Toward LLM-Driven GDPR Compliance Checking for Android Apps

Marco Alecci
University of Luxembourg
Luxembourg, Luxembourg
marco.alecci@uni.lu

Nicolas Sannier
University of Luxembourg
Luxembourg, Luxembourg
nicolas.sannier@uni.lu

Marcello Ceci
University of Luxembourg
Luxembourg, Luxembourg
marcello.ceci@uni.lu

Sallam Abualhaija
University of Luxembourg
Luxembourg, Luxembourg
sallam.abualhaija@uni.lu

Jordan Samhi
University of Luxembourg
Luxembourg, Luxembourg
jordan.samhi@uni.lu

Domenico Bianculli
University of Luxembourg
Luxembourg, Luxembourg
domenico.bianculli@uni.lu

Tegawendé F. Bissyandé
University of Luxembourg
Luxembourg, Luxembourg
tegawende.bissyande@uni.lu

Jacques Klein
University of Luxembourg
Luxembourg, Luxembourg
jacques.klein@uni.lu

Abstract

Android apps extensively collect sensitive personal data from our devices daily. Despite stringent regulations like the European Union's General Data Protection Regulation (GDPR), many applications (apps) fail to comply with these legal requirements. While previous studies have focused on the compliance of privacy policies, checking how these policies are implemented in the actual code has not yet been extensively investigated. Moreover, previous efforts have often been limited in scope.

This paper explores the potential of Large Language Models (LLMs) to address the challenge of verifying privacy regulation compliance in Android apps. Specifically, we address scenarios where source code is unavailable by investigating whether LLM can work with Smali code—a human-readable representation of Android bytecode extracted from APK files. Through this exploratory investigation, we aim to uncover if LLMs can bridge the gap between legal privacy requirements and their technical implementation in mobile apps. Through initial experiments, we assess the feasibility and effectiveness of a straightforward LLM-driven method for identifying compliance issues and provide directions for our future research efforts to improve our approach and perform large-scale experiments.

CCS Concepts

• **Software and its engineering**; • **Security and privacy**;

Keywords

Static Analysis, Android, Data Protection Compliance

1 Introduction

Recent studies have shown that many mobile applications (apps) may collect sensitive personal data excessively or without obtaining proper user consent [1–3]. As a result, to protect users, mobile apps are generally required to adhere to legal frameworks such as the European Union's General Data Protection Regulation (GDPR) [4] on data access, storage, and processing. Apps distributed through official app stores are expected to be accompanied by a privacy policy, i.e., an accessible document that outlines how personal data is handled and how the app complies with GDPR principles, helping users make informed decisions. In this paper, we focus specifically on Android apps which must also comply with these regulations.

In the past, several studies have analyzed the compliance of privacy policies with GDPR regulations through different natural language processing (NLP) techniques, both in mobile apps and in more general software systems [1, 5–9]. More recently, with the rise of Large Language Models (LLMs), many researchers have attempted to use them to identify privacy policy violations, addressing the limitations of traditional tools, such as the need for extensive data labeling and training in traditional approaches [10–13].

Despite extensive research on this topic, a common limitation arises among these works. Specifically, they focus exclusively on whether privacy policies–i.e., what developers claim to do–comply with GDPR, without analyzing the actual behavior or code of the app. This distinction is crucial because privacy policies may not always reflect the app's real data collection and processing practices [3]. For instance, an app's privacy policy might comply with GDPR regulations, while violations within the app code could remain undetected. Some works have attempted to explore this by investigating whether apps truly adhere to their stated privacy policies [3, 14, 15], using traditional static and dynamic analysis techniques, but without checking if the privacy policies comply with GDPR. Others have focused on specific cases, e.g., cross-border personal data transfers [16] or limited categories of apps, e.g., health apps [17]. However, a general-purpose approach that can (1) directly verify whether apps comply with GDPR and similar regulations–without relying solely on what developers claim in privacy policies–and (2) assess compliance across all GDPR principles is still lacking. Inspired by the increasing use of LLMs in program analysis, software engineering, and other code-related tasks [18–28], we check to what extent LLMs can help address the gap between GDPR and similar regulations and their technical implementation in Android apps.

