Revisiting Android App Categorization

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

Jordan Samhi
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
jordan.samhi@cispa.de

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

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

ABSTRACT

Numerous tools rely on automatic categorization of Android apps as part of their methodology. However, incorrect categorization can lead to inaccurate outcomes, such as a malware detector wrongly flagging a benign app as malicious. One such example is the *SlideIT Free Keyboard* app, which has over 500 000 downloads on Google Play. Despite being a "Keyboard" app, it is often wrongly categorized alongside "Language" apps due to the app's description focusing heavily on language support, resulting in incorrect analysis outcomes, including mislabeling it as a potential malware when it is actually a benign app. Hence, there is a need to improve the categorization of Android apps to benefit all the tools relying on it.

In this paper, we present a comprehensive evaluation of existing Android app categorization approaches using our new ground-truth dataset. Our evaluation demonstrates the notable superiority of approaches that utilize app descriptions over those solely relying on data extracted from the APK file, while also leaving space for potential improvement in the former category. Thus, we propose two innovative approaches that effectively outperform the performance of existing methods in both description-based and APK-based methodologies. Finally, by employing our novel description-based approach, we have successfully demonstrated that adopting a higher-performing categorization method can significantly benefit tools reliant on app categorization, leading to an improvement in their overall performance. This highlights the significance of developing advanced and efficient app categorization methodologies for improved results in software engineering tasks.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Security and privacy** → **Software and application security**.

KEYWORDS

Android Security, Static Analysis, App Categorization

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ICSE '24, April 14–20, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0217-4/24/04.
https://doi.org/10.1145/3597503.3639094

ACM Reference Format:
Marco Alecci, Jordan Samhi, Tegawendé F. Bissyandé, and Jacques Klein. 2024. Revisiting Android App Categorization. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3597503.3639094

1 INTRODUCTION

Automatic categorization of Android apps has proven to be a useful tool for addressing numerous software engineering tasks. The applications of categorization encompass a wide range of tasks, including but not limited to anomaly detection [8, 22, 61], detection of miscategorized apps [6, 55], malware detection [31, 60], or classifying malware into distinct families [28, 35, 37].

One of the most popular examples is the CHABADA framework developed by Gorla et al. [22], which focuses on detecting applications that exhibit behavior inconsistent with their provided descriptions in order to find potential malicious apps. CHABADA leverages app descriptions to categorize Android apps, and subsequently employs unsupervised One-Class SVM anomaly detection to identify outliers based on API usage patterns. However, an inadequate categorization approach can significantly impact the overall accuracy of CHABADA, leading to numerous false positives (i.e., a goodware detected as a malware) and/or false negatives. For example, the authors themselves acknowledged in their paper that the app *SlideIT Free Keyboard*, which allows users to insert text by sliding their finger along the keyboard, was categorized alongside language apps because over half of the app's description focuses on language support. The app was wrongly detected as an anomaly and potential malware, despite its harmless nature, due to the wrong categorization. However, without a manual inspection, similar to the one conducted by the authors of CHABADA for this app, it becomes challenging to determine whether factors like the number of false positives are primarily influenced by the app categorization module or the anomaly detection phase, as we can only evaluate the combined effect of both.

Inaccurate categorization, as seen in *SlideIT Free Keyboard*, underscores the need for a more precise categorization to minimize such errors. Hence, as a starting point, we initiated our research on automatic app categorization with a comprehensive literature search to gather existing categorization approaches with the primary aim of evaluating and effectively comparing their performance. However, during our attempt to compare the existing approaches that we retrieved, we encountered a significant obstacle: *how can we*

evaluate and compare their performances? Indeed, despite the extensive research conducted on automatic app categorization, there remains a lack of a ground-truth dataset, as highlighted in previous studies [3, 18]. Thus, this hinders the evaluation and comparison of different categorization approaches.

Some researchers attempted to overcome this limitation by using Google Play categories as a basis for evaluating their approaches [28, 38]. However, both previous studies and our own research (Section 3.2), demonstrate that apps within the same Google Play category exhibit only a broad sense of similarity, with significant variations in granularity observed across different categories. [4, 22, 32, 55]. For example, the HOUSE_AND_HOME category on Google Play encompasses apps for buying/renting houses and controlling Smart Home IoT devices. Although both relate to "House," their purposes differ, rendering this categorization unsuitable as a ground-truth.

Therefore, a reasonable and up-to-date alternative to evaluating a categorization approach is to create a dataset that relies on human judgment to manually categorize the apps. Previous researchers have evaluated their approaches on datasets that were manually crafted but often limited in either the number of apps, like Subaihin et al. [3] which only manually analyzed 300 apps in their research, or in the number of categories, with Ebrahimi et al. [18] testing only two categories. In contrast, our paper addresses this limitation by developing AndroCatSet: the first ground-truth dataset for app categorization, which comprises 5000 apps categorized into 50 classes. To prevent confusion with Google Play's "categories," we use the term "classes" to define clusters of apps with shared purposes and functionalities (e.g., calculators, navigation apps, weather apps, etc.).

Leveraging our new ground truth AndroCatSet, we were able to confirm, at first, the observations made in prior studies concerning the inefficiency of the Google Play Store categorization schema [4, 22, 32, 55] and secondarily, compare the effectiveness of current categorization approaches. Our comparison of existing approaches has revealed the remarkable superiority of categorization methods that leverage app descriptions compared to those that rely solely on data extracted from the APK file, such as method names or strings used by the app. However, despite the success of description-based approaches, our evaluation also revealed that there is still room for improvement within these methods. Therefore, we developed two innovative approaches: one description-based and the other APK-based, and thoroughly evaluated them using AndroCatSet, providing a comparison with existing methodologies. The results of our evaluation demonstrated a significant improvement in performance for both the description-based and APK-based methodologies, as compared to the existing approaches.

One particularly noteworthy approach is our new description-based method, which we named G-CatA: ***G**pt-based **CAT**egorization of **A**ndroid apps*, as it relies on Gpt-based models developed by OpenAI[1] (as we will describe in detail in Section 5.2). When using G-CatA, the app *SlideIT Free Keyboard*, previously discussed, can be appropriately categorized alongside other keyboard apps. This correction addresses the error made by CHABADA, which erroneously categorized it under language apps, thus highlighting how G-CatA can benefit tools relying on automatic categorization like CHABADA. To prove this point further, we conducted an additional experiment where we implemented the complete CHABADA framework, incorporating both the original and G-CatA approaches to categorize apps. Results indicate that our approach G-CatA exhibited greater precision in detecting anomalies compared to the original CHABADA approach.

The main contributions of our work are as follows:

- We release AndroCatSet: the first ground-truth dataset for evaluating Android app categorization approaches.
- We show that apps belonging to the same Google Play category exhibit only a general sense of similarity, confirming previous studies on the subject.
- We conduct a comparative analysis of existing categorization approaches, revealing that description-based approaches outperform APK-based ones.
- We propose two novel approaches, one description-based (G-CatA) and one APK-based, that improve the performance of existing methods.
- We demonstrate that our new description-based approach G-CatA can offer significant benefits to tools that depend on app categorization, such as CHABADA.

