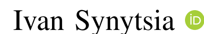
Towards Generating App Feature Descriptions Automatically with LLMs: the Setapp Case Study

Yevhenii Peteliev
MacPaw
Kyiv, Ukraine
zhenya.peteliev@macpaw.com

Ivan Synytsia 
MacPaw
Kyiv, Ukraine
sip@macpaw.com

Nataliia Stulova
MacPaw
Kyiv, Ukraine
nata.stulova@macpaw.com

***Abstract*—For any application to understand what it allows its users to do, we must rely on app functionality descriptions provided by software developers on app pages in marketplaces and in the release notes, the *developer view* or *claimed features*. User reviews and public discussions on thematic forums can serve as another source of information about app's features, and sometimes new features are inspired by such *user view*. However, little research has been done on app artifact analysis to distill *actual high-level features*, with researchers focusing on bytecode analysis to understand low-level app behaviors, such as API calls, without necessarily mapping those to features. Herein, we explore the possibilities of LLMs to reconstruct the app features and functionality descriptions from the (middle-level) app artifact information to bridge the perspective and knowledge gaps. We extract diverse unstructured text strings from 235 macOS app artifacts obtained from the Setapp app store and prompt the GPT-4o LLM for a list of possible feature descriptions, which we later compare with the human-written app's feature list in the app store. We observe minor differences in lexical structure in terms of part-of-speech counts, and the semantic similarity (cosine) score varies between 0.47–0.76 with GloVe embeddings and between 0.57–0.77 with BERT ones, meaning that even naïve prompting can produce similar enough app feature descriptions w.r.t. the human-produced oracle. Our results show the potential of the LLM use for automatic/assisted app feature description generation in marketplaces and for contrasting the claimed and actual app behavior for detecting any discrepancies.**

***Index Terms*—Large language models, app behavior, macOS**

I. Introduction

As of August 2024, major app stores hold over 4 million apps[1], each listed with a human-written description of the app functionality, its technical specifications, and various user-produced metadata like rankings, reviews, numbers of downloads, etc. Extracting and comparing app features from these diverse artifacts via app store mining [1]–[3] allows to better understand the interplay between the business, developer, and customer perspectives on the software and its functionality. While there is no standard format for describing app functionality in a store description, describing key app features is a common practice. Moreover, as the app evolves, features are added or deprecated, and prioritizing feature elicitation during the app's evolution has a significant impact on the app's performance in the marketplace [4]. Determining which app features are important to add and maintain is a challenging business problem from the requirements engineering perspective, and natural language artifacts from the app stores are naturally among the first ones studied. One useful source of information is user reviews, from which (requested) app features can be extracted [5]–[8] and prioritized in app evolution, or features that need fixing [9] for app maintenance. Research in the area has also shown that candidate app features can be mined from social media [10] discussions, or even similar app descriptions [11]–[13], with app feature descriptions holding the key data about the app behavior [14].

Combined with source code information, app feature detection from user reviews can guide the implementation of requested features and release planning [15], but also to compare the *review-to-behavior fidelity* [16] and *description-to-behavior fidelity* [17]–[19] of an app. For these tasks, multiple artifacts can be studied jointly, either pairwise, like app description topics and API calls [20], app binaries and release notes [21], or in many-to-many format between artifacts from the app store and the app package file, such as in [22]. App features can also be extracted from the non-textual artifacts automatically in the spirit of reverse engineering, such as execution traces of instrumented apps [23], or purely from the strings appearing in various files of app packages [24], [25]. Such approaches contribute to the security research in app ecosystems, with a special focus on malware detection [26].

However, in many cases, the proposed tools and techniques are tailored to specific artifacts as inputs, reducing their scalability, as noted in [19], and extensibility to other app ecosystems than the one they are proposed for. At the same time, advances in recent years in large language models (LLMs) and the broadness of their application for tasks involving bidirectional code and natural language generation [27], [28] make it interesting to explore the use of LLMs for app feature extraction tasks from app package contents. While coming with the disadvantage of being a black box, LLMs can potentially solve the tool extensibility problem or offer a complementary solution if we *consider app feature descriptions extraction as a sequence-to-sequence modeling task*.

