scanner is proposed to utilize feature representations pretrained by Sequential Neural Network and further classified by XGBoost for the final prediction. The model analyzes the applications consensual metadata and intents to efficiently detect potential malware. The proposed Hybrid Sequential Attention Network model contributes to the field by exploring lightweight AI-driven detection methods that require minimal manual intervention and offers a complementary approach to existing detection methods, reducing reliance on manually updated signature databases by leveraging static analysis with deep learning. In contrast to traditional Android security techniques, the model improves detection speed and adaptability while maintaining a minimal false positive rate. The proposed model demonstrates an accuracy of 94.35%, precision of 94.05%, recall of 93.30% and F1-score of 94.12%, outperforming existing models thus proving the effectiveness in identifying both known and emerging threats.

# Introduction

Android stands as the best-selling mobile operating system, commanding 72.15% of the global marketplace [1]. The diversity, convenience, and open-source nature of Android encourage widespread adoption, but also increase its exposure to persistent cyber threats. Malicious software (malware) is growing at an alarming rate, posing serious security challenges to users and underlining the urgent need for practical and effective malware detection solutions [2]. In real-world applications, malware scanners offer a practical first line of defence by automatically analysing applications before installation or during runtime, significantly reducing user risk with minimal manual effort. However, traditional malware detection techniques such as manual inspection and signature-based methods, fall short in modern threat landscapes [3][4]. These methods struggle with detecting zero-day attacks, generalizing to novel malware variants, and keeping up with the sheer volume and variety of Android applications. Consequently, there is a growing demand for intelligent, adaptive systems that can proactively identify evolving malware threats.

# RELATED WORK

The widespread use of Android creates the perfect environment for attackers to take advantage of security flaws using techniques like ransomware distribution, privilege escalation, phishing apps, and data exfiltration. Traditional techniques for spotting malware are not able to do well with new threats or those hidden in complicated ways, since they depend only on existing databases. To overcome these limitations, several advancements in malware detection now incorporate both static and dynamic analysis techniques [5]. Static analysis is the process of inspecting the source code or essential configuration details without executing the app, but dynamic analysis focuses on what the app does while running by detecting suspicious behavior that static tasks can miss.

Machine learning (ML) and deep learning (DL) integration into Android malware detection assists in signature-based approaches. Due to their simplicity and the fact that they are easy to interpret, K-Nearest Neighbors (KNN), Decision Trees (DT), Support Vector Machines (SVM), and Logistic Regression (LR) have been broadly applied. combination of features from both static and dynamic application analysis results in models that are good at detection. However, their ability to capture non-linear dependencies and contextual feature relationships remains limited.

DL models have gained traction, particularly various neural network architectures. These architectures can automatically learn hierarchical representations from app metadata, permissions, and intent sequences. Models like MalDozer [6] employed CNNs for static bytecode analysis, while DL-Droid [7] used LSTM networks on runtime behavior. With attention mechanisms, recent improvements in this field can now choose important patterns and boost the results and explanations of classification.

Combining DL feature extraction with ML classifiers such as SVM or XGBoost has been shown to improve generalization and robustness in hybrid frameworks [8]. The expressive capabilities of deep networks and the effectiveness of conventional algorithms in boundary learning on tabular data are advantageous to these models.

# METHODOLOGY

The study presents an AI-powered Android malware scanner that delivers rapid, efficient, user-focused protection against malware threats. To reduce this manual workload and the rate of false positives, the proposed system (Figure 1) takes advantage of ML in conjunction with DL models to improve malware detection. A hybrid DL detection model for Android malware for static app metadata without need to run applications in a sandbox or emulate runtime behaviour. Static analysis speeds up detection and drastically lowers resource consumption, which makes the model appropriate for real-time use on mobile devices.

A diagram of a network

AI-generated content may be incorrect.

**FIGURE 1.** The architecture of the proposed Hybrid Sequential Attention Network (HSAN) model

Static analysis is built on these two elements, permissions control what resources or actions an application can access, and intents show how components within and between apps communicate. When usage patterns are not normal, it may suggest that someone is using the system for malicious purposes [9][10]. The system reads the JSON files that come with each APK, and for every predefined permission and intent, it creates a binary vector with a 1 for present and 0 for absent [11]. Manual feature engineering is not required as it is in binary representation, the model works efficiently, and inference happens fast. It allows neural network to pick up discriminatory behavioural patterns linked to malicious applications.