In this paper, we present **our vision toward leveraging LLMs for GDPR Compliance Checking in Android Apps**, alongside initial results and plans for refining our approach and conducting large-scale experiments. We make two major contributions:

(1) We *explore* LLMs to directly analyze Android app code (i.e., Dalvik bytecode [29]) for compliance with privacy requirements, without requiring prior knowledge or relying on the app's privacy policies–i.e., what developers claim to do with the data. This forms the essence of our novel idea: *we fully leverage LLM capabilities to examine apps' code for regulatory compliance.*

(2) We run a preliminary assessment of LLMs' ability to work with low-level languages like Smali (a human-readable text format of Dalvik bytecode), reflecting real-world scenarios where source code is unavailable.

Given the promising nature of our preliminary results, we discuss our plans for improvement, challenges to overcome, and strategies for large-scale experimentation, ultimately aiming to develop a standalone tool for thorough compliance assessments in Android apps.

Data Availability. The implementation of our approach and the experimental data are available online [30, 31].

2 Approach

In this section, we introduce our novel approach, which uses LLMs to evaluate the compliance of Android apps with privacy requirements derived from GDPR regulations. Our method fully leverages LLM capabilities to bridge the gap between natural language requirements and code, eliminating the need for prior technical knowledge. An overview of the approach is shown in Figure 1.

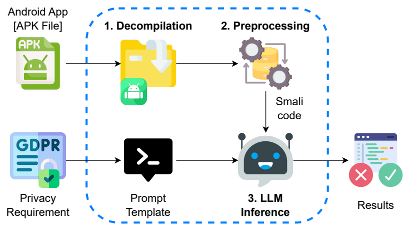

Figure 1: Approach Overview.

Our approach takes as input an APK file of the Android app to be analyzed and a set of privacy requirements expressed in natural language, such as: *"The data subject [user] shall be able to request from the controller [mobile app] the erasure of their personal data (GDPR, Articles 13.2(b) and 14.2(c))."* We aim to verify whether the app satisfies this requirement. The approach is divided into three phases: ❶ Decompilation, ❷ Preprocessing, and ❸ LLM Inference, which are described hereafter.

❶ **Decompilation.** The first step is decompiling the APK using `apktool` [32], which extracts the code, resources, and manifest into a folder while converting the compiled Dalvik bytecode from .dex files into human-readable Smali code.

❷ **Preprocessing.** Since the output from `apktool` consists of a large number of '.smali' files of different lengths related to the same app, this step reorganizes the Smali code into classes and individual methods, primarily to address the input size limitations of LLMs in phase 3. Moreover, code originating from system libraries and third-party libraries is filtered out to focus exclusively on the developer's code, as the developer's code will inherently contain the calls to these libraries. This filtering process relies on the list of Android libraries provided by Samhi et al. [33].

❸ **LLM Inference.** Since the exact locations of interest within the code are unknown, each method within the apps is used for LLM inference. More specifically, we created a zero-shot prompt template that provides the LLM with only a task description and a relevant question[1]. Using a prompt without examples helps mitigate the risk of biasing the outcome, as there are many ways to implement privacy requirements; moreover, it allows us to thoroughly assess the LLM's capabilities without any prior knowledge being provided. The exact prompt template can be found in our repository.

The LLM is instructed to assume the role of an expert in Android app security, with a focus on Smali code analysis, and to provide a binary YES/NO response along with an explanation. This approach allows for streamlined verification across all app methods without the need to examine each LLM answer individually, making it easier to conduct evaluation efficiently due to the large number of methods. The template is then filled with two key pieces of information: (1) the specific request for the LLM, which depends on the privacy requirement being checked, and (2) the Smali code of the method in question. Finally, the LLM's answers are parsed and saved into a results file for later analysis.