In this work, we perform the initial validation of this idea. We describe a pilot case study on 235 apps from the Setapp marketplace[2], focusing on apps for macOS. We aim to answer

[1] https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[2] https://setapp.com/

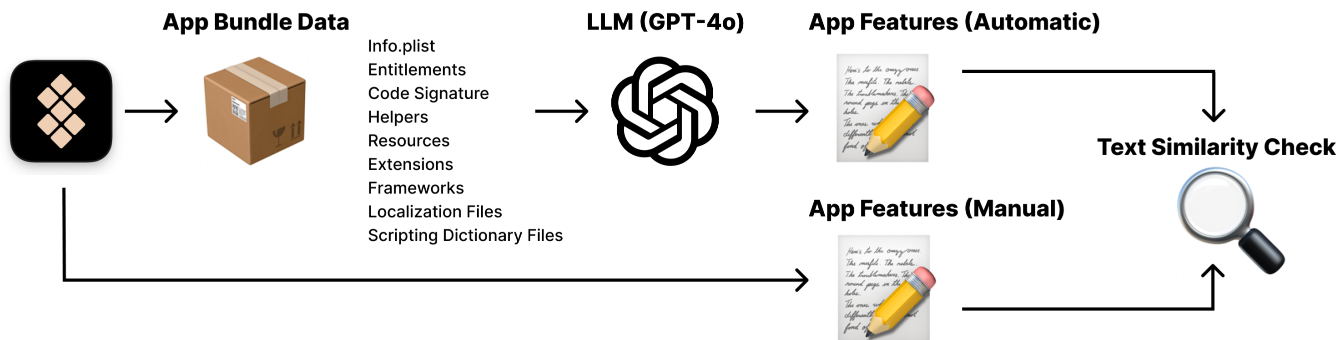


Fig. 1: General approach pipeline

the following question: *given the unstructured textual information extracted from non-code and non-text app artifacts, how closely can we match the app feature descriptions that are mentioned in the textual app artifacts?* We report our preliminary results and the plans for extending this work.

II. macOS Application Bundles Contents

A macOS application bundle[3] is a directory that contains the executable code and all the resources required for the application to function. Bundles present a user-friendly package, but they are, in fact, structured directories that follow a well-defined hierarchy. This structure ensures that the application remains self-contained, simplifying management, execution, and interaction within the macOS environment.

At the core of the bundle is the Contents directory, which organizes key subdirectories and files. The MacOS subdirectory contains the Mach-O executable binary, which includes the app's machine code, dynamic linking information, and metadata. The Resources directory includes non-executable assets like images, localized strings, sound files, and other interface components, enabling the app to support multiple languages and customization. The Info.plist file in it is an XML-based property list that contains essential configuration information about the app. It stores key-value pairs such as the app's bundle identifier, version number, display name, supported interface orientations, privacy permissions, and other data required for the system to run the app correctly.

Other important subdirectories include Frameworks, which stores shared libraries or custom frameworks the app relies on, and Helpers, which contains auxiliary executables like background agents or daemons. The Extensions and XPCServices directories enable modular functionality, allowing the app to integrate with system features or run isolated tasks securely. The Library directory may store additional resources, such as configuration files, databases, or caches necessary for runtime.

The _CodeSignature directory contains cryptographic information that verifies the app's integrity and authenticity. In modern macOS environments, it is critical for ensuring that the app meets security requirements, enforced by Gatekeeper and notarization. Without a valid code signature, the system may block the app from running, as it prevents tampering and unauthorized modifications.

[3] https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/BundleTypes/BundleTypes.html

III. Approach

Figure 1 shows the main steps of our case study. We first obtain the application bundles from the app store and the human-written app feature descriptions that would serve as an oracle (see Sec. IV for details). Next, from the app bundles we extract the textual data (Sec. III-A), and pass it to a LLM in a prompt, getting back the synthetic app feature descriptions (Sec. III-B). Finally, we compare the LLM output with the oracle (Sec. III-C).

A. App Bundle Analysis

The process begins by analyzing the middle-level macOS bundle artifacts (i.e. non-binary files that contain text strings) to collect the necessary data for further analysis[4] During the data collection phase, the algorithm begins with *Mach-O Analysis*, examining the Mach-O executable binary. By analyzing this file, the algorithm collects vital information such as entitlements, code-signing details, and dynamic frameworks. This step does not require decompiling the binary file and inspecting its source code.

The next step involves *Info.plist Analysis*, where key configuration details from the information property list file are collected. Additionally, the algorithm performs *Resources Analysis*, examining non-executable assets such as fonts, images, documents, and localization files, as well as analyzing AppleScript and JavaScript scripting dictionary files[5].

