RAML: Toward Retrieval-Augmented Localization of Malicious Payloads in Android Apps

Tiezhu Sun[1], Marco Alecci[1], Yewei Song[1], Xunzhu Tang[1], Kisub Kim[2]
Jordan Samhi[1], Tegawendé F. Bissyandé[1], Jacques Klein[1]
[1]University of Luxembourg, Luxembourg
[2]DGIST, Korea
{firstname.lastname}@uni.lu, kisub.kim@dgist.ac.kr

***Abstract*—Android malware detection and family classification have been extensively studied, yet localizing the exact malicious payloads within a detected sample remains a challenging and labor-intensive task. We propose *RAML*, a novel Retrieval-Augmented Malicious payload Localization pipeline inspired by retrieval-augmented generation (RAG), which leverages large language models (LLMs) to bridge high-level behavior descriptions and low-level Smali code. *RAML* generates class-level descriptions from Smali code, embeds them into a vector database, and performs semantic retrieval via similarity search. Matched candidates are re-ranked with LLM assistance, followed by method-level LLM analysis to precisely identify malicious methods and provide insightful role explanations. Preliminary results show that *RAML* effectively localizes corresponding malicious payloads based on behavioral descriptions, narrows the analysis scope, and reduces manual effort—offering a promising direction for automated malware forensics.**

***Index Terms*—Android Malware Analysis, Malicious Payload Localization, Retrieval-Augmented Generation**

I. Introduction

Android powers billions of devices and supports a vast app ecosystem [1], but its openness also creates a significant attack surface. Android malware continues to evolve, employing sophisticated evasive techniques to evade traditional static and dynamic analyses [2], [3]. Consequently, accurately localizing malicious payloads within apps remains technically challenging, requiring novel approaches to reason about code semantics under adversarial conditions.

Despite progress in malware detection [4]–[9] and family classification [10]–[14], existing techniques rarely provide fine-grained, code-level insights. Most models lack explicit localization capabilities and operate as black boxes, reducing interpretability and hindering analysts' ability to extract meaningful features and validate predictions. This limitation restricts their effectiveness in practical, security-critical scenarios. Prior works [15], [16] have explored class-level localization of malicious payloads, but they offer limited precision in pinpointing specific malicious code and fail to provide detailed descriptions of how identified payloads behave. Recently, large language models (LLMs) have been applied to Android malware analysis, but existing approaches primarily focus on detection rather than precise payload localization [17], [18].

To address these limitations, we introduce *RAML*, a novel **R**etrieval-**A**ugmented **M**alicious payload **L**ocalization approach. Inspired by recent advances in retrieval-augmented generation (RAG) [19], [20], *RAML* leverages LLMs to precisely localize malicious payloads at the method level and provide meaningful explanations. It is designed to assist human experts by bridging the gap between family classification and actionable malware defense.

RAML leverages malware family knowledge to efficiently localize malicious payloads, supporting human analysts by reducing the need for manual analysis. Given a malware sample and its predicted family label, *RAML* retrieves behavior queries (see Section III for an example query) from a predefined family-behavior lookup table to guide the localization process. It then applies a retrieval-augmented localization approach: it first uses an LLM to generate natural-language descriptions of Smali classes (i.e., a low-level representation of Android bytecode), which are embedded and indexed in a vector database for semantic similarity search against each behavior query. Matched candidates undergo LLM-based re-ranking, followed by method-level LLM analysis to pinpoint specific malicious methods and explain their functionalities. This pipeline enables precise, interpretable localization guided by malware family insights. To the best of our knowledge, this is the first work to apply the RAG paradigm to bridge high-level behavioral descriptions and low-level Smali code semantics for localizing malicious payloads in Android malware.

A major challenge in evaluating malicious payload localization is the lack of ground-truth datasets with fine-grained annotations. To address this, we developed *LocApp*, a custom Android app that implements several common malicious behaviors, such as privacy theft and aggressive advertising, with precise ground-truth annotations for controlled, quantitative evaluation. We further assess *RAML* on a real-world malware sample from MalRadar [21], manually validating its predictions. Experimental results demonstrate the effectiveness of *RAML* in delivering accurate and explainable payload localization. We believe this work can inspire future directions in dynamic modeling of localized payloads, interpretable malware detection, and behavior-driven mitigation strategies.