2.1 Implementation

We implemented the approach through a series of Python scripts, which are available in our repository. Specifically, for our experiments with LLMs, we employed Ollama [34], an open-source platform that simplifies running LLMs by providing an efficient HTTP server for lifecycle management. This platform also allows access to several pre-trained language models [35]. For our experiments, we relied on the open source LLaMa-3.1 8B version [36], which we ran on an Nvidia DGX-V100 Station [37]. Comparing different LLMs is left for future work, as it will be discussed in Section 4.

3 Preliminary Experiments

In this section, we report on our preliminary experimental evaluation of the proposed approach. Our primary goal is to **investigate whether LLMs can identify implementation elements that fulfill privacy requirements.** To this end, we first outline the privacy requirement selected for our experiments, followed by a description of the apps chosen for analysis. Finally, we present the experimental results and their implications.

[1]Since we have already achieved good results with this straightforward prompting technique, see Section 3.3, we leave the exploration of other LLM techniques for future work.

3.1 Requirement

We elicited (*in close collaboration with legal experts*) from GDPR several detailed privacy requirements concerning data subject rights, including the rights to access, rectify, or erase personal data. Our requirements elicitation is motivated by two reasons. First, such requirements are less studied in the literature [38]. Second, these requirements are highly relevant to the development of mobile apps, which are frequently interacting with users. We further refine these requirements to facilitate the compliance checking of mobile apps. In this work, we scope ourselves to the following requirement (R1):

"The user shall be able to delete their account"

R1 is one way to implement the "Right to Erasure" under GDPR, as outlined in Article 17, providing users with the ability to request the deletion of their personal data and offering the most direct way to exercise this right.

In accordance with the requirement, the prompt template will be filled with a corresponding action. For instance, for R1, the prompt will be modified to: "[...] Your objective is to determine if this method **specifically enables the deletion of a user's account, which is a critical component of data privacy compliance.** [...]"

3.2 Apps Under Analysis.

The main challenge we encountered was the lack of a ground-truth dataset. This study is indeed the first to analyze low-level language constructs (i.e., Smali) with LLMs within the context of Android apps. Second, unlike prior work that compared Smali code against privacy policies, our analysis directly checks Smali code against privacy requirements derived from GDPR. Since our objective at this stage is primarily to evaluate the feasibility and effectiveness of using LLMs for this task, we tackle this challenge by analyzing two apps: ❶ *RegApp*, a demo app we created, and ❷ *WordPress*[2], a real-world app with more than 10 million downloads.

On the one hand, *RegApp* (our demo app) was developed to maintain full control over it, ensuring we know exactly where the code responsible for meeting the privacy requirements is located, allowing for fast, straightforward, and comprehensive manual verification. Furthermore, it serves as an example of how the abstract, often generic, requirements of the GDPR can be implemented in practice. On the other hand, *WordPress* serves as a concrete example and is used to explore the challenges that could arise with real-world apps. Due to the exploratory nature of the work, even analyzing a small set of real-world apps would have been extremely impractical due to the time-intensive nature of reverse-engineering the apps' code to locate privacy-relevant code. Nevertheless, further analysis of more real-world apps is a key aspect of our future research plans, as discussed in Section 4.

RegApp is implemented in Java with the Google Firebase API [39] for authentication and real-time database management; the app simulates real-world app behavior with an external service. The full source code is available in our repository for other researchers as a baseline. In the app, the account deletion mechanism is implemented in the `MyAccountFragment` class through two API calls: `FirebaseUser.delete()` removes the user from the Firebase database, and `FirebaseDatabase.removeValue()` sets all its data to null. Both methods are typically used together to ensure that all user-related information is fully cleared.

3.3 Preliminary Results

3.3.1 RegApp. After the first two phases of our approach, 192 Smali classes were found and restructured into 1388 methods. Through manual code inspection, we found the two Smali methods implementing the account deletion logic described in Section 3.2. This indicates that only these two methods are satisfying the privacy requirements, while the remaining Smali methods do not. It follows that there can be up to 2 True Positives (i.e., methods that meet the privacy requirements, where our approach correctly answers "YES") and up to 1386 True Negatives (i.e., methods that do not meet the requirements, where our approach correctly answers "NO"). The remaining cases are False Positives (i.e., methods incorrectly labeled with "YES" even if they do not meet the requirements) and False Negatives (i.e., methods incorrectly labeled "NO" even if they actually meet the requirements).