Data Availability. Our ground-truth dataset AndroCatSet, our new approach G-CatA, and all our artifacts are available at:

https:/github.com/Trustworthy-Software/Revisiting-Android-App-Categorization

2 BACKGROUND

In this section, we introduce terms and concepts used throughout our paper. First, we describe the clustering evaluation metric we relied on. Second, we provide an overview of text-embedding models, as they are employed in both existing categorization approaches and the new ones we are proposing.

Clustering Evaluation Metrics. Clustering evaluation metrics serve to assess the performance of clustering algorithms, and they can be categorized into two primary groups: intrinsic and extrinsic metrics [5]. Intrinsic metrics assess cluster quality and cohesion, focusing on the internal structure of the clusters, without relying on external information or ground truth. Examples of intrinsic metrics include the Silhouette Coefficient [48], Davies-Bouldin Index [15], and Calinski-Harabasz Index [12]. On the other side, extrinsic metrics are used to measure the similarity between two data clusterings, where one of the clusterings may consist of a ground-truth made of known labels. Examples of extrinsic metrics include the Rand Index [47] and Fowlkes-Mallows Index [20].

In our paper, we primarily use the Adjusted Rand Index (ARI) [24] as our main metric. The ARI is an extrinsic metric that adjusts the Rand Index for chance and yields a value between -1 and 1. A value of 1 indicates perfect agreement between the two clusterings, 0 indicates random agreement, and -1 indicates complete dissimilarity between the two clusterings. More in detail, given a set of n elements, and two partitions (e.g., clusterings) of these elements, namely $X = \{X_1, X_2, \ldots, X_r\}$ and $Y = \{Y_1, Y_2, \ldots, Y_s\}$ the overlap between X and Y can be summarized in a contingency table $\left[n_{ij}\right]$, as shown in (1). Each entry n_{ij} denotes the number of objects in

[1] https://openai.com/

common between X_i and Y_j:$n_{ij} = |X_i \cap Y_j|$.

X^Y	Y_1	Y_2	$\cdots$	Y_s	sums
X_1	n_{11}	n_{12}	$\cdots$	n_{1s}	a_1
X_2	n_{21}	n_{22}	$\cdots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_r	n_{r1}	n_{r2}	$\cdots$	n_{rs}	a_r
sums	b_1	b_2	$\cdots$	b_s	

(1)

Once the contingency table is defined, the ARI is computed as follows:

$$ARI(X,Y) = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}} \quad (2)$$

where n_{ij}, a_i, b_j are values from the contingency table. For instance, in our paper, we consider X as the categorization approach that we want to evaluate and Y as our ground-truth against which we compare the approach.

Text Embedding. Text embedding models are an advanced approach to convert textual information into numerical representations. These models, such as Word2Vec [33], FastText [9], GloVe [45], and BERT [17], capture the semantic and syntactic meaning of words, sentences, or entire documents by mapping them to dense vector spaces. While Word2Vec, FastText, and GloVe primarily focus on word-level embeddings, transformer-based models such as BERT can handle not only individual words but also whole sentences or documents. Moreover, OpenAI, the company behind ChatGPT, has released their own text embedding models [39], which are readily accessible through their official API. They can be leveraged for numerous applications, including search, clustering, recommendations, anomaly detection, and classification, as stated in the official documentation [43]. The text embedding models are based on pre-trained GPT (Generative Pre-trained Transformer) models [36], which are a class of transformer-based models developed by OpenAI [11]. These models, such as GPT-3 or the most recent GPT-4 [40], are built upon the powerful transformer architecture and have achieved significant breakthroughs across various NLP tasks [11].

3 GROUND-TRUTH DATASET

Existing studies have brought attention to the lack of a ground-truth dataset for assessing categorization approaches [3, 18]. Furthermore, additional studies have pointed out that utilizing Google Play categories for such evaluations is unsuitable since apps within the same Google Play category demonstrate only a broad sense of similarity, with significant variations in granularity across the different categories [4, 22, 32, 55].

To bridge this gap, we developed AndroCatSet: the first ground-truth dataset, specifically designed to evaluate categorization approaches, comprising 5000 apps distributed across 50 classes, with 100 apps for each class. Unlike Google Play's utilization of the term "category," we have opted for the term "class" to denote a cluster of applications that share a common purpose and similar functionalities. Some examples are calculator apps, navigation apps, weather apps, and more. Throughout this paper, we will consistently use the term "classes" to refer to the distinct groups of apps within our ground-truth dataset. This choice is deliberate, aimed at distinguishing them from the Google Play "categories," despite their similar meaning.

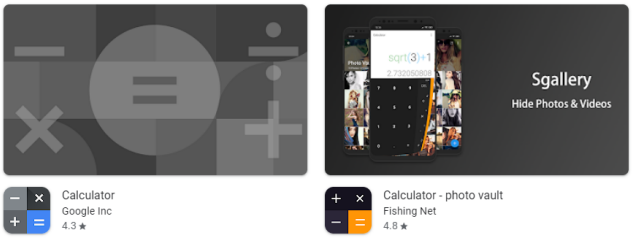


Figure 1: Searching for a calculator app on Google Play often leads to a mix of genuine and fake ones.

In our dataset, we have collected information for each app, including its package name, the Google Play Category ID (which uniquely identifies the app's category), and the app's description. Table 1 provides a comprehensive list of all defined classes in the dataset. On average, the apps have an APK size of around 20.38 MB, with a standard deviation of 18.19, indicating variations in size across the apps. The average length of app descriptions is approximately 2083 characters, with a standard deviation of 1087, suggesting differences in the level of detail provided (bearing in mind Google Play's 4000-character limit for descriptions). For more details, a table showing the average APK size and average description length for each class can be found in the repository.

In Section 3.1, we outline, step by step, the process of constructing our new ground-truth dataset. Then, in Section 3.2, we provide some interesting insights about the (non)alignment between app categorization in the Google Play Store and AndroCatSet.

3.1 Dataset Creation

Due to the need for high-quality data, relying on a fully automated process for creating the dataset was not feasible. Indeed, simply searching on Google Play for apps with the desired purpose, e.g., searching for "calculator", would have been inadequate due to the presence of apps intentionally disguising themselves as something else. For example, conducting a search for "calculator" on Google Play would yield numerous apps that appear to be regular calculators but actually allow users to hide their photos within them, as illustrated in Figure 1. This highlights the importance of implementing a manual verification process during the dataset construction phase, as each app requires individual manual checking.

The dataset was created according to the following steps, which were repeated for each one of the 50 classes:

(1) **Class Definition.** We define a class, e.g. "calculator", trying to be as specific as possible to ensure fine granularity with respect to Google Play Categorization. Some examples of classes include weather apps, translator apps, dating apps, and more. We intentionally decided not to define any classes related to games because their functionalities tend to vary too much, requiring excessive manual work to build our ground truth.