Further analysis includes the *Helpers Analysis*, which focuses on collecting file names from Helpers, XPCServices, and Library folders, contributing to the app's background processes and functionality. Similarly, the *Extensions Analysis* and *Dependencies Analysis* involve gathering file names from app extensions, plug-ins, and dynamic libraries to identify linked libraries and external dependencies.

We do not deliberately collect any core branding information from the bundle, such as the application's name,

[4] see a sample of extracted data at [29] for reference.

[5] https://developer.apple.com/library/archive/documentation/LanguagesUtilities/Conceptual/MacAutomationScriptingGuide/AboutScriptingTerminology.html

developer, or other brand-related details, though it might still be mentioned in the rest of the collected data.

B. LLM-Based Feature Extraction

We utilize the GPT-4o LLM to extract key features from raw application data. The model operates with a context window size of 128,000 input tokens and can generate a maximum of 16,384 output tokens. A custom prompt, provided as a parameter in our application, guides the model's focus during the extraction process. It has three parts: (a) modeling the feature format, (b) modeling the output JSON structure, and (c) the unstructured data we extract from the app bundles. The (a) part looks like this: *'You need to analyze information about a macOS app and create a human-readable list of top-4 likely tasks that users can do with this app. Start the task description with a verb, and keep the description short, between 5 and 15 words. You should use only the information I provide for analysis. Do not use data from the Internet or your knowledge base. If you are unsure of the answer, simply write "N/A".'*

C. Text Comparison

We work with app feature descriptions, which are short sentences in English, and in the context of the chosen app store, it is always 4 sentences per app. To compare two app feature descriptions with each other, we convert each app feature description sentence into a word vector and work with these vectors jointly by arranging them into a matrix. After obtaining two matrices, one for the original, human-written feature descriptions and one produced by an LLM, we calculate pairwise similarities between vectors from both matrices and finally compute the average *cosine similarity*.

We are using spaCy 3.7.5[6] and the pre-trained GloVe word embeddings of en_core_web_md to convert our source English sentences into word vectors. We also explore an alternative transformer-based embeddings of 'en_core_web_trf' from the spacy-transformers package to see, if context information is represented better. For computing cosine similarity between the matrices, we are using sklearn.metrics.pairwise[7].

IV. Evaluation

We collected all 235 unique apps available in February 2024 from the Setapp marketplace. Our decision to focus on this macOS app marketplace is motivated by two factors: (a) unlike in other app distribution platforms, Setapp app descriptions always list 4 features for every app, and (b) those descriptions are written by the developers and then additionally verified by the store managers. Figure 2 shows the app category distribution, with 3 categories dominating 2/3 of the dataset.

The batch size of requests to send concurrently to the ChatGPT was set to the default value of 20 to reduce the risk of API timeouts. We have set the temperature parameter to 0.7 out of 2.0 as per best current practices to reduce hallucinations risk.[8] We performed a single API query. With our prompt,

[6]https://spacy.io/models/en

[7]https://scikit-learn.org/stable/modules/metrics.html

[8]https://platform.openai.com/docs/api-reference/chat/create

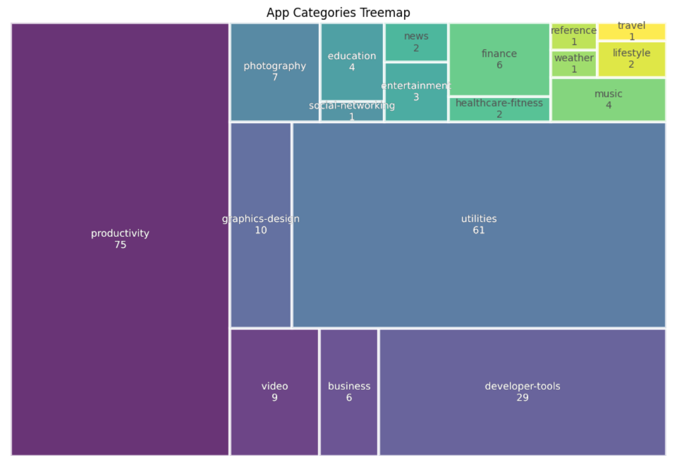


Fig. 2: App distribution in categories

we have managed to approximate the shape for the feature sentences well in length, even though LLM-generated texts use less unique words compared to human-produced texts (Tab. I).