The contributions of this work are summarized as follows:

- We propose a novel retrieval-augmented framework that bridges high-level family behavior knowledge with low-level Smali code to localize malicious payloads. Unlike

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00351

prior works, *RAML* provides method-level localization and human-readable behavioral explanations.
- We develop a demo app and analyze a real-world malware sample, enabling preliminary method-level evaluation and demonstrating *RAML*'s potential to enhance both precision and interpretability in malware analysis.
- We publicly release the dataset and source code of *RAML* at: https://github.com/Trustworthy-Software/RAML

II. Background

Android Malware Analysis. Learning-based approaches have greatly advanced Android malware detection and family classification over the past decade [4], [6], [22]–[24], with recent efforts exploring large language models (LLMs) [17], [18]. Despite this progress in foundational stages, these approaches alone are insufficient for enabling effective malware defense. A critical, yet under-explored next stage is *fine-grained malicious payload localization*, i.e., identifying and interpreting specific methods or code segments responsible for malicious behaviors. In practice, such payloads may appear as standalone methods or be interleaved within benign ones, which complicates precise identification and can obscure small but critical malicious instructions. Without this capability, security analysts lack the necessary insight to fully understand, explain, and defend against malware attacks. Our work addresses this gap: the core novelty of *RAML* lies in providing precise localization and behavioral explanations, thereby bridging foundational detection/classification stages and actionable malware defense.

Smali Code. Android apps are typically written in Java or Kotlin and compiled into Dalvik Executable (DEX) bytecode [25], stored in `.dex` files within APKs. Since APKs rarely include source code, high-level inspection is often infeasible. Instead, tools like `ApkTool` [26] decompile DEX into Smali, a low-level but human-readable format. Smali serves as a practical intermediate for behavior analysis when source code is unavailable. Recent work shows that LLMs can effectively interpret Smali code [27], which further motivates our Smali-based approach.

III. Approach

We begin by introducing a baseline, followed by our proposed *RAML* approach.

Baseline. We implement a baseline approach to illustrate the limitations of direct LLM prompting without retrieval. It applies an LLM to scan every Smali class in an app to identify and explain malicious behaviors from a predefined list. This list includes 12 distinct behavior types curated from MalRadar [21], a high-quality benchmark of real-world Android malware with manually verified family labels. These 12 behaviors span all 148 malware families in the dataset, as each family is associated with specific behaviors from this list. The behavior list and prompt template used in the baseline are shown in Figure 1.

As we show later in Section IV-B, the approach above fails to accurately localize malicious behaviors. Our analysis attributes this to the complexity of the task and the lack of sufficient contextual guidance. In particular, the baseline prompt implicitly asks the LLM to perform three tasks at once: *Task* ❶ decide whether the input Smali class is malicious or benign; *Task* ❷ identify which behaviors are present if malicious; and *Task* ❸ localize the specific methods responsible. These challenges highlight the need for a more structured and guided approach, which we introduce hereafter.

Context:
You are an expert in Android malware analysis. Analyze the following Smali class and determine if it implements any malicious behaviors.

Input – Smali Class:
`{class_content}`

Possible Malicious Behaviors:
1. Privacy Stealing; 2. SMS/CALL Abuse; 3. Remote Control;
4. Bank/Financial Stealing; 5. Ransom; 6. Accessibility Abuse;
7. Privilege Escalation; 8. Stealthy Download; 9. Aggressive Advertising; 10. Miner; 11. Tricky Behavior; 12. Premium Service Abuse.

Instruction:
Use the following format:

```
IS_MALICIOUS: <yes or no>
CONFIDENCE: <confidence score 0-100>
EXPLANATION: <detailed explanation>
BEHAVIOR: <comma-separated behaviors>

METHOD: <method signature>
ROLE: <role description>
METHOD: <...>
ROLE: <...>
```

Fig. 1. The prompt template of the baseline approach.

RAML. To address the limitations of the baseline, as illustrated in Figure 2, *RAML* decomposes these tasks into a multi-stage retrieval process.