Table 1: The 50 classes present in our ground-truth dataset.

Ground-Truth Classes				
Airlines	Alarm	Antivirus	Astrology	Banking
BarcodeAndQRcodeScanner	BikeAndScooterSharing	BooksReader	Browser	BuyAndRentHome
Calculator	Calendar	CarBuying	Dating	Dialer
Drawing	Email	ExpenseTracker	FileManager	FoodDelivery
FoodDiary	HikingAndTrekking	Insurance	Investment	JobSearch
Keyboard	LanguageLearning	Launcher	Messenger	MusicProduction
Navigator	News	Notepad	OnlineTravelAgency	PhotoEditor
PromoAndDeals	PublicTransit	Radio	Recipes	RemoteControl
Shopping	SmartHome	Streaming	Translator	TravelGuide
VideoPlayer	Vpn	Wallpaper	Weather	WorkoutAndTraining

(2) **Apps Collection.** We use a custom Python script leveraging the *google-play-scraper*[2] to gather apps from Google Play by searching for keywords related to the class we want to populate. As *google-play-scraper* restricts the maximum number of apps returned to 30, it is advisable to search for multiple keywords for each class to gather a more comprehensive set of apps. For instance, in the case of the "Notepad" class, we conduct searches for both "notebook" and "notepad". Additionally, to overcome this limitation, we leverage two parameters of *google-play-scraper*: `country` and `lang`. The `country` parameter represents the two-letter country code (defaulted to "us") used for application retrieval, while the `lang` parameter denotes the two-letter language code (defaulted to "en") to be used. By modifying the `country` parameter while keeping `lang` fixed at "en", we retrieve more applications for each class, ensuring consistency with only English descriptions.
(3) **Filtering out malicious apps.** To ensure that each app in our class functions exactly as intended, we carefully exclude any possible malicious behavior that could alter its functionality and misalign the app with the rest of the class. To accomplish this, we search for the package name of each app in the AndroZoo dataset [2] [1] and keep only apps with a VirusTotal [56] score equal to zero, i.e., benign apps.
(4) **Manual Verification.** We then conduct a manual verification process for each app that remains after the previous steps, to determine whether it qualifies as a member of the respective class. This involves a thorough examination of the Google Play page, including the app's name, description, screenshots, and user reviews. In instances of uncertainty, we took an extra step by installing the app ourselves and personally verifying its functionalities. We stop the manual verification process once we reach a total of 100 apps for each class.
(5) **Download and Information Retrieving.** We complete the process by downloading all the apps from AndroZoo [2] [1]. Moreover, we save in a file the package name, the Google Play Category ID, and the description of the app, previously retrieved using *google-play-scraper*.

3.2 Dataset Insights.

After creating the AndroCatSet ground-truth, we conducted an analysis of the categories assigned by the Google Play Store to the apps included in it. Now, we present the insights yielded by this analysis regarding the granularity level of Google Play's categorization as well as instances of miscategorized apps that were revealed.

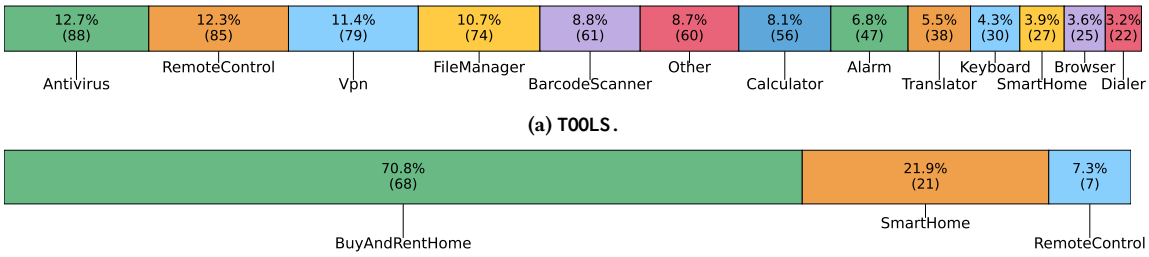
Granularity Level. In Figure 2, we focus on the Google Play categories `TOOLS` and `HOUSE_AND_HOME`. For each category, we provide the count of apps assigned to that specific category, segmented into the classes they belong to within AndroCatSet. Regarding the `TOOLS` category (Fig. 2a), our ground-truth includes a total of 690 apps distributed among 28 distinct classes. To enhance visual clarity, we have grouped classes that constitute less than 3% of the total under the "Other" class.

We can clearly see a contrast in the level of granularity between the two Play Store categories, emphasizing that apps within the same category only share a general sense of similarity, as noted in previous studies [4, 32]. The Google Play Category `TOOL` serves as a perfect illustration of this phenomenon, as it contains numerous apps that, despite falling under the umbrella of "tools", exhibit significant variations in their functionality and similarity. Furthermore, even within the seemingly more specific category `HOUSE_AND_HOME` there exist apps with vastly different purposes. For instance, there are apps available for purchasing and renting houses, as well as apps designed for managing Smart Home IoT Devices. This emphasizes the inadequacy of using the categorization system provided by the Google Play Store, especially in anomaly detection approaches. Indeed, in order to get better results, it is preferable to employ custom categorization approaches, such as the ones relying on the description as already observed by Gorla et al. [22].

Miscategorization. Another issue regarding the current categorization system of the Google Play Store is the presence of miscategorized apps, as previously highlighted by Surian et al. [55]. Upon reviewing the apps within our ground-truth data under the Google Play Category `TOOLS`, we have identified a clear instance of a miscategorized app in the Google Play Store. The app in question, *Daily Weather*[3], functions as a weather forecast app. In our ground-truth, we appropriately manually assigned it to the "weather" class. However, the app is currently listed under the Google Play Category `TOOLS`, which indicates a case of miscategorization since the category `WEATHER` already exists in the Google Play Store. For instance, 98% of the apps classified as "weather" in AndroCatSet are appropriately categorized under the Google Play category `WEATHER`.

[2]https://github.com/JoMingyu/google-play-scraper

[3]https://play.google.com/store/apps/details?id=com.rlk.weathers



(a) TOOLS.

(b) HOUSE_AND_HOME.

Figure 2: Number of apps assigned to a Google Play Store category, segmented into the classes they belong to within our dataset.

4 COMPARISON OF EXISTING APPROACHES.

In this section, our objective is to compare various existing categorization approaches and address the following research question:

RQ1: What is the performance of the existing categorization approaches on AndroCatSet?

In Section 4.1, we outline our systematic literature search methodology to gather existing categorization approaches from the literature. Section 4.2 elaborates on our approach selection criteria, explaining how we identified the approaches to be evaluated. Finally, in Section 4.3, we present the evaluation results to answer RQ1.

