International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3740-3743
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0491

Android-Malware Detection Using LLM

Prof. Namrata D. Ghuse[1], Pratiksha D. Bhusare[2]
[1, 2]Department of Computer Engineering, Savitribai Phule Pune University, Pune, India
Emails: *namratagioe@bkc.met.edu[1], pratikshabhusare05@gmail.com[2]*

Abstract
The Android operating system dominates the mobile market, making it a prime target for malicious software (malware). For years, security researchers have relied on machine learning and deep learning models to detect these threats. While effective, these traditional methods often act as "black boxes," providing a "malicious" or "benign" label with little to no explanation. This lack of interpretability makes it difficult to analyze new threats or trust the model's decisions. The recent rise of Large Language Models (LLMs) presents a new paradigm. This paper reviews the emerging field of LLM-based Android malware detection. We use the LLM-MalDetect framework, which achieves 98.97% accuracy by fine-tuning a model on string-based features, as a baseline to demonstrate the power of LLMs as direct classifiers. We then survey the different ways researchers are applying these models: not just for classification, but as semantic feature extractors (AppPoet), synthetic data generators (Syn-detect), and, most importantly, as explainable analysis tools (MalLoc, TraceRAG). We find that the true innovation of LLMs lies not just in their high accuracy but in their unique ability to provide human-readable, code-grounded explanations for why an application is malicious, shifting the field from simple detection to true behavioral analysis.

Keywords: *Android, Malware Detection, Large Language Models (LLMs), Explainable AI (XAI), Cybersecurity, RAG, Static Analysis*

1. Introduction
The Android operating system is the most popular mobile platform in the world, powering billions of devices. Unfortunately, its open-source nature and massive user base make it an attractive target for cybercriminals. Android malware poses a severe threat, enabling everything from data theft and financial fraud to unauthorized device control. For the past decade, the primary defense against this has been machine learning (ML) and deep learning (DL) models. These "traditional" learning-based methods, such as the well-known Drebin, analyze an application's features (e.g., requested permissions, API calls) to classify it as malicious or benign. While these models have achieved high accuracy on known datasets, they suffer from two major drawbacks:

- Poor Generalization: They struggle to detect new, "zero-day" threats or malware that uses obfuscation to hide its true intent[1].
- Lack of Interpretability: Most models are "black boxes". They can tell you an app is dangerous, but they cannot tell you why. They fail to provide human-readable insights, which is a significant problem for security analysts who need to understand the threat.

The introduction of powerful Large Language Models (LLMs) like BERT, GPT, and Mistral has opened a new frontier. These models are trained on vast amounts of text and code, giving them a deep understanding of programming logic and natural language. A prime example of this new approach is LLM-MalDetect, the base paper for this review. This framework achieves a state-of-the-art accuracy of 98.97% on the Drebin dataset. It works by fine-tuning a Mistral 7B model using Low-Rank Adaptation (LoRA). Its key insight is to feed the LLM a rich set of string-based features, including permissions, API calls, and even text strings found inside the app's code. This high accuracy proves that LLMs are not just a novelty but a powerful and viable tool for direct malware classification. However, high accuracy is only part of the story. This review paper explores the different ways researchers are now using LLMs to revolutionize Android malware detection[2]. We categorize these new methods based on their core strategy: using LLMs as feature extractors, as data generators, as explainable analyzers, and as practical

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3740-3743
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0491

tools for handling massive codebases[3].

2. Evolving Methodologies In Llm-Based Detection

Researchers are moving beyond simple classification and are creatively applying LLMs to solve multiple problems in the malware analysis pipeline. We have grouped these approaches into three main categories[4].

2.1. LLMs as Semantic Feature Extractors

Instead of just feeding a list of features (like READ_CONTACTS and SEND_SMS) into a classifier, this approach uses an LLM to first understand the relationship between those features. The AppPoet framework is a perfect example. It collects "multi-view" features from an app, including permissions, APIs, and URLs. It then prompts an LLM to act as a security analyst and write a "behavioral summary" based on those features. For instance, the LLM can infer that an app requesting both contact permissions and internet access might be trying to steal and upload a user's address book. This natural-language summary, which captures the semantic meaning of the features, is then fed to a standard DNN classifier for the final decision. Here, the LLM acts as an intelligent pre-processor, enriching the features before classification.

2.2.LLMs as Synthetic Data Generators

A constant problem in malware security is a lack of training data for new threats. If a model has never seen a particular type of attack, it can't detect it. Two papers in our review use LLMs to solve this by generating synthetic malware data for training.