As for handling the order and connections between the features, the model has two stacked BiGRU layers. With bidirectional configuration, the network looks at interactions from both directions, helping it detects hard-to-see details in the data [12]. Following BiGRU, an attention mechanism that is specifically built for the task is used. The mechanism helps the model concentrate on the key parts of the input sequence. It calculates attention scores for all the features that come from the BiGRU layers. They show how each feature helps in identifying if an application is harmful. Then, the mechanism gives different weights to each feature based on their significance and puts them all together into one summary vector. This helps the model focus on suspicious traits and not be influenced by unimportant ones, which improves its ability to classify things accurately.

The output of the attention layer is passed through a fully connected Dense layer with LeakyReLU activation, followed by Batch Normalization and Dropout layers. This approach helps the model do well on new data and helps avoid overfitting [13]. The last layer of the network has a sigmoid-activated neuron that estimates if the application is malicious. The model is trained with a binary cross-entropy loss function with label smoothing to minimize overconfidence in predictions [15] and the Adam optimizer with a learning rate of 0.0005 [14]. Further stabilization of training is achieved using callbacks like reducing the learning rate when no progress is seen and stopping the training early if the validation loss does not decrease.

Mixed precision training is applied using TensorFlow’s native support for float16 computation to optimize performance and reduce training time. To further boost classification performance and interpretability, a hybrid approach is introduced post-training. Feature representations are extracted from the penultimate Dense layer of the trained SNN because this layer encodes abstract and high-level patterns learned from the sequential feature data. These representations are then used as input to an XGBoost classifier, leveraging its robustness to overfitting, interpretability, and superior performance on tabular data [16]. This fusion enables better boundary learning than either model alone, allowing for more precise discrimination between benign and malicious applications. While the final prediction in the hybrid setup is made by the XGBoost classifier, the SNN forms the core of the architecture, serving as a deep feature extractor that captures complex sequential relationships from static metadata. The proposed solution does not rely on signature databases or frequent updates, which are often required by traditional antivirus scanners, making it more scalable and maintenance-free in practical deployments [17].

## Hyperparameters and Optimization

In our suggested Hybrid Sequential Attention Network (HSAN) framework, we used Grid Search to adjust the hyperparameters of the SNN backbone and the XGBoost classifier. Grid Search was selected because it systematically searches of different combinations of parameters to find optimal configurations. Extensive testing was made easier by Scikit-learn's GridSearchCV function, which produced dependable and reproducible outcomes [18]. Although, Random Search has faster exploration but may ignore the best combinations. Bayesian optimization is theoretically efficient but is infeasible to implement given its complexity and computational demands.

A separate experimental setup was used for hyperparameters optimization for the HSAN’s SNN component sensing using Grid Search. It was found that the Adam optimizer outperformed other such as SGD, RMSprop, and Nadam, resulting in the best balance between convergence speed and model accuracy [19]. Running the model with a learning rate of 0.0005 proved the optimal one because values higher than that caused the instability, while those lesser than it slows down the learning. The model was trained with a batch size of 32 and 30 training epochs with stable and consistent performance.

In dense layer, L2 regularization with λ as 0.001 was used to control the overfitting [20], dropout regularization of 0.4 was used over the hidden layers to improve generalization [21]. Hidden layer behaviours were set to LeakyReLU since it is robust against inactive neurons, and the output layer was set with sigmoid function as a way of knowing the binary malware classification. To highlight the most instructive features that the Bidirectional GRUs were able to extract, a custom attention layer was added. In addition, mixed precision training (mixed\_float16) was additionally used to increase computational efficiency and decrease memory usage. Table 1 provides a summary of hyperparameters for the HSAN framework setup.

**TABLE 1.** Hyperparameter Used in the HSAN framework

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SNN** | | | | **XGBoost** | |
| **Hyperparameter** | **Value** | **Hyperparameter** | **Value** | **Hyperparameter** | **Value** |
| Optimizer | *Adam* | Dropout Rate | *0.4* | Learning Rate | *0.05* |
| Learning Rate | *0.0005* | Hidden Layer Activation | *LeakyReLU* | n\_estimators | *200* |
| Batch Size | *32* | Output Activation | *Sigmoid* | max\_depth | *5* |
| Epochs | *30* | Attention Mechanism | *Custom Attention Layer* | subsample | *0.8* |
| L2 Regularization (λ) | *0.001* | Mixed Precision | *Enabled (mixed\_float16)* | colsample\_bytree | *0.7* |