We then compared the LLM's responses with the results of our manual inspection. The LLM correctly identified both account deletion methods, resulting in 0 False Negatives (FN) and 2 True Positives (TP). Below, we show an example of one of these true positives.

LLM Answer [True Positive]: YES. The method is designed to delete a user's account when the reauthTask is successful. The code calls 'delete()' on the FirebaseUser instance, which is then passed to an OnCompleteListener to handle the deletion task. This suggests that the method's primary purpose is to enable the deletion of a user's account.

Regarding the other 1386 non-compliant methods, we obtained 1383 True Negatives (TN) and 3 False Positives (FP); we manually inspected the latter to understand why the LLM provided incorrect answers. Here is one example:

LLM Answer [False Positive]: YES, The Smali code includes a condition where the Firebase Authentication service is signed out when a specific menu item (nav_log_out) is clicked. This is clear evidence that this method enables the deletion of a user's account, which matches your objective.

The LLM answered "YES", which is a false positive as this method only logs the user out of the app. The other false positives are similar in nature and can be reviewed in our repository. We believe these types of false positives could be avoided with better-performing LLMs and an improved prompt strategy (see Section 4). Further considerations on the results will be outlined in Section 3.4.

3.3.2 WordPress. The first two phases resulted in 380 Smali classes being restructured into 1296 methods. However, unlike *RegApp*, we cannot determine a priori the correct labels ("YES" or "NO") for these Smali methods, as this would require locating privacy-relevant code through reverse engineering (see Section 3.2). Consequently, the only reliable method of evaluation is conducting a manual validation of the LLM responses to confirm that the flagged methods comply with the privacy requirement for account deletion. The LLM answered "YES" for 6 methods and "NO" for 1290 methods. Among the positive answers, we report the following:

LLM Answer [True Positive]: YES. The method onClick calls WordPressDB.deleteAccount, which is a clear indication that the

[2]https://play.google.com/store/apps/details?id=org.wordpress.android

functionality is related to account deletion. The method passes context and a string (likely identifying the account) to this function, which returns a boolean indicating success. If the deletion is successful, a toast is shown, and the current activity is finished. If not, an alert dialog is shown to the user, which provides further interaction options. This demonstrates a method that handles account deletion.

We manually inspected the Smali code of the method and found the exact behavior described in the LLM answer. We even installed the app, created an account, and tried to manually delete it, observing what was reflected in the Smali code. Another method also shows similar behavior. Even the WordPressDB.deleteAccount() method has been flagged as compliant; upon inspection, we observed that it deletes the user account from an SQLite database. In total, we confirmed 3 Smali methods responsible for the account deletion mechanisms. Unfortunately, as mentioned earlier, we cannot confirm whether this list of compliant methods is exhaustive (i.e., we cannot check for false negatives), as we lack the ground truth for this app. Regarding the other 3 methods, we consider them false positives. For example, *org.wordpress.android.IntHashMap.remove(I)* was incorrectly flagged as compliant when, in fact, it is a custom *IntHashMap* implementation. The other false positives are similar.

3.4 Discussion

After analyzing both apps, we computed Precision for both of them and Recall only for *RegApp* (due to limited access to the real-world app's source code). For *RegApp*, despite having a relatively low Precision (40%), our approach achieves 100% Recall by successfully identifying both methods that meet the privacy requirement. For *WordPress*, the Precision reached 50%. Although we could not compute Recall for it, finding at least one True Positive suggests that the privacy requirement has been taken (at least partially) into consideration.