- Synthetic Network Traffic (Syn-detect): The Syn-detect model focuses on network-based malware. It uses GPT-2, an older but effective LLM, to learn the patterns of malicious TCP network traffic. It then generates new, realistic, synthetic TCP packet data. This synthetic data is added to the real data, creating a much larger and more diverse dataset. A separate BERT model is then trained on this augmented dataset to become a more robust network-traffic malware classifier.
- Synthetic Feature Records: A similar study by Rollinson and Polatidis uses a fine-tuned GPT-4.1-mini to generate structured malware records. Instead of raw traffic, it generates feature-based descriptions of malware families like BankBot. Their findings show that training a classifier on 100% synthetic data isn't effective, but using synthetic data to augment a smaller real dataset significantly boosts performance.

2.3. Llms As Direct Classifiers

This is the approach taken by our base paper, LLM-MalDetect. Here, the LLM is not a pre-processor or a data generator; it is the classifier. By fine-tuning the Mistral 7B model on a curated set of permissions, API calls, and string features, the model learns to directly output a "malicious" or "benign" label. The 98.97% accuracy achieved by LLM-MalDetect on the Drebin dataset and 94.51% on the more recent AndroZoo-2024 dataset demonstrates that this is a very powerful and direct method for malware detection[5].

3. The Explainability Frontier: Beyond "What" To "Why" And "Where"

The most significant shift enabled by LLMs is the move from simple detection to in-depth, explainable analysis. For a security analyst, a "yes/no" answer is not enough. They need to know why it's malicious, where the bad code is, and how it works[6].

3.1. The "Why": Identifying The Interpretability Gap

A study by Li et al. explicitly investigates this problem by comparing traditional models to ChatGPT. They found that traditional "decision-centric" models like Drebin, while accurate, are completely opaque. In contrast, when given the same features, ChatGPT provides comprehensive analysis reports, enhancing interpretability. The study also surveyed experienced developers, who overwhelmingly stated they preferred ChatGPT's detailed reports, finding them more useful and trustworthy, even if the LLM couldn't make a final "yes/no" decision. This proves the demand for explainability[7].

3.2. The "Where": Malicious Payload Localization

The MalLoc framework is designed to find the exact "where". It uses an LLM to perform a two-phase analysis of an app's Smali code (the low-level assembly code for Android)[8].

Phase 1: The LLM identifies which Smali classes are likely to contain malicious code.

Phase 2: The LLM then "drills down" into those

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3740-3743
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0491

classes to pinpoint the specific methods responsible for the malicious behavior[9].

This approach is revolutionary because it moves beyond a simple app-level label and provides the analyst with the exact location of the threat, saving hours of manual reverse-engineering[10].

3.3. The "How": Tracing Behavior With RAG

The TraceRAG framework provides the "how" by building a system an analyst can "talk" to. It uses a Retrieval-Augmented Generation (RAG) pipeline. First, it scans all the Java source code of an app, uses an LLM (LLM-Describer) to summarize every method, and stores these summaries in a vector database. A security analyst can then ask a natural language question, like:

> "Does this application access or collect sensitive user data?"

The RAG system retrieves the most relevant code snippets from the database. These snippets are then given to "LLM-Analyzer," which "reads" the code and traces its logic, following function calls to see how it works. Finally, it generates a full report for the analyst, grounded in the actual code, explaining exactly how the app steals data[11].

4. Overcoming Practical Challenges

While powerful, LLMs are not a silver bullet. They face two major practical challenges when analyzing real-world Android apps: context limits and a lack of evaluation standards[12].

4.1. Context Window Problem (LAMD)

An Android app can contain millions of lines of code. This is far too large to fit into any current LLM's context window. The LAMD framework provides an elegant solution to this problem. Instead of feeding the entire app to the LLM, it first identifies "suspicious APIs" (e.g., sendTextMessage). It then uses a technique called backward program slicing to automatically extract only the code that is relevant to that suspicious API call. This creates a small, focused "slice" of code that contains the critical logic. This slice is small enough to fit in the LLM's context window but contains all the necessary information for the LLM to make an accurate, context-driven decision[13].

4.2. The Evaluation Problem (Cama)

How do we even know if an LLM's analysis is correct? There is no ground-truth dataset for "malicious function locations". The CAMA framework addresses this by proposing a new way to benchmark and evaluate Code LLMs for this task. It provides a structured output format (summaries, refined function names, maliciousness scores) and proposes new evaluation metrics like Fidelity (does the LLM's score correctly identify malicious functions?) and Semantic Relevance (do the LLM's summaries match a human's understanding?). CAMA gives researchers a standardized way to measure and compare their LLM-based analysis tools[16].