# Comparative Analysis of HSAN

The proposed HSAN outperformed the other models in terms of overall performance, proving its resilience in detecting Android malware. Table 2 provides an overview of the various model’s performance metrics. HSAN outperformed the standalone SNN and conventional machine learning classifiers on all important metrics, achieving 94.35% accuracy, 94.95% precision, 93.30% recall, and 94.12% F1-score. The results highlight HSAN's ability to reduce both false positives and false negatives more accurately than these prior methods and thereby offers highly reliable performance for its deployment in real world settings which confirms that learning complex patterns from permissions and intents is possible in the base SNN model, thus surpassing other classifiers with 94.16% accuracy and 93.93% F1-score. Even though LR produced competitive metrics, it faces a linear nature of limitations in modeling nonlinear interactions in app metadata.

Support Vector Machines (SVM) and K-Nearest Neighbors (KNN) showed strong but slightly lower F1-scores, indicating a less optimal balance between precision and recall. Interpretable in Decision Tree model, although lowest performance, indicated that simpler tree-based methods cannot accurately generalize on a complicated Android malware dataset. Compared to other approaches, our proposed DL-based approach, HSAN, is efficient in detecting malicious applications, provides the best detection ability, as well as a healthy system integrity and protection over mobile users. Table 2 gives an overview of the performance metric results for the tested models.

**TABLE 2.** Performance metrics of various models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy (%)** | **Precision (%)** | **Recall (%)** | **F1-Score (%)** |
|  | **94.35** | **94.95** | **93.30** | **94.12** |
| SNN | 94.16 | 94.70 | 93.17 | 93.93 |
| LR | 93.27 | 93.48 | 92.56 | 93.02 |
| SVM | 92.96 | 93.55 | 91.81 | 92.67 |
| KNN | 92.42 | 93.04 | 91.19 | 92.11 |
| Decision Tree | 92.18 | 92.25 | 91.56 | 91.91 |

## Confusion Matrix for Model Performance Evaluation

An analysis of the proposed model's performance was conducted through a confusion matrix plot to provide supplementary information regarding classification results. The classification scheme distinguishes between True Positives (TP) and False Positives (FP) as well as True Negatives (TN) and False Negatives (FN).

Figure 2 illustrates the confusion matrices for the models. In contrast to other models, the Decision Tree model tends to produce somewhat more FP (62) and FN (68), despite having a reasonably balanced prediction. The Decision Tree model and KNN have a similar performance, whilst there are fewer FP (55), they are also more FN (71) which means that it might miss some malicious instances. There are relatively few FP (51), FN (66), TP (740) and TN (806) in the SVM model. This indicates SVM has a good tradeoff between the ability to correctly classify benign and malicious instances and errors. Similarly to the SVM model, but with a marginally higher number of TP (746), is the LR model. Additionally, it has low FP and FN rates. The SNN model has a TP of 750 and a TN of 819. It also has relatively low FN (56) and the lowest FP (38), which shows that it is the most accurate to determine whether a sample is malicious or not. The HSAN model achieves a strong balance between detecting benign and malicious applications. It correctly classifies 817 benign and 752 malicious instances, with only 40 FP and 54 FN.

A blue squares with white text

AI-generated content may be incorrect.A blue squares with white text

AI-generated content may be incorrect.A blue squares with white text

AI-generated content may be incorrect.A blue squares with white text

AI-generated content may be incorrect.

(a) KNN

(c) SVM

(b) Decision Tree

A graph of a number of different colored squares

AI-generated content may be incorrect.A blue squares with white text

AI-generated content may be incorrect.A blue squares with white text

AI-generated content may be incorrect.

(f) HSAN

(e) SNN

(d) Logistic Regression

**FIGURE 2.** Confusion matrices for HSAN and other models used in Android malware detection

With the highest combined number of accurate classifications (TP and TN) and the lowest combined number of errors (FP and FN), the HSAN model outperforms the others based on the confusion matrices. The TP and FP for the SNN model are close to those of the HSAN model with the lowest FP and slightly higher TN, but by striking a better balance with fewer FN’s HSAN marginally outperforms it. Notably, models like Decision Tree and KNN are less effective in minimizing classification errors, hence it does not guarantee reliability for malware detection in complex data sets.