TABLE I: Word count statistics

Source	Min	Max	Avg	Median	Total	Unique
Human	2	12	5.96	6	5639	1213
GPT-4o	4	12	6.50	6	6195	1084

Restricting the output length was straightforward, but we noticed that if we do not describe in detiail the desired feature description shape, the results vary:

- App name is listed: *MarsEdit allows users to create and edit blog posts with a rich text editor.*
- Addition of 'Users can' phrase *Users can manage multiple blogs from different platforms like WordPress, Tumblr, and Blogger.*, *Allows users to set wallpapers to change at regular intervals.*
- Buzzwords and generalizations returned: *General utility functions to enhance productivity by managing sleep and display settings.*

We collected part of speech (POS) statistics from the spaCy representations of the texts. Table II showcases lexical variability, with human texts using more adverbs, pronouns, and determiners, while LLM-generated texts contain more nouns, propositions, and coordinating conjunctions. We also observed that the lemma *you* appears 248 times in the human-written feature descriptions and 24 times in the LLM-generated ones.

For comparing the average semantic similarity scores (Fig. 3) we have observed (as expected) that BERT embeddings yield higher similarity scores compared to GloVe embeddings, with scores in both cases being quite high. The above results are reassuring in the sense that LLMs can be used, though as a black-box, for a sequence-to-sequence modeling task to obtain high-level app feature descriptions from mid-level app artifacts. While a fine-grained qualitative study would allow us to understand more nuances on the accuracy of the generated feature descriptions, for instance, if

TABLE II: Parts of speech (POS) statistics for all 235 apps

Source	ADJ	ADP	ADV	AUX	CCONJ	DET	INTJ	NOUN	NUM	PART	PRON	PROPN	SCONJ	VERB	X
Human	435	682	156	33	267	266	3	2055	22	58	290	249	46	1063	14
GPT-4o	407	699	93	10	511	146	0	2652	3	61	26	435	16	1135	1

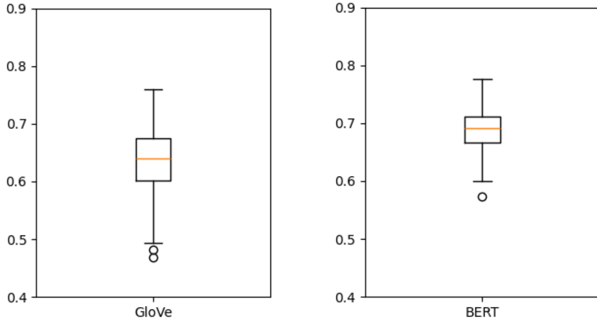


Fig. 3: Box plots of similarity scores distributions for different embeddings on a scale from 0 (min) to 1 (max).

the top high-level feature descriptions generated by the LLM match those chosen and described by the human author(s), it is out of scope of the current paper.

V. Related Work

To the best of our knowledge, this is the first paper to study the task of generating app features for macOS applications with LLMs, but similar work on analyzing multiple non-code app artifacts is reported for the Android ecosystem.

Shabtai *et al.* [24] statically extract data from .apk files of 2285 Android apps for subsequent ML-based classification of apps into tools and games categories. Authors extract various features of the .apk files, parse relevant XML files, and (unlike us) analyze the Java bytecode of the .dex files. Also, unlike us, they do not collect app metadata describing its functionality, using only the app category. In the continuation of our current work, we also aim to rank the usefulness of the features we are extracting from the macOS .app bundles

Kowalczyk *et al.* [22] perform an in-depth study of 14 popular apps from the Google Play Store. They collect much more varied app store metadata, compared to us: developer rating, user rating, user comments, number of installs, screenshots, video (if uploaded), permission request text, listed interactive elements text, whether the app contained in-app purchases, and the text in the app's description. Also unlike us, but similarly to [24], they work with the binary file of the app, disassembling the .apk file into smali byte code files with apktool and decompiling the .apk file with dex2jar to obtain the app's Java source code files. Overall, the authors study a set of 16 different artifacts for each app, and conclude that the output of these must be pieced together to form a complete understanding of the app's true behavior, a conclusion that we share. Currently, we do not work with the contents of the Mach-O binary file, but we will explore that in future work.