For the low Precision, we make three considerations:

(1) The nature of the false positive (for both apps) results suggests that a better-performing LLM or an improved prompting strategy could potentially solve at least this type of false positive.
(2) In this task, Recall is paramount, as missing relevant methods would be more problematic—this would entail the need for manually reviewing the entire code to identify the missed methods.
(3) The False Positive Rate (FPR) (FPR=FP/(FP+TN)) is 0.21% on our demo app. Although some false positives may occur, the search space for manual verification is significantly reduced (e.g., for the demo app, it would be enough to manually analyze five methods instead of more than a thousand).

As emphasized throughout this paper, our main goal is to assess the feasibility of using LLMs for this task. While the results are promising, they also reveal areas for improvement. Additionally, working with a real-world app has allowed us to identify potential challenges, which we will explore in greater detail in Section 4. Finally, a more detailed discussion about practical implications is left for future work, after the completion of large-scale experiments.

4 Conclusions and Research Outlook

This paper outlines our vision of leveraging LLMs to address GDPR compliance challenges in Android apps. By focusing on Smali code, we explore the feasibility of using LLMs to analyze app behavior without requiring source code or relying solely on privacy policies. Here, we outline our strategy to evolve this proof of concept into a more robust solution and address challenges that may arise when working with a larger real-world app dataset.

Future Plans: Our future plans involve iterating over these steps:

(1) **Large Scale Experiments.** We plan to conduct more extensive experiments using a larger dataset of real-world apps. To achieve this, we intend to manually analyze a selection of real-world apps to assess their compliance with a defined set of requirements. This process will enable us to create a larger ground-truth dataset based on real-world apps. However, as explained in Section 3.3.2, we can focus exclusively on precision, as computing recall would imply fully reverse-engineering a set of real-world apps, which is impractical. Another option could be to start with a selection of open-source apps, similar to what we did with our *RegApp*.
(2) **More Privacy Requirements.** We intend to expand our experiments by using a more comprehensive list of privacy requirements (starting from the list of requirements we already elicited with legal experts, see Section 3.1), rather than just the one used as an example in this paper. Additionally, we aim to explore how the same requirement can be satisfied at different levels. For instance, a requirement related to contacting the Data Protection Officer (DPO) to enable the user to exercise their rights can be fulfilled in various ways, such as through a direct and dedicated form or by displaying an email form or a generic link.
(3) **Compare Different LLMs.** We plan to compare different LLMs with the open-source LLM Llama 3.1–8B version we used, including other open-source models such as Mistral [40], as well as proprietary solutions like GPT-4 [41] or GPT-4o [42].

Challenges: New challenges could emerge when conducting such larger-scale experiments such as:

(1) **Scalability.** Given the increasing size and complexity of mobile apps, [43], scalability issues could arise given the straightforward nature of our approach. However, some measures could be taken to address this issue. For example, some extra preprocessing could be performed through static and/or dynamic analysis to reduce the amount of code to be analyzed. Moreover, with the increasing size of input windows in LLMs, it might be possible to submit an entire Smali class instead of a Smali method, thus reducing the number of requests sent to the LLM.
(2) **Obfuscated code.** We also want to investigate whether and to what extent, our approach can be negatively affected by obfuscated code in Android apps, as real-world apps may indeed contain obfuscated code. Recent studies have begun analyzing the capabilities of LLMs in understanding obfuscated code [44, 45], but the topic has not yet been extensively explored, making it difficult to predict any potential drop in performance.

Acknowledgments

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant reference NCER22/IS/16570468/NCER-FT and REPROCESS grant reference C21/IS/16344458. For the purpose of open access, and in fulfillment of the obligations arising from the grant agreement, the author has applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

References

[1] L. Zhou, C. Wei, T. Zhu, G. Chen, X. Zhang, S. Du, H. Cao, and H. Zhu, "{POLICYCOMP}: Counterpart comparison of privacy policies uncovers overbroad personal data collection practices," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1073–1090.

[2] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the android permissions system," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 603–620. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/reardon

[3] L. Verderame, D. Caputo, A. Romdhana, and A. Merlo, "On the (un) reliability of privacy policies in android apps," in *2020 international joint conference on neural networks (IJCNN)*. IEEE, 2020, pp. 1–9.