4.1 Systematic Literature Search

We conducted a systematic literature search to explore the various existing methodologies employed for categorizing Android Apps. Our goal is to systematically retrieve existing approaches in a structured manner to evaluate them on our ground-truth dataset. We conducted our search by exploring three widely used paper repositories: ACM Digital Library [29], IEEE Xplore [25], and Google Scholar [50]. To ensure comprehensive coverage, we formulated a set of six queries. These queries comprised terms connected by the logical operator "AND," mandating the presence of both terms and terms enclosed in quotation marks, specifying an exact match. The following are the queries we utilized:

- android AND "apps categorization"
- android AND "apps clustering"
- "android app" AND "topic modeling"
- "android app" AND "description analysis"
- "android app" AND "behavior classification"
- "android app" AND "miscategorization"

Results. In Table 2, we present the outcomes of our literature search. The *Hits* column indicates the total number of papers gathered during the process. We then proceeded to eliminate irrelevant publications. *Title* represents the number of papers after removing publications that are clearly irrelevant based on the title. *Abstract* indicates the papers remaining after inspecting the abstracts and removing the not relevant papers. In the final row of the table, we present the overall count of papers corresponding to each step. Finally, we eliminated duplicate papers identified by multiple queries, obtaining a collection of 40 distinct papers.

Literature Search Insights. In our retrieved collection of papers, we have come across surveys that provide valuable insights into multiple categorization approaches simultaneously [32, 51].

Table 2: Paper filtered for each step of the literature search.

Query	Hits	Title	Abstract
android AND "apps categorization"	170	46	22
android AND "app clustering"	69	26	14
"android app" AND "topic modelling"	236	45	17
"android app" AND "description analysis"	62	19	12
"android app" AND "behavior classification"	88	14	3
"android app" AND "miscategorization"	34	12	2
Total	659	162	66
Removing Duplicates			**40**

We have discovered numerous papers that utilize app categorization for a variety of purposes, including but not limited to malware detection [22, 60], finding miscategorized apps [6, 55], and even classifying malware into distinct families [28, 35, 37]. As mentioned earlier in Section 1, these findings reinforce the significance and practicality of app categorization.

Within the collection of papers we have gathered, two major trends can be identified. The first trend centers around the absence of a reliable ground-truth for evaluating app categorization approaches. Numerous papers rely on the Google Play categories [23, 28, 38], but as previous studies have indicated and as we discussed in Section 3.2, these categories are overly generalized to be considered accurate [4, 32]. Additionally, some papers compare their approaches using the same dataset used by others, which hampers the research community's ability to gain a comprehensive overview based on a consistent dataset. For instance, Shamsujjoha et al. [52] attempted to compare their approach, REACT, against CHABADA [22], but they were only able to retrieve 75% of the apps utilized in the study by Gorla et al. [22]. Others have attempted to overcome the lack of a ground-truth by relying on human judgment, but the inherent time constraints associated with this method only allow for relatively small datasets to be manually verified. For example, Ebrahimi et al. [18] conducted a study involving 600 apps divided into two categories, whereas Al-Subaihin et al. [4] manually evaluated 300 apps in their research. The second trend pertains to the reliance of several papers on a supervised approach, often utilizing Google Play categories as labels [19, 34, 49, 58]. As a result, the number of unsupervised approaches employed for app categorization is significantly reduced.

4.2 Approaches Selection.

To identify existing categorization approaches to be evaluated on our ground-truth dataset, we began by re-filtering the 40 distinct papers related to app categorization. Out of the initial 40 papers, only 28 propose a specific approach for app categorization, while the remaining papers consist of reviews discussing multiple approaches or papers relying directly on the categorization already provided by Google Play. Among the 28 remaining papers, only 9 included available code for implementation. However, the links provided for 3 out of the 9 papers were broken [13, 35, 55]. Despite making several attempts to contact the authors, we either received no response, or communication ceased after a few emails. Moreover, we tried reaching out to some of the authors of papers that initially did not share their code, but unfortunately, these attempts were also unsuccessful. For our last step, we excluded one [27] of the 6 remaining papers since its authors overlapped with another paper and utilized the same approach (the provided link led to the same web page[57]). In the repository, a table showcasing the code availability, utilized features, and reasons for exclusion for each of the 40 papers can be found, providing more in-depth information.

Eventually, the five approaches selected are the following:

- **CHABADA by Gorla et.al [22].** The objective of CHABADA is to identify applications that do not align with their descriptions. To categorize the apps, they follow a two-step process: clustering the apps using LDA to extract topics from descriptions, and applying K-means for further clustering.
- **REACT by Shamsujjoha et al. [52].** In this paper, the authors present REACT as an alternative approach to CHABADA for scenarios where descriptions are unavailable. Their proposed method follows the same strategy as CHABADA but leverages different sources of information, such as method names, XML data, and GUI text values.
- **Ebrahimi et al. [18].** The authors conducted a comparison of various word embedding models to create numerical semantic representations of app descriptions. These representations were then utilized for the classification of app categories. As our paper primarily focuses on the unsupervised categorization of apps, we adopted the initial part of their methodology and applied K-Means clustering on the embeddings to compare it with other approaches. Out of the word embedding models utilized by the authors, we specifically chose GloVe [45], as they demonstrated that it yielded the most favorable outcomes.
- **Sun et al. [54]** The authors introduced a novel mobile app clustering scheme that utilizes various features extracted from the APK file. These features include activity names, certificate issuer information, and sets of keywords. By employing Affinity Propagation [21], they demonstrated that clustering based on the similarity of keywords, which represent app functionality, yields superior performance. Hence, for our evaluation, we exclusively focused on the keyword set of strings.
- **Yang et al. [60]** In this paper, the authors introduce a novel method for characterizing malicious apps by analyzing their descriptions and data-flow information. They employ LDA to extract topics from app descriptions. However, unlike CHABADA [22], they directly cluster apps based on the most relevant topic.

4.3 RQ1 Results.

To assess the performance of the selected approaches, we rely on the ARI Score, which we extensively discussed in Section 2. The ARI Score is a straightforward metric that provides a measure of how well the categorization approach aligns with a ground truth. A score closer to 1 indicates a higher overlap with the ground truth and, therefore, better performance, while a score closer to 0 suggests random clustering and poorer performance.

In Figure 3, we present the ARI Score for each of the five approaches, showing a clear and significant distinction among the evaluated methods. The approaches that rely on the description (CHABADA and the ones proposed by Ebrahimi et al. and Yang et al.) demonstrate high performance, while the other two approaches (REACT and the one proposed by Sun et al.) exhibit poor performance when evaluated against on AndroCatSet. Both of the poorly performing approaches do not rely on the description. Instead, they extract data from the APK file of an app, including XML values, Method Names, GUI Text for REACT, and Strings used by the app for the approach from Sun et al. An explanation for this behavior has already been proposed by the authors of REACT in their paper [52]. They observed that in Android apps, method names and XML data values can be influenced by data obfuscation and encryption. Moreover, as demonstrated in their paper [52], it is possible for two distinctly different types of Android apps to have a significant overlap in terms of XML data values.