The most related to our is the work of Md. Shamsujjoha *et al.* [25], where the authors collect 24k Android apps, and their tool REACT collects an app's .apk file extracts all method names used in the raw code, XML data values, and GUI strings, and then categorizes apps based on the similarity of extracted features using topic modeling results. REACT parses three types of information from decompiled app directories: (i) the entire directory of smali files to extract all method names, (ii) string.xml file to extract all XML data values, (iii) every image files from entire decompiled app directory to extract the GUI label text used in the app. We are not working with image files at the moment, and we do not decompile the binary files, but we also collect relevant information from the XML files available in the app's package.

VI. Conclusions and Future Work

We have performed a pilot study into generating app feature descriptions with LLMs from unstructured data extracted from app artifacts of macOS apps to test a OS- and ecosystem-agnostic approach for app features extraction from app artifacts based on LLM sequence-to-sequence modeling capabilities. Our results show that the approach is promising, though a more thorough qualitative evaluation and a quantitative one on a larger dataset are required.

Specifically, we plan to conduct an evaluation of our technique on a larger app set[9] with more diverse app descriptions. Additionally, we plan to assess whether a LLM could generate all the features that a human would list in an app description, using precision and recall metrics for this evaluation. We will clarify how much information is needed from different artifacts within the app bundle and whether any artifacts, such as the Info.plist file, should be given higher priority or weight. We plan to extend our analysis to include classes, methods, and other relevant information from the app's binary. We also plan to look in more detail into the lexical richness [30] of the generated text (e.g., MATTR and MTLD metrics [31]) and study the possible impact of the app category on the feature description dictionary, to account for the observed category imbalance. Lastly, we plan to compare the performance and resource requirements of larger cloud-based OpenAI LLMs to local models such as the LLaMA family.

VII. Acknowledgements

We thank the Armed Forces of Ukraine for providing security to complete this work. We thank Mykola Antoniuk for his help during the development of this work. We thank the anonymous reviewers whose comments have helped to improve this paper.

[9]The macOS Applications Bundles and Metadata dataset from https://research.macpaw.com/datasets is a straightforward option.

REFERENCES

[1] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for app stores," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 108–111, June 2012.

[2] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "App store analysis: Mining app stores for relationships between customer, business and technical characteristics," 2014.

[3] F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang, "Feature lifecycles as they spread, migrate, remain, and die in App Stores," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pp. 76–85, Aug. 2015.

[4] H. Wu, W. Deng, X. Niu, and C. Nie, "Identifying Key Features from App User Reviews," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 922–932, May 2021.

[5] T. Johann, C. Stanik, A. M. Alizadeh B., and W. Maalej, "SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pp. 21–30, Sept. 2017.

[6] F. A. Shah, K. Sirts, and D. Pfahl, "Is the SAFE Approach Too Simple for App Feature Extraction? A Replication Study," in *Requirements Engineering: Foundation for Software Quality* (E. Knauss and M. Goedicke, eds.), (Cham), pp. 21–36, Springer International Publishing, 2019.

[7] H. Malik, E. M. Shakshuki, and W.-S. Yoo, "Comparing mobile apps by identifying 'Hot' features," *Future Generation Computer Systems*, vol. 107, pp. 659–669, June 2020.

[8] Q. Motger, A. Miaschi, F. Dell'Orletta, X. Franch, and J. Marco, "T-FREX: A Transformer-based Feature Extraction Method from Mobile App Reviews," in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 227–238, Mar. 2024.

[9] S. A. Licorish, B. T. R. Savarimuthu, and S. Keertipati, "Attributes that Predict which Features to Fix: Lessons for App Store Mining," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE '17, (New York, NY, USA), pp. 108–117, Association for Computing Machinery, June 2017.

[10] M. Nayebi, H. Cho, and G. Ruhe, "App store mining is not enough for app improvement," *Empirical Software Engineering*, vol. 23, pp. 2764–2794, Oct. 2018.

[11] H. Jiang, J. Zhang, X. Li, Z. Ren, D. Lo, X. Wu, and Z. Luo, "Recommending New Features from Mobile App Descriptions," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, pp. 22:1–22:29, Oct. 2019.

[12] H. Liu, Y. Wang, Y. Liu, and S. Gao, "Supporting features updating of apps by analyzing similar products in App stores," *Information Sciences*, vol. 580, pp. 129–151, Nov. 2021.

[13] M. Assi, S. Hassan, Y. Tian, and Y. Zou, "FeatCompare: Feature comparison for competing mobile apps leveraging user reviews," *Empirical Software Engineering*, vol. 26, p. 94, July 2021.