Smali Class Representation. To facilitate initial analysis, *RAML* first processes each Smali file by removing debug information, comments, and metadata. It then extracts key details such as class names, method definitions, and Android permissions. An LLM is used to generate natural-language descriptions for each class, summarizing its functionality and highlighting potential indicators of malicious behavior. → *This addresses Task* ❶ *(malicious vs. benign classification) by providing interpretable summaries without premature judgment.*

Vector Database Construction. Each class description is embedded using an embedding model and stored in a Chroma [28] vector database alongside metadata. This structure enables efficient semantic search across all classes, significantly narrowing the search space for downstream analysis. → *RAML avoids repeatedly scanning the entire app when localizing* ***multiple*** *behaviors—since the vectorized representation can be reused across multiple distinct behavior queries.*

Retrieval-Augmented Localization. For each malware sample, behavior queries are obtained from a predefined family-behavior lookup table based on its family label. These queries are carefully crafted to reflect the 12 behavior categories illustrated in Figure 1. Below is an example query:

> **SMS/CALL Abuse**: - Methods that manipulate SMS and phone call functionality:
> (1) - Sending SMS messages without user consent
> (2) - Intercepting/blocking incoming SMS (especially 2FA messages)
> (3) - Deleting SMS messages (to hide evidence)
> (4) - Making calls without user awareness
> (5) - Monitoring call logs
> Look for: SMS manager operations, broadcast receivers for SMS/calls, telephony API usage, SMS deletion commands.

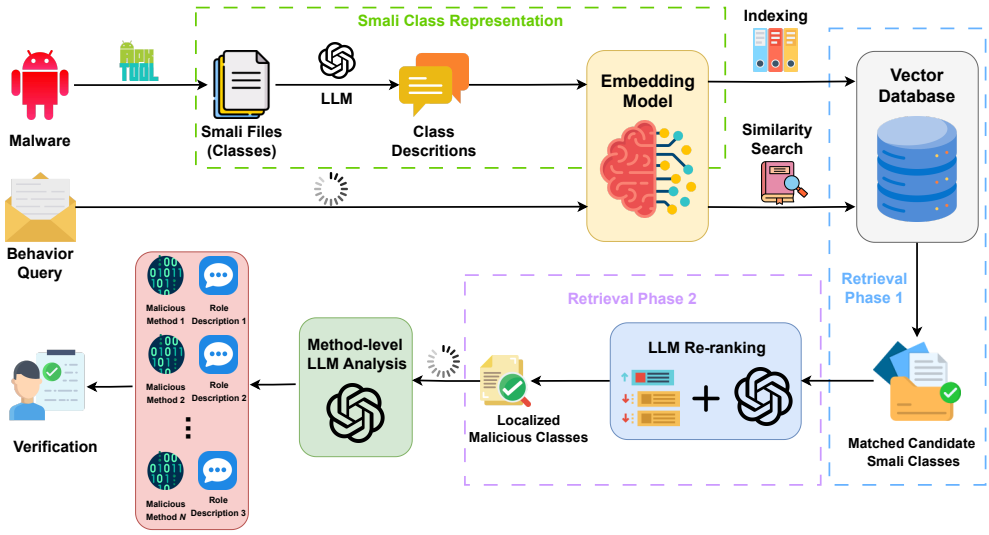

Fig. 2. The overview of *RAML* Pipeline.

Given one such query, *RAML* performs two-phase retrieval:

- *Phase 1 – Semantic Search:* The behavior query is vectorized using the embedding model and compared against the vector database of class descriptions. This identifies candidate classes with semantically similar functionality based on cosine similarity.
- *Phase 2 – LLM Re-ranking:* An LLM re-ranks the top candidates by analyzing their relevance to the specific behavior, assigning scores and generating explanations.

→ *This addresses Task ❷ (behavior identification) by narrowing and validating candidate classes per behavior.*

Method-Level Analysis. For each behavior-relevant class, *RAML* performs method-level analysis, where an LLM reviews each method in context and identifies those involved in implementing the target behavior. It also provides role-specific explanations and confidence scores. → *This addresses Task ❸ (method localization) with fine-grained, interpretable outputs.*

Through structured decomposition, *RAML* overcomes the baseline's ambiguity and overload, enabling precise and explainable malicious payload localization guided by high-level behavior descriptions. All prompt templates and behavior queries used in our approach are available in the replication package.