# Conclusion

This research presents a novel AI-based Android malware scanner built on a HSAN, combining deep sequential learning with attention mechanisms and a hybrid XGBoost classification stage. By leveraging static features, permissions and intent, the model achieves robust detection performance while maintaining low computational overhead. Through the integration of Bidirectional GRUs, a custom attention layer, and mixed precision training, the SNN component effectively learns complex patterns in app behavior. The addition of XGBoost enhances interpretability and classification performance, outperforming traditional ML methods and even the standalone SNN. The proposed system demonstrated superior accuracy (94.35%) and balanced precision-recall metrics, surpassing existing classifiers across multiple benchmarks. Its lightweight architecture and deployment readiness via TensorFlow Lite position as a practical and scalable solution for mobile malware detection. Overall, HSAN contributes a novel, hybrid, and deployable approach to Android malware detection, bridging the gap between academic innovation and real-world security needs.

# References

1. R. Geuens, “What’s Android’s market share? (updated Jan 2025),” SOAX (2025). Retrieved April 10, 2025, from <https://soax.com/research/android-market-share>
2. Y. Chen, Z. Ding, and D. Wagner, “Continuous Learning for Android Malware Detection,” in *Proceedings of the 32nd USENIX Security Symposium (SEC ’23)* (USENIX Association, USA, 2023), pp. 1127–1144.
3. E. Mbunge, B. Muchemwa, J. Batani, and N. Mbuyisa, “A review of deep learning models to detect malware in Android applications,” *Computer Security* **1**, 100014 (2023).
4. H. Alkahtani and T. H. H. Aldhyani, “Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices,” *Sensors* **22**(6), 2268 (2022).
5. D. Gibert, C. Mateu, and J. Planes, “HYDRA: A multimodal deep learning framework for malware classification,” *Computers & Security* **95**, 101873 (2020).
6. O. N. Elayan and A. M. Mustafa, “Android Malware Detection Using Deep Learning,” *Procedia Computer Science* **184**, 847–852 (2021).
7. H. Bragança, V. Rocha, L. Barcellos, E. Souto, and M. M. Oliveira, “A dataset for Android malware detection research,” *Data in Brief* **51**, 109750 (2023).
8. S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, “Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder,” *Journal of Network and Systems Management* **30**(22), 1–20 (2022).
9. A. Guerra-Manzanares, H. Bahsi, and M. Luckner, “Leveraging the first line of defense: A study on the evolution and usage of Android security permissions for enhanced Android malware detection,” *Journal of Computer Virology and Hacking Techniques* **19**, 65–96 (2022).
10. K. Khariwal, J. Singh, and A. Arora, “IPDroid: Android Malware Detection using Intents and Permissions,” in *Proceedings of the 2020 Fourth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, London, UK (IEEE, 2020), pp. 197–202.
11. M. U. Rashid, S. Qureshi, A. Abid, S. S. Alqahtany, A. Alqazzaz, M. U. Hassan, M. S. Al Reshan, and A. Shaikh, “Hybrid Android Malware Detection and Classification Using Deep Neural Networks,” *International Journal of Computational Intelligence Systems* **18**(52), 1–25 (2025).
12. M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, “DL-Droid: Deep learning-based Android malware detection using real devices,” *Computers & Security* **89**, 101663 (2020).
13. J. Brownlee, “Dropout Regularization in Deep Learning Models with Keras,” *Machine Learning Mastery* (2022).
14. D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv preprint* arXiv:1412.6980 (2014).
15. C. Müller, M. Kornblith, and G. Hinton, “When Does Label Smoothing Help?” *arXiv preprint* arXiv:1906.02629 (2019).
16. P. Mavaie, L. Holder, and M. K. Skinner, “Hybrid deep learning approach to improve classification of low-volume high-dimensional data,” *BMC Bioinformatics* **24**, 419 (2023).
17. Y.H. Tay, S.Y. Ooi, Y.H. Pang, Y.H. Gan, and S.L. Lew, “Ensuring Privacy and Security on Banking Websites in Malaysia: A Cookies Scanner Solution,” *Journal of Informatics and Web Engineering* **2**(2), 153–167 (2023).
18. J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization,” *Journal of Machine Learning Research* **13**, 281–305 (2012).
19. E. Hassan, M. Y. Shams, and S. Elmougy, “The effect of choosing optimizer algorithms to improve computer vision tasks: a comparative study,” *Multimedia Tools and Applications* **81**, 1–17 (2022).
20. I. Nusrat and S.-B. Jang, “A Comparison of Regularization Techniques in Deep Neural Networks,” *Symmetry* **10**, 648 (2018).
21. N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research* **15**, 1929–1958 (2014).