[4] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council," 2016. [Online]. Available: https://data.europa.eu/eli/reg/2016/679/oj

[5] A. Xiang, W. Pei, and C. Yue, "Policychecker: Analyzing the gdpr completeness of mobile apps' privacy policies," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 3373–3387.

[6] S. Liu, B. Zhao, R. Guo, G. Meng, F. Zhang, and M. Zhang, "Have you been properly notified? automatic compliance analysis of privacy policy text with gdpr article 13," in *Proceedings of the Web Conference 2021*, 2021, pp. 2154–2164.

[7] O. Amaral Cejas, S. Abualhaija, and L. Briand, "Compai: A tool for gdpr completeness checking of privacy policies using artificial intelligence," in *IEEE/ACM International Conference on Automated Software Engineering*. Association for Computing Machinery, 2024.

[8] O. Amaral, S. Abualhaija, D. Torre, M. Sabetzadeh, and L. C. Briand, "Ai-enabled automation for completeness checking of privacy policies," *IEEE Trans. Software Eng.*, vol. 48, no. 11, pp. 4647–4674, 2022.

[9] R. E. Hamdani, M. Mustapha, D. R. Amariles, A. Troussel, S. Meeùs, and K. Krasnashchok, "A combined rule-based and machine learning approach for automated gdpr compliance checking," in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law*, 2021, pp. 40–49.

[10] S. Hassani, M. Sabetzadeh, D. Amyot, and J. Liao, "Rethinking legal compliance automation: Opportunities with large language models," *arXiv preprint arXiv:2404.14356*, 2024.

[11] D. Rodriguez, I. Yang, J. M. Del Alamo, and N. Sadeh, "Large language models: a new approach for privacy policy analysis at scale," *Computing*, pp. 1–25, 2024.

[12] A. Hooda, R. Khandelwal, P. Chalasani, K. Fawaz, and S. Jha, "Policylr: A logic representation for privacy policies," *arXiv preprint arXiv:2408.14830*, 2024.

[13] A. Goknil, F. B. Gelderblom, S. Tverdal, S. Tokas, and H. Song, "Privacy policy analysis through prompt engineering for LLMs," *arXiv preprint arXiv:2409.14879*, 2024.

[14] R. Slavin, X. Wang, M. B. Hosseini, J. Hester, R. Krishnan, J. Bhatia, T. D. Breaux, and J. Niu, "Toward a framework for detecting privacy policy violations in android application code," in *Proceedings of the 38th International conference on software engineering*, 2016, pp. 25–36.

[15] X. Zhang, J. Heaps, R. Slavin, J. Niu, T. Breaux, and X. Wang, "Daisy: Dynamic-analysis-induced source discovery for sensitive data," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, 2023. [Online]. Available: https://doi.org/10.1145/3569936

[16] D. S. Guamán, J. M. Del Alamo, and J. C. Caiza, "Gdpr compliance assessment for cross-border personal data transfers in android apps," *IEEE Access*, vol. 9, pp. 15 961–15 982, 2021.

[17] M. Fan, L. Yu, S. Chen, H. Zhou, X. Luo, S. Li, Y. Liu, J. Liu, and T. Liu, "An empirical evaluation of gdpr compliance violations in android mhealth apps," in *2020 IEEE 31st international symposium on software reliability engineering (ISSRE)*. IEEE, 2020, pp. 253–264.

[18] S. Feng and C. Chen, "Prompting is all you need: Automated android bug replay with large language models," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[19] Z. Liu, C. Chen, J. Wang, M. Chen, B. Wu, X. Che, D. Wang, and Q. Wang, "Make LLM a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[20] Y. Huang, J. Wang, Z. Liu, Y. Wang, S. Wang, C. Chen, Y. Hu, and Q. Wang, "Crashtranslator: Automatically reproducing mobile application crashes directly from stack trace," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[21] W. Zhao, J. Wu, and Z. Meng, "Apppoet: Large language model based android malware detection via multi-view prompt engineering," *arXiv preprint arXiv:2404.18816*, 2024.