IV. Experiments

A. Apps Under Analysis

Due to the lack of fine-grained ground-truth datasets for malicious payload localization, we evaluate *RAML* on two representative Android apps: ❶ *LocApp*, a controlled demo app, and ❷ a real-world malware sample from the MalRadar benchmark.

1) LocApp: The developer code of *LocApp* contains 51 Smali classes and 165 methods, implementing three controlled malicious behaviors: *Privacy Stealing* (extracting and transmitting contact data), *Aggressive Advertising* (fake click generation), and *Tricky Behavior* (app hiding and icon manipulation). The app provides precise method-level ground truth to facilitate reliable, quantitative evaluation. All external functionalities are simulated for safety and ethical compliance.

2) RuMMs App: We selected a samll representative malware sample from MalRadar's largest family, RuMMs, annotated with five malicious behaviors: *Privacy Stealing*, *SMS/CALL Abuse*, *Remote Control*, *Bank/Financial Stealing*, and *Tricky Behavior*. The APK includes 21 classes and 66 methods, requests multiple sensitive permissions (e.g., `READ_CONTACTS`, `SEND_SMS`), and masquerades as an MMS/SMS messenger. Manual verification of the generated localization outputs was performed to assess *RAML*'s accuracy and interpretability, highlighting its robustness in analyzing real-world malware.

B. Results

TABLE I
Comparison of Baseline and *RAML* on *LocApp*.

Approach	C-Prec	C-Rec	C-F1	M-Prec	M-Rec	M-F1
Baseline	0.67	0.67	0.67	0.70	0.58	0.64
RAML	0.75	1.00	0.86	0.71	1.00	0.83

1) LocApp: For evaluation, we used OpenAI's `text-embedding-ada-002` for embeddings and `GPT-4.1` as the LLM [29]. Table I presents the results, where **C-** and **M-** denote class- and method-level metrics: Precision, Recall and F1 Score. True positives (TP) indicate a correctly identified malicious class or method with the correct behavior label. False positives (FP) refer to benign components incorrectly flagged or behaviors mislabeled, while false negatives (FN) are missed malicious components.

The baseline performs poorly across almost all metrics. In contrast, *RAML* substantially improves performance by

Class: `Lorg/regular/random/b;` (obfuscated)
Behavior: *Privacy Stealing*
Method: `.method public static f(Landroid/content/Context;)Ljava/util/ArrayList;`
Confidence Score: 100
Role: Reads the device's contact list (names and phone numbers) from the contacts content provider, which is a typical data theft behavior.
Method: `.method public static b(Landroid/content/Context;ILjava/lang/String;)V`
Confidence Score: 95
Role: Collects device identifiers (android_id), packages arbitrary data (potentially exfiltrated), encodes it, and forwards it for transmission—indicative of data exfiltration.
Method: .....

Fig. 3. An example prediction by *RAML*, illustrating class-level behavior and method-level role explanations.

integrating the same LLM within our retrieval-augmented framework. As shown in Table I, *RAML* achieves perfect recall—capturing all malicious classes and methods—alongside improved precision and F1. This is especially valuable for manual verification, ensuring no critical behaviors are missed. *RAML* flags 4 classes and 19 methods as malicious, out of 165 total methods in the app—reducing the analyst's review scope by 88.5%. This underscores its effectiveness in supporting accurate and efficient malicious payload localization.

2) RuMMs App: We further evaluate *RAML* on the selected real-world malware sample from the RuMMs family. While MalRadar does not provide method-level annotations, our manual analysis—cross-checking API calls, control flow, and semantics against known malicious behaviors—indicates that *RAML* can reliably identify the five annotated malicious behaviors. Specifically, *RAML* accurately localizes 7 malicious classes and 25 malicious methods among the app's total of 21 classes and 66 methods, achieving 100% precision at both class and method levels. Key localized behaviors include contact data exfiltration (*Privacy Stealing*), unauthorized SMS sending (*SMS/CALL Abuse*), persistent background services and remote interaction (*Remote Control*), and auto-clicking consent dialogs (*Tricky Behavior*). An example prediction is shown in Figure 3.

These results highlight *RAML*'s capability for precise and interpretable localization, even in obfuscated, real-world malware scenarios. Despite the absence of formal ground truth, its predictions closely align with expert analysis, demonstrating strong potential to support practical malware inspections.

V. Discussion

We reflect on the key insights from our evaluation and outline future directions for advancing Android malware analysis.

Insights. Rather than aiming for full automation, our goal is to assist human analysts by narrowing the search space and reducing the manual effort to identify and interpret malicious behaviors. A key feature of *RAML* is the generation of natural-language role explanations for each analyzed method, clarifying how individual methods contribute to the overall malicious behavior of the app. One of the critical design choices in *RAML* is its *two-level localization approach*: first at the class level, then refined to the method level. This granularity is essential because many malicious methods, when analyzed in isolation, may appear benign or unrelated to any malicious activity.

For example, in the *RuMMs App* we analyzed, we identified a method named `org.regular.Random.Bee.onHandleIntent()`, which intercepts intents related to receiving SMS messages. This behavior is not inherently malicious—it is common in legitimate applications. However, in the context of a high-level behavior such as *"SMS/CALL Abuse"*, the method becomes highly significant. Here, *RAML*'s `role_explanation` proves especially valuable. In this case, the explanation describes the method as: *"Entry point for the service; routes incoming intents to the appropriate handler (either SMS interception or command execution)"*. Such contextualization enables analysts to understand the method's role within a larger malicious workflow. By providing step-by-step insight into how each method contributes to malicious payloads, *RAML* supports the reconstruction of complex behaviors and aids in forensic analysis, even when the malicious logic is spread across multiple, superficially innocuous methods.

While our evaluation demonstrates promising precision and explainability, further research is needed to validate whether such performance generalizes to more complex, diverse, or heavily obfuscated real-world malware scenarios.

Research Outlook. We envision *RAML*'s retrieval-augmented localization as a foundation for downstream Android security tasks such as dynamic behavior modeling, explainable malware detection, and behavior-specific mitigation. This work can foster more interpretable, context-aware analysis, strengthening the synergy between software engineering and security while integrating LLM-based reasoning into practical workflows. Beyond malicious behavior, the pipeline could also aid Android-specific tasks like detecting policy violations, compliance issues, or misuse of sensitive data.

VI. Conclusion

We presented *RAML*, a novel retrieval-augmented framework for fine-grained localization of malicious payloads in Android apps. Unlike traditional work primarily focused on coarse-grained detection or family classification, *RAML* offers deeper insights by localizing malicious logic precisely at the class and method levels, accompanied by clear, functional explanations.

Future Work. We identify several promising research directions. First, we plan to broaden our evaluation by including additional LLMs (particularly open-source models) and a wider variety of real-world malware samples to better assess generalizability across diverse behaviors and obfuscation techniques. Second, we aim to conduct ablation studies to thoroughly understand each component's impact. Finally, we will explore strategies to enhance *RAML*'s efficiency and scalability, especially for analyzing large, complex Android applications.

Acknowledgment

This research was funded in whole or in part by the Luxembourg National Research Fund (FNR), grant references 16344458 (REPROCESS) and 18154263 (UNLOCK).

References

[1] A. Turner, "How many android users are there? global and us statistics (2025)," https://www.bankmycell.com/blog/how-many-android-users-are-there, 2025, accessed: June 2025.

[2] P. Faruki, R. Bhan, V. Jain, S. Bhatia, N. El Madhoun, and R. Pamula, "A survey and evaluation of android-based malware evasion techniques and detection frameworks," *Information*, vol. 14, no. 7, p. 374, 2023.

[3] A. Ruggia, D. Nisi, S. Dambra, A. Merlo, D. Balzarotti, and S. Aonzo, "Unmasking the veiled: A comprehensive analysis of android evasive malware," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 383–398.

[4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, no. 1, 2014, pp. 23–26.

[5] Y. Wu, X. Li, D. Zou, W. Yang, X. Zhang, and H. Jin, "Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 139–150.

[6] N. Daoudi, J. Samhi, A. K. Kabore, K. Allix, T. F. Bissyandé, and J. Klein, "Dexray: a simple, yet effective deep learning approach to android malware detection based on image representation of bytecode," in *Deployable Machine Learning for Security Defense: Second International Workshop, MLHat 2021, Virtual Event, August 15, 2021, Proceedings 2*. Springer, 2021, pp. 81–106.