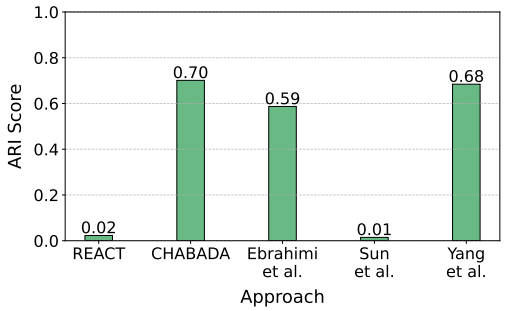


Figure 3: ARI Scores of existing categorization approaches.

Answer to RQ1: *We evaluated five existing approaches using AndroCatSet. The approaches relying on app descriptions show relatively high performance, while those extracting data from APK files perform poorly.*

5 IMPROVEMENTS OF DESCRIPTION-BASED CATEGORIZATION APPROACHES

RQ1 revealed the superiority of description-based approaches but also acknowledged the potential for further improvement within this methodology. Thus, in this section, we explore possible advancements in description-based approaches. In Section 5.1, we introduce G-CataA: our innovative method that leverages app descriptions. Then, in Section 5.2, we evaluate our own approach, addressing the following research question:

RQ2: Can the performance of app categorization approaches leveraging app descriptions be improved?

5.1 Novel Description-based Approach.

We propose our approach G-CatA, which stands for ***G**pt-based **CAT**egorization of **A**ndroid apps*. The main idea behind our new description-based approach is to leverage OpenAI's powerful GPT-based text embedding models [39], which have been introduced in Section 2, to effectively process and represent app descriptions, enabling automatic categorization of the apps.

Prior to generating the embeddings, we leverage standard NLP techniques to preprocess the app descriptions, following the approach adopted by similar studies [4, 22, 55]. Our preprocessing steps involve removing non-textual items, eliminating stop-words (e.g., "the," "is," "at," etc.), and performing lemmatization. Lemmatization, unlike stemming, considers the grammatical context and aims to produce meaningful and valid base forms (e.g., "caring" to "care" and not "car" like stemming).

After completing the preprocessing stage, we use the OpenAI second-generation model `text-embedding-ada-002` [42]. This new embedding model significantly outperforms first-generation models initially introduced by OpenAI in various NLP tasks, such as natural language processing and code tasks, as highlighted in OpenAI's official blog post announcement [42]. This model employs the `cl100k_base` tokenizer, which is the same tokenizer used in ChatGPT3.5 and ChatGPT4 [41]. It allows for a maximum input context length of 8192. Notably, the embeddings generated by this model have 1536 dimensions, and their cost is reduced by 90% compared to first-generation models [42].

Finally, we use the Scikit-learn [44] implementation of the K-Means [30] clustering algorithm as the final step to partition the apps into 50 clusters.

5.2 RQ2 Results.

To address RQ2, we compared our new approach G-CatA against existing description-based approaches that were already evaluated in Section 4.3. In Figure 4, we present a comparison of ARI scores among four approaches: G-CatA (shown in red) and three existing approaches previously evaluated in RQ1 (shown in green). G-CatA outperforms the other three, achieving an impressive ARI score of 0.91, which represents a remarkable 32% improvement compared to the previous leading method, CHABADA. Notably, the achieved ARI score is remarkably close to 1, indicating a nearly perfect alignment with the ground truth. This result underscores the effectiveness of combining descriptions with powerful models like OpenAI's text embedding models.

Answer to RQ2: *We have proposed G-CatA, a novel approach for categorizing Android apps by leveraging app descriptions and OpenAI's powerful embedding models. G-CatA achieved an impressive ARI score of* 0.91*, outperforming existing description-based approaches.*

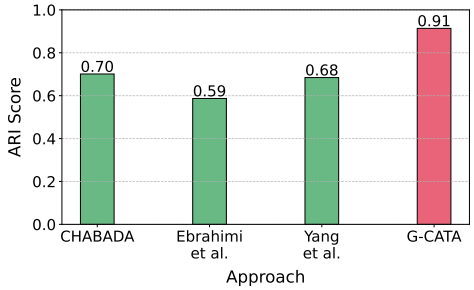


Figure 4: ARI scores of our new approach G-CatA (red) compared to existing description-based approaches (green).

6 IMPROVEMENTS OF APK-BASED CATEGORIZATION APPROACHES

RQ1 demonstrated the inadequate performance of categorization approaches relying on APK file data. However, app descriptions may not always be accessible or undergo changes [52], emphasizing the need to categorize apps using only data within the APK. In this section, we explore advancements in APK-based approaches and propose two alternative solutions, both centered around the concept of expanding the set of features used in conventional APK-based methods. These solutions are now presented individually in Section 6.1 and Section 6.2.

6.1 Leveraging Existing App Representations from Unrelated Tasks.

One simple idea is to leverage existing tools from other unrelated tasks, such as malware detection, to gather diverse app representations (i.e., representations that differ from the one presented in RQ1). These diverse representations can then be used for our categorization task. Let's consider the example of DREBIN [7], one of the most popular Android malware detectors. Instead of using it to distinguish between benign and malicious apps, we can repurpose the same app representation for categorizing apps. This approach requires minimal implementation effort since we can utilize existing features from the malware detection process.

Thus, we aim to answer the following question:

RQ3: How do different approaches for app representation, borrowed from unrelated tasks, impact the performance of app categorization?

To address this RQ, we investigated various app representations proposed in the research literature, aiming to diversify the types of representations explored. In particular, we investigate the following four types of representations that can be generated by available tools.

- **Feature-based app representation.** For the "traditional" feature-based representation, we employ DREBIN by Arp et al. [7], which performs extensive static analysis by extracting various application features. These features are then organized in a unified vector space to distinguish between benign and malicious apps. Specifically, DREBIN employs eight distinct feature sets, namely:

Hardware features, Requested permissions, App components, Filtered intents, Restricted API calls, Used permissions, Suspicious API calls, and Network addresses.

- **Image-based app representation.** The image-based representation is obtained using DexRay by Daoudi et al. [14]. DexRay takes the bytecode from the app's DEX files and transforms them into grayscale images. These images are then utilized by a 1D Convolutional Neural Network (CNN) model to identify malware.
- **Icon-based app representation.** To derive this representation, we implement the method proposed by Rajasegaran et al. [46]. They introduced a novel icon encoding technique that efficiently detects potential counterfeits for a specific app. This method leverages neural embeddings from CNNs, enhancing search accuracy.
- **BERT-based app representation.** To obtain this representation, we adopt LaFiCMIL by Sun et al. [53]. The proposed approach, LaFiCMIL, offers a versatile framework that can be effectively applied to diverse BERT-based large file classification tasks, including Android Malware Detection.

RQ3 Results. In Table 3, we present the ARI scores for the four app representations we previously selected. All four approaches exhibit scores close to zero, which indicates clustering results similar to random chance. Notably, these outcomes align with the two APK-based approaches we discussed in RQ1. Despite our efforts to expand our approaches by leveraging various representations based on diverse features such as app icons, app components, permissions, and bytecode image representations, we were unable to achieve improved performance. However, it is essential to note that we relied on existing tools to extract the app representations from the apps in our dataset, without making any modifications to the feature extraction process or the employed embedding techniques.

Table 3: ARI scores using various app representations.

Approach	Representation	ARI Score
DREBIN	Features-based	0.02
DexRay	Image-based	0.01
Rajasegaran et al.	Icon-based	0.02
LaFiCMIL	BERT-based	0.01

Answer to RQ3: *We leveraged various app representations from diverse approaches unrelated to app categorization, e.g., malware detection. Our observations indicate that these representations do not enhance the categorization performance.*

6.2 Novel APK-based Approach.

The second option involves creating a new approach from scratch, independent of existing tools, while drawing inspiration from valuable insights offered by the aforementioned app representations used in other, unrelated tasks. In this case, our research question is as follows:

RQ4: Which specific types of features extracted from the APK file, demonstrate the most significant impact on app categorization performance?

To answer this question, we first compiled a list of data that can be extracted from an Android APK file and is consistently available, unlike metadata such as the description. Our selected data includes:

(1) **Name**: The name/title of the application.
(2) **Permissions**: The permissions requested by the app.
(3) **Restricted API Calls**: The API calls protected by permissions.
(4) **Strings**: The textual content used within the app.
(5) **Icon**: The graphical representation of the app.
(6) **Libraries**: The list of libraries utilized by the app.

We have defined this specific list of data with the intention of characterizing apps in a similar manner as the authors of REACT [52] and also taking inspiration from the existing tools that we presented in Section 6.1. This data is expected to assist us in identifying commonalities among apps with similar purposes. For example, when comparing two "calculator" apps, we expect to find similarities in their app names, rely on a similar set of permissions, and the presence of shared strings within the apps. These shared strings could include terms related to mathematics, such as "calc", "plus", "minus", and similar expressions.

To extract the desired data from the app within our dataset, we have developed a set of custom Python scripts that automate the usage of tools such as Androguard [16] and ApkTool [59]. Our repository also includes these valuable scripts, which are readily available for others to utilize. Androguard, a Python-based software developed for analyzing Android applications, was used to extract essential information, including the app's name, icon, and other pertinent details. In some specific instances, we leveraged ApkTool for reverse engineering Android apps and then parsing XML and DEX files to extract the necessary data.

To handle the diverse nature of the collected information, we employed various techniques to generate numerical feature vectors from the extracted data. When dealing with the names of the apps, we utilized the text embedding models from OpenAI, which were also employed for the descriptions in Section 5.1. However, we encountered a limitation with the OpenAI text embedding model's maximum input length, preventing us from generating embeddings for other types of features. For the app icon, we adopted the same approach of Rajasegaran et al. [46](already presented in Section 6.1). However, in contrast to their method, we only embedded the content of the app icon, as we believe this provides more value for categorization purposes. All the remaining data can be conveniently represented as a list of strings. This suggests how a simple TF-IDF vectorization can effectively measure the significance of individual strings, such as permission names or library usage.

Finally, following the approach we employed for the descriptions, we leveraged the K-Means clustering algorithm as the final step to divide the apps into 50 clusters for each of the six types of data we proposed. This allowed us to assess the effectiveness of these features in categorizing the apps on an individual basis.

RQ4 Results. Figure 5 illustrates a comparison of ARI Scores for various feature types used in our categorization, represented in red, contrasting with the two existing APK-based methods evaluated in RQ1 shown in green. As we can observe, using the app icon, the used libraries, and restricted API calls yield results comparable to the two existing APK-based approaches. Instead, notable performance improvements can be achieved when leveraging the app strings,

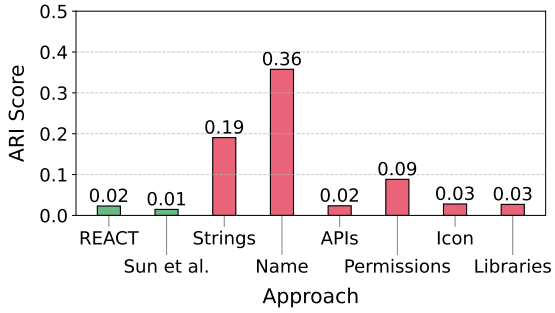


Figure 5: ARI scores of our new categorization approach (red) compared to existing APK-based approaches (green).

app name, and the requested permissions. Indeed, even if the ARI score obtained through the App Name is still distant from the scores obtained with the description-based approaches, it represents an impressive increase of 1700% compared to the previous leading APK-based approach, REACT. The remarkable improvement observed can likely be attributed to the utilization of OpenAI's powerful text embedding models, as already discussed for RQ2.

Another key factor worth highlighting is the remarkable performance improvement achieved by leveraging the strings used by the apps, in contrast to the existing APK-based methods that rely on similar characteristics. This emphasizes the substantial influence of embedding techniques and the clustering algorithm on task performance, even when utilizing the same feature set.

Combining multiple features extracted from the APK file. After evaluating the various types of features individually, as shown in Figure 5, and observing their positive impacts, we attempted to further enhance the overall performance by combining all the features simultaneously. To achieve this, we concatenated all the previously generated feature vectors and applied normalization using the `MinMaxScaler` from Scikit-learn [44]. Next, we employed Principal Component Analysis (PCA) [26] for dimensionality reduction and subsequently utilized the K-Means algorithm to cluster the apps once more. Unfortunately, our efforts did not produce the expected improvement. The ARI score obtained was 0.14 when using all six feature types and 0.30 when using only the top three performing ones. These scores did not surpass the best score achieved using only the app name, thus, we decided not to include this score in Figure 5. This outcome implies that the different feature types may be capturing similar underlying relationships among the apps. For future research, exploring alternative methods to leverage all features, such as employing ensemble clustering techniques [10], could be beneficial.

Answer to RQ4: *We tested various APK-based features for app categorization and discovered that combining the app name with OpenAI's model achieved the best performance. Our approach outperformed the leading APK-based method achieving a score of 0.36.*

7 DOWNSTREAM TASKS IMPROVEMENT.

In Section 4.3, we demonstrated the superiority of our new approach G-CatA over existing methods. Building on that, our focus shifts toward examining the potential impact of an enhanced categorization approach on the overall performance of tools reliant on automatic app categorization. We seek to address the following RQ:

RQ5: How does the performance of tools relying on app categorization improve with a better categorization approach?