5. Challenges And Future Directions

This highlights a clear trend, but significant challenges remain.

- LLM Hallucinations: LLMs can "make up" facts. In security, this is unacceptable. Methods like LAMD's "factual consistency verification" and TraceRAG's RAG-based grounding are crucial, as they force the LLM to base its reasoning on actual code[15].
- Code Obfuscation: Malicious authors intentionally scramble their code (obfuscation) to confuse analysts. While TraceRAG's "LLM-Cleanser" attempts to de-obfuscate code, this remains a major arms race. Future LLMs will need to be explicitly trained to see through these techniques.
- Data Scarcity: While synthetic data generation (Syn-detect, Rollinson) helps, the field still needs large, high-quality, and labeled datasets of malicious code snippets, not just full apps[14].

Conclusion

Large Language Models are fundamentally changing the field of Android malware detection. As demonstrated by our base paper, LLM-MalDetect, LLM-based classifiers can achieve state-of-the-art accuracy, surpassing many traditional models. However, this review finds that the true power of LLMs is not just in their accuracy. The real revolution is in explainability. Frameworks like AppPoet use LLMs to translate cryptic feature lists into human-readable behavioral summaries. MalLoc uses them to pinpoint the exact location of malicious code, and TraceRAG allows analysts to trace malicious logic by asking simple questions. Practical tools like LAMD are solving the context-window problem by applying intelligent code slicing. We are moving from

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3740-3743
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0491

"decision-centric" detection, which simply flags a file, to "analysis-centric" detection, which provides a full, evidence-based report. While challenges like obfuscation and hallucinations remain, LLMs are paving the way for a new generation of smart, explainable security tools that act as true partners for security analysts

References

[1]. R. Feng, H. Chen, S. Wang, M. M. Karim, and Q. Jiang, "LLM-MalDetect: A Large Language Model-Based Method for Android Malware Detection," IEEE Access, vol. 13, pp. 81347–81364, 2025.

[2]. M. Naseer et al., "Obfuscated Malware Detection and Classification in Network Traffic Leveraging Hybrid Large Language Models and Synthetic Data," Sensors, vol. 25, no. 202, 2025.

[3]. T. Sun et al., "MalLoc: Toward Fine-grained Android Malicious Payload Localization via LLMs," arXiv:2508.17856v1 [cs.CR], 25 Aug 2025.

[4]. N. Rollinson and N. Polatidis, "LLM-Generated Samples for Android Malware Detection," arXiv, 2025.

[5]. W. Zhao, J. Wu, and Z. Meng, "AppPoet: Large Language Model based Android malware detection via multi-view prompt engineering," arXiv:2404.18816v3 [cs.CR], 22 Oct 2024.

[6]. Y. He et al., "On Benchmarking Code LLMs for Android Malware Analysis," arXiv:2504.00694v2 [cs.CR], 23 Apr 2025.

[7]. G. Zhang et al., "TraceRAG: A LLM-Based Framework for Explainable Android Malware Detection and Behavior Analysis," arXiv:2509.08865v1 [cs.SE], 10 Sep 2025.

[8]. Y. Li, S. Fang, T. Zhang, and H. Cai, "Enhancing Android Malware Detection: The Influence of ChatGPT on Decision-centric Task," arXiv:2410.04352v2 [cs.CR], 26 Feb 2025.

[9]. X. Qian, X. Zheng, Y. He, S. Yang, and L. Cavallaro, "LAMD: Context-driven Android Malware Detection and Classification with LLMs," arXiv:2502.13055, 2025.

[10]. R. Mariammal, B. Haribabu, K. Venkatesh, and K. R. Reddy, "Android Mobile Malware Detection using Machine Learning Techniques," International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), vol. 3, no. 8, Apr. 2023.

[11]. A. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket,"NDSS Symposium, 2014.

[12]. W. Wang, M. Zhao, J. Wang, and Z. Wang, "Malware Traffic Classification Using Convolutional Neural Network for Representation Learning," IEEE International Conference on Information Networking, 2017.

[13]. D. Li, L. Wang, and H. Jin, "Deep Learning Based Android Malware Detection Using API Call Sequences," Journal of Information Security and Applications, vol. 52, 2020.

[14]. K. Kim, J. Park, and H. Kim, "Android Malware Detection Using Graph Neural Networks Based on API Call Graphs," IEEE Access, vol. 9, pp. 123456–123467, 2021.

[15]. T. Brown et al., "Language Models are Few-Shot Learners," Advances in Neural Information Processing Systems (NeurIPS), 2020.

[16]. J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," NAACL-HLT, 2019.