[7] T. Sun, N. Daoudi, K. Allix, and T. F. Bissyandé, "Android malware detection: looking beyond dalvik bytecode," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 2021, pp. 34–39.

[8] T. Sun, N. Daoudi, K. Kim, K. Allix, T. F. Bissyandé, and J. Klein, "Detectbert: Towards full app-level representation learning to detect android malware," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2024, pp. 420–426.

[9] T. Sun, N. Daoudi, K. Allix, J. Samhi, K. Kim, X. Zhou, A. K. Kabore, D. Kim, D. Lo, T. F. Bissyandé *et al.*, "Android malware detection based on novel representations of apps," in *Malware: Handbook of Prevention and Detection*. Springer, 2024, pp. 197–212.

[10] F. Alswaina and K. Elleithy, "Android malware family classification and analysis: Current status and future directions," *Electronics*, vol. 9, no. 6, p. 942, 2020.

[11] C. Ding, N. Luktarhan, B. Lu, and W. Zhang, "A hybrid analysis-based approach to android malware family classification," *Entropy*, vol. 23, no. 8, p. 1009, 2021.

[12] H.-I. Kim, M. Kang, S.-J. Cho, and S.-I. Choi, "Efficient deep learning network with multi-streams for android malware family classification," *IEEE Access*, vol. 10, pp. 5518–5532, 2021.

[13] S. Freitas, R. Duggal, and D. H. Chau, "Malnet: A large-scale image database of malicious software," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 3948–3952.

[14] T. Sun, N. Daoudi, W. Pian, K. Kim, K. Allix, T. F. Bissyandé, and J. Klein, "Temporal-incremental learning for android malware detection," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 4, pp. 1–30, 2025.

[15] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "A multi-view context-aware approach to android malware detection and malicious code localization," *Empirical Software Engineering*, vol. 23, pp. 1222–1274, 2018.

[16] T. Sun, K. Allix, K. Kim, X. Zhou, D. Kim, D. Lo, T. F. Bissyandé, and J. Klein, "Dexbert: Effective, task-agnostic and fine-grained representation learning of android bytecode," *IEEE Transactions on Software Engineering*, vol. 49, no. 10, pp. 4691–4706, 2023.

[17] X. Qian, X. Zheng, Y. He, S. Yang, and L. Cavallaro, "Lamd: Context-driven android malware detection and classification with llms," *arXiv preprint arXiv:2502.13055*, 2025.

[18] W. Zhao, J. Wu, and Z. Meng, "Apppoet: Large language model based android malware detection via multi-view prompt engineering," *Expert Systems with Applications*, vol. 262, p. 125546, 2025.

[19] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[20] J. Chen, H. Lin, X. Han, and L. Sun, "Benchmarking large language models in retrieval-augmented generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 754–17 762.

[21] L. Wang, H. Wang, R. He, R. Tao, G. Meng, X. Luo, and X. Liu, "Malradar: Demystifying android malware in the new era," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 2, pp. 1–27, 2022.

[22] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," *arXiv preprint arXiv:1612.04433*, 2016.

[23] J. Liu, J. Zeng, F. Pierazzi, L. Cavallaro, and Z. Liang, "Unraveling the key of machine learning solutions for android malware detection," *arXiv preprint arXiv:2402.02953*, 2024.

[24] M. Alecci, J. Samhi, L. Li, T. F. Bissyande, and J. Klein, " Improving Logic Bomb Identification in Android Apps via Context-Aware Anomaly Detection ," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 05, pp. 4735–4753, Sep. 2024. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TDSC.2024.3358979

[25] "Dalvik executable format," https://source.android.com/docs/core/runtime/dex-format, accessed: June 2025.

[26] "Apktool," https://apktool.org/, accessed: June 2025.

[27] M. Alecci, N. Sannier, M. Ceci, S. Abualhaija, J. Samhi, D. Bianculli, T. F. d. A. BISSYANDE, and J. Klein, "Toward llm-driven gdpr compliance checking for android apps," in *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion'25)*, 2025.

[28] Chroma, "Chroma: The ai-native open-source embedding database," https://www.trychroma.com, 2023, accessed: July 2025.

[29] OpenAI, "Gpt-4.1 api," https://openai.com/index/gpt-4-1/, 2025.