[14] M. K. Uddin, Q. He, J. Han, and C. Chua, "Comparison of Text-Based and Feature-Based Semantic Similarity Between Android Apps," in *Web Information Systems Engineering – WISE 2020* (Z. Huang, W. Beek, H. Wang, R. Zhou, and Y. Zhang, eds.), (Cham), pp. 530–545, Springer International Publishing, 2020.

[15] A. Ciurumelea, A. Schaufelbuhl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, (Klagenfurt, Austria), pp. 91–102, IEEE, Feb. 2017.

[16] D. Kong, L. Cen, and H. Jin, "AUTOREB: Automatically Understanding the Review-to-Behavior Fidelity in Android Applications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, (New York, NY, USA), pp. 530–541, Association for Computing Machinery, Oct. 2015.

[17] L. Yu, X. Luo, C. Qian, S. Wang, and H. K. N. Leung, "Enhancing the Description-to-Behavior Fidelity in Android Apps with Privacy Policy," *IEEE Transactions on Software Engineering*, vol. 44, pp. 834–854, Sept. 2018.

[18] C. Zhang, H. Wang, R. Wang, Y. Guo, and G. Xu, "Re-checking App Behavior against App Description in the Context of Third-party Libraries," pp. 665–710, July 2018.

[19] R. Schmicker, F. Breitinger, and I. Baggili, "AndroParse - An Android Feature Extraction Framework and Dataset," in *Digital Forensics and Cyber Crime* (F. Breitinger and I. Baggili, eds.), (Cham), pp. 66–88, Springer International Publishing, 2019.

[20] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), pp. 1025–1035, Association for Computing Machinery, May 2014.

[21] D. Domínguez-Álvarez, D. Toniuc, and A. Gorla, "ReChan: an automated analysis of Android app release notes to report inconsistencies," in *Proceedings of the 9th IEEE/ACM International Conference on Mobile Software Engineering and Systems*, MOBILESoft '22, (New York, NY, USA), pp. 73–83, Association for Computing Machinery, Oct. 2022.

[22] E. Kowalczyk, A. M. Memon, and M. B. Cohen, "Piecing together app behavior from multiple artifacts: A case study," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 438–449, Nov. 2015.

[23] Q. Xin, F. Behrang, M. Fazzini, and A. Orso, "Identifying Features of Android Apps from Execution Traces," in *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pp. 35–39, May 2019.

[24] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated Static Code Analysis for Classifying Android Applications Using Machine Learning," in *2010 International Conference on Computational Intelligence and Security*, pp. 329–333, Dec. 2010.

[25] M. Shamsujjoha, J. Grundy, L. Li, H. Khalajzadeh, and Q. Lu, "Checking App Behavior Against App Descriptions: What If There are No App Descriptions?," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pp. 422–432, May 2021.

[26] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, "Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[27] S. Fakhoury, A. Naik, G. Sakkas, S. Chakraborty, and S. K. Lahiri, "LLM-Based Test-Driven Interactive Code Generation: User Study and Empirical Evaluation," *IEEE Transactions on Software Engineering*, vol. 50, pp. 2254–2268, Sept. 2024.

[28] K. Shi, D. Altınbüken, S. Anand, M. Christodorescu, K. Grünwedel, A. Koenings, S. Naidu, A. Pathak, M. Rasi, F. Ribeiro, B. Ruffin, S. Sanyam, M. Tabachnyk, S. Toth, R. Tu, T. Welp, P. Yin, M. Zaheer, S. Chandra, and C. Sutton, "Natural Language Outlines for Code: Literate Programming in the LLM Era," Aug. 2024. arXiv:2408.04820 [cs].

[29] Y. Peteliev, I. Synytsia, and N. Stulova, "Towards Generating App Feature Descriptions Automatically with LLMs: the Setapp Case Study, Running Example ," Dec. 2024. https://doi.org/10.5281/zenodo.14450038.

[30] A. Muñoz-Ortiz, C. Gómez-Rodríguez, and D. Vilares, "Contrasting Linguistic Patterns in Human and LLM-Generated News Text," *Artificial Intelligence Review*, vol. 57, p. 265, Aug. 2024.

[31] G. Martínez, J. A. Hernández, J. Conde, P. Reviriego, and E. Merino-Gómez, "Beware of Words: Evaluating the Lexical Diversity of Conversational LLMs using ChatGPT as Case Study," *ACM Trans. Intell. Syst. Technol.*, Sept. 2024.