In particular, Section 7.1 details the experiments we conducted using G-CatA on top of CHABADA. Then, in Section 7.2, we provide the findings and answer RQ5.

7.1 Testing CHABADA as Malware Detector.

CHABADA by Gorla et al. [22] can be used to find potential malicious apps by identifying inconsistencies between the exhibited behavior and the app descriptions. It consists of two phases: app categorization, which involves utilizing app descriptions, and Latent Dirichlet Allocation (LDA), followed by unsupervised anomaly detection using the One-Class SVM algorithm. This process helps identify outliers based on API usage patterns.

We evaluated the effectiveness of CHABADA in detecting potential malicious apps when employing two different categorization approaches. The objective was to understand how these approaches influence the anomaly detection phase of CHABADA. We trained OC-SVM models on benign apps and used them as classifiers on a test set containing both benign apps and known malware apps, following the methodology of the original paper [22]. However, we first relied on the original categorization approach based on LDA (i.e., the one presented in [18]) before relaunching all the experiments using our novel approach G-CatA. Below, we provide comprehensive details about the experiments conducted.

(1) **Dataset.** We started by organizing AndroCatSet into a Training Set and a Test Set. The Training Set consisted of 4500 apps, with 90 apps representing each of the 50 classes, with the Test Set comprising the remaining 500 apps. Since our AndroCatSet consists solely of benign apps, we expanded our Test Set by including 500 malicious apps sourced from the AndroZoo repository. We gathered apps that were flagged as malware by at least ten antivirus scanners in VirusTotal [56], and also those that were available on the Google Play Store, in order to easily access their descriptions.
(2) **Clustering.** Once we organized the dataset and retrieved the descriptions of all the apps, we proceeded to cluster the Training Set into 50 clusters using both the CHABADA original approach and G-CatA. While clustering the apps, we made sure to save the machine learning models for future reuse during the testing process.
(3) **Sensitive APIs Extraction.** For all the apps in both the Training and Test sets, we extracted the sensitive API calls, which are API calls that require Android permission settings for protection and can access sensitive information (e.g., camera, microphone, etc.) or perform sensitive tasks (e.g., altering system settings, sending messages, etc.). For example, the `getLastKnownLocation()` API call is protected by `ACCESS_FINE_LOCATION` permission.

Table 4: Performance of CHABADA as a malware detector: original approach vs. G-CATA ($F1 = \frac{TP}{TP+\frac{1}{2}(FP+FN)}$).

	TN Rate	FP Rate	FN Rate	TP Rate	F1
Chabada	86.40%	13.60%	80.40%	19.60%	0.29
G-CATA	**92.00%**	**8.00%**	**69.20%**	**30.80%**	**0.44**

To extract all sensitive API calls, including the number of call sites for each API, we used static analysis tools such as Androguard [16] and ApkTool [59].

(4) **Training.** CHABADA leverages sensitive APIs as binary features and employs a One-Class SVM (OC-SVM) to train distinct models for each cluster of similar applications. These models are specifically tailored for detecting anomalies or novelties, which, in our case, refer to applications whose API usage significantly deviates from the established norm within their respective clusters. As described in the CHABADA paper, we trained an OC-SVM for each cluster of benign apps found in the training set, aiming to learn their "normal" behavior. We repeated this step first by using the clusters produced using the original CHABADA approach and then by using the clusters produced by G-CATA (with a total of 50+50 models).

(5) **Testing.** Finally, we used the cluster-specific OC-SVM models as malware detectors. We assign each app in the Test Set to one of the 50 clusters using the models saved from the clustering phase. This enables us to select the appropriate OC-SVM model to be used as an anomaly detector. Then, we verify whether the malicious apps are identified as anomalies, indicating differences in their APIs compared to the typical usage of the API within the same cluster. On the contrary, we expect the benign apps in the Test Set not to be flagged as anomalies i.e., potentially malicious, since their behavior should align with the common behavior of the specific cluster.

7.2 RQ5 Results.

The primary goal of RQ5 is to explore the potential advantages of G-CATA when applied to tools reliant on automated app categorization, such as CHABADA. The results of our experiments are presented in Table 4. As in the original CHABADA paper, we consider a malicious app detected as an anomaly as a True Positive, and a benign app not flagged as an anomaly as a True Negative [22]. The remaining definition of False Positive and False Negative, come easily. G-CATA demonstrates improvements in both the False Positive Rate and the True Positive Rate. In the case of the False Positive Rate, we observed a slight reduction from 13.60% to 8.00%, indicating fewer benign apps being incorrectly identified as malicious. On the other hand, there has been a significant improvement in the True Positive Rate, which increased from 19.60% to 30.80%. This increase of 57.14% clearly demonstrates the impact of G-CATA in identifying a larger number of malicious apps, when CHABADA relies on it. In terms of overall performance, the F1 Score has increased from 0.29 to 0.44, confirming the positive impact of G-CATA, as just mentioned. Moreover, it is important to note that the OC-SVM models, which do not have any prior knowledge about malicious apps, can also be used to detect unknown malware, making the results even more remarkable.

Answer to RQ5: *We demonstrated that relying on a better categorization approach, such as G-CATA, can yield a substantial influence on the final outcome of tools that depend on categorization, such as CHABADA.*

8 LIMITATIONS AND THREATS TO VALIDITY.

Like any other study, our research is susceptible to threats to validity, which arise from various limitations in our approach. Below, we outline the most significant threats and limitations.

Human error and subjectivity. Constructing the first ground-truth dataset from scratch posed challenges as we could only rely on manual verification to ensure high-quality data. It is important to note that despite following a consistent process for building ANDROCATSET, human subjectivity may still influence certain aspects. To mitigate this threat to validity we share ANDROCATSET, as well as the tools and scripts used to create it, with the research community.

Not exhaustive literature search. Our literature search for categorization approaches, based on the six queries we defined in 4.1, may not be exhaustive. However, as it serves as a systematic means to identify existing approaches in a structured manner, our decision was primarily driven by practical considerations. Indeed, including broader queries would have been impractical due to the overwhelming number of papers retrieved. For instance, the query "app clustering" returns over 200 000 results on Google Scholar, making it impractical to manually select the relevant approaches for evaluation.

Approaches selection and code availability. The selection of categorization approaches for evaluation, carried out in Section 4.2, was mainly influenced by code availability. The lack of code hindered a comprehensive performance overview, highlighting how much important it is to provide code alongside the paper to benefit the entire research community.

9 RELATED WORK

To the best of our knowledge, we are the first to provide a comprehensive ground-truth dataset for evaluating Android app categorization. However, we now present a concise summary of the primary findings derived from previous research conducted on this topic.

The previous research conducted by Al-Subaihin et al. [3] is highly relevant to our work as it presents an empirical comparison of various text-based app clustering techniques such as Latent Dirichlet Allocation (LDA), as well as keyword feature extraction methods. They experimented with 12 664 Google Play Store apps sampled from 24 categories. They utilized intrinsic measures such as the Silhouette Score and human judgment (analyzing 300 apps in total) due to the absence of a ground-truth, as outlined in their paper. Our work assumes even greater significance due to the creation of ANDROCATSET, which could have proven immensely valuable in evaluating studies like the one conducted by Al-Subaihin et al. [3]. Furthermore, in our paper, we not only assessed approaches that rely solely on app descriptions for categorization but also tried to distinguish between the features utilized by the evaluated methods. For instance, we took into consideration all types of data that can

be extracted from the APK in order to provide a comprehensive evaluation.

Martin et al. [32] conducted a comprehensive survey on App Store Analysis, which included a dedicated section on App Clustering approaches. In this section, they presented an overview of various techniques and features utilized in clustering apps highlighting how the app descriptions are commonly leveraged to categorize apps according to their functionality. In their review, about Android Security using NLP, Sen et al. [51] provide a concise overview of categorization approaches commonly employed in malware detection tasks, which typically rely on the descriptions of the apps. Although our literature search may not have been as comprehensive as the studies conducted by Martin et al. [32] and Sen et al. [51], we have gone beyond by not only offering an overview of existing approaches but also assessing their performance on AndroCatSet ground-truth.

Several research papers have consistently highlighted the inadequacy of Google Play's current app categorization system. Apps within the same category often exhibit only a vague sense of similarity, indicating a significant lack of effective categorization [4, 22, 32, 55]. In our paper, we conducted a comprehensive analysis of our own original dataset affirming the findings of the just cited prior studies. We even provided concrete evidence of a misclassified app, as defined by Surian et al. in their paper [55].

10 CONCLUSION

In our study, we conducted a comprehensive evaluation of various Android app categorization approaches found in the existing literature. Our analysis emphasized the remarkable superiority of approaches that utilize app descriptions, as opposed to those relying solely on data extracted from the APK file. This evaluation was made possible thanks to our new AndroCatSet ground-truth. We believe this dataset will be a valuable resource for future research in this field, addressing a gap that has been previously highlighted by similar studies [3, 18]. Furthermore, we developed two innovative approaches that effectively improve the performance of existing methods in both description-based and APK-based methodologies. Notably, for what concerns our description-based approach G-CatA, we achieved an impressive ARI score of 0.91 by leveraging the powerful text embedding models provided by OpenAI. In our final experiments, we demonstrated the impact of a better-performing categorization approach when implemented within a tool reliant on automatic app categorization, such as CHABADA [22]. This highlights how G-CatA can provide substantial benefits to future software engineering tools that rely on automatic categorization, emphasizing the importance of developing advanced and efficient app categorization methodologies.

Future work may involve expanding AndroCatSet by incorporating additional apps and classes, as well as intensifying efforts to address the existing disparity between APK-based and description-based approaches. By offering valuable insights, introducing a new ground-truth dataset, and presenting two innovative approaches to address existing limitations in the literature, our research aims to make a significant contribution to the field of automatic app categorization in software engineering.

11 DATA AVAILABILITY

The repository including all artifacts is available at:

https://github.com/Trustworthy-Software/Revisiting-Android-App-Categorization

12 ACKNOWLEDGEMENT

This research was funded in part by the Luxembourg National Research Fund (FNR), grant reference NCER22/IS/16570468/NCER-FT and REPROCESS grant reference C21/IS/16344458.

REFERENCES

[1] Marco Alecci, Pedro J. R. Jiménez, Kevin Allix, Tegawendé F. Bissyandé, and Jacques Klein. 2024. AndroZoo: A Retrospective with a Glimpse into the Future. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. IEEE Computer Society.

[2] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 468–471. https://doi.org/10.1145/2901739.2903508

[3] Afnan Alsubaihin, Federica Sarro, Sue Black, and Licia Capra. 2019. Empirical comparison of text-based mobile apps similarity measurement techniques. *Empirical Software Engineering* 24 (12 2019). https://doi.org/10.1007/s10664-019-09726-5

[4] Afnan Alsubaihin, Federica Sarro, Sue Black, L. Capra, Mark Harman, Yue Jia, and Y. Zhang. 2016. Clustering Mobile Apps Based on Mined Textual Features. 1–10. https://doi.org/10.1145/2961111.2962600

[5] Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. 2009. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval* 12, 4 (Aug 2009), 461–486. https://doi.org/10.1007/s10791-008-9066-8

[6] Azmi Aminordin, Mohd Faizal Abdollah, Robiah Yusof, and Rabiah Ahmad. 2018. Preliminary Findings: Revising Developer Guideline Using Word Frequency for Identifying Apps Miscategorization. In *Proceedings of the Second International Conference on the Future of ASEAN (ICoFA) 2017 – Volume 2*, Rizauddin Saian and Mohd Azwan Abbas (Eds.). Springer Singapore, Singapore, 123–131.

[7] Dan Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Network and Distributed System Security Symposium*.

[8] Vitalii Avdiienko, Konstantin Kuznetsov, Isabelle Rommelfanger, Andreas Rau, Alessandra Gorla, and Andreas Zeller. 2017. Detecting Behavior Anomalies in Graphical User Interfaces. In *Proceedings of the 39th International Conference on Software Engineering Companion* (Buenos Aires, Argentina) *(ICSE-C '17)*. IEEE Press, 201–203. https://doi.org/10.1109/ICSE-C.2017.130

[9] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146. https://doi.org/10.1162/tacl_a_00051

[10] Tossapon Boongoen and Natthakan Iam-On. 2018. Cluster ensembles: A survey of approaches with recent extensions and applications. *Computer Science Review* 28 (2018), 1–25. https://doi.org/10.1016/j.cosrev.2018.01.003

[11] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[12] T. Caliński and J Harabasz. 1974. A dendrite method for cluster analysis. *Communications in Statistics* 3, 1 (1974), 1–27. https://doi.org/10.1080/03610927408827101

[13] Ning Chen, Steven Hoi, Shaohua Li, and Xiaokui Xiao. 2015. SimApp: A Framework for Detecting Similar Mobile Applications by Online Kernel Learning. *WSDM 2015 - Proceedings of the 8th ACM International Conference on Web Search and Data Mining*. https://doi.org/10.1145/2684822.2685305

[14] Nadia Daoudi, Jordan Samhi, Abdoul Kader Kabore, Kevin Allix, Tegawendé F. Bissyandé, and Jacques Klein. 2021. 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*, Gang Wang, Arridhana Ciptadi, and Ali Ahmadzadeh (Eds.). Springer International Publishing, Cham, 81–106.

[15] David L. Davies and Donald W. Bouldin. 1979. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1, 2 (1979), 224–227. https://doi.org/10.1109/TPAMI.1979.4766909

[16] Anthony Desnos. 2011. https://github.com/androguard/androguard.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[18] Fahimeh Ebrahimi, Miroslav Tushev, and Anas Mahmoud. 2021. Classifying Mobile Applications Using Word Embeddings. *