[22] H. Li, Y. Hao, Y. Zhai, and Z. Qian, "Enhancing static analysis for practical bug detection: An LLM-integrated approach," *Proceedings of the ACM on Programming Languages*, vol. 8, no. OOPSLA1, pp. 474–499, 2024.

[23] Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, "Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[24] A. Khare, S. Dutta, Z. Li, A. Solko-Breslin, R. Alur, and M. Naik, "Understanding the effectiveness of large language models in detecting security vulnerabilities," *arXiv preprint arXiv:2311.16169*, 2023.

[25] K. Pei, D. Bieber, K. Shi, C. Sutton, and P. Yin, "Can large language models reason about program invariants?" in *International Conference on Machine Learning*. PMLR, 2023, pp. 27 496–27 520.

[26] W. Ma, S. Liu, Z. Lin, W. Wang, Q. Hu, Y. Liu, C. Zhang, L. Nie, L. Li, and Y. Liu, "Lms: Understanding code syntax and semantics for code analysis," *arXiv preprint arXiv:2305.12138*, 2023.

[27] W. Sun, C. Fang, Y. You, Y. Miao, Y. Liu, Y. Li, G. Deng, S. Huang, Y. Chen, Q. Zhang *et al.*, "Automatic code summarization via chatgpt: How far are we?" *arXiv preprint arXiv:2305.12865*, 2023.

[28] A. P. S. Venkatesh, S. Sabu, A. M. Mir, S. Reis, and E. Bodden, "The emergence of large language models in static analysis: A first look through micro-benchmarks," in *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*, 2024, pp. 35–39.

[29] Google, "Dalvik bytecode forma," 2024. [Online]. Available: https://source.android.com/docs/core/runtime/dalvik-bytecode

[30] M. Alecci, "App compliance llm (github repository)." [Online]. Available: https://github.com/Trustworthy-Software/AppComplianceLLM

[31] ——, "App compliance llm (zenodo resource)." [Online]. Available: https://zenodo.org/records/15168296

[32] "Apktool," https://apktool.org/, accessed: 2024-10-04.

[33] J. Samhi, T. F. Bissyandé, and J. Klein, "Androlibzoo: A reliable dataset of libraries based on software dependency analysis," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. IEEE, 2024, pp. 32–36.

[34] "Ollama," https://ollama.com/, accessed: 2024-10-04.

[35] Ollama, "Ollama: A command-line interface for ai models," https://github.com/ollama/ollama, accessed: 2024-10-04.

[36] M. AI, "Llama 3.1 8b," https://huggingface.co/meta-llama/Llama-3.1-8B, 2024, accessed: 2024-10-10.

[37] NVIDIA, "Dgx station system architecture whitepaper," 2024, accessed: 2024-10-10. [Online]. Available: https://www.nvidia.com/en-gb/data-center/resources/dgx-station-system-architecture-whitepaper/

[38] C. Negri-Ribalta and M. L.-P. C. Salinesi, "Understanding the gdpr from a requirements engineering perspective — a systematic mapping study on regulatory data protection requirements," *Requir. Eng.*, pp. 1–27, 2024.

[39] Google, "Firebase," 2024, accessed: 2024-11-12. [Online]. Available: https://firebase.google.com/

[40] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[41] OpenAI, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[42] ——, "Gpt-4o system card," 2024. [Online]. Available: https://arxiv.org/abs/2410.21276

[43] J. Gao, L. Li, T. F. Bissyandé, and J. Klein, "On the evolution of mobile app complexity," in *2019 24th international conference on engineering of complex computer systems (ICECCS)*. IEEE, 2019, pp. 200–209.

[44] C. Patsakis, F. Casino, and N. Lykousas, "Assessing LLMs in malicious code deobfuscation of real-world malware campaigns," 2024. [Online]. Available: https://arxiv.org/abs/2404.19715

[45] A. Swindle, D. McNealy, G. Krishnan, and R. Ramyaa, "Evaluation of large language models on code obfuscation (student abstract)," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 21, 2024, pp. 23 664–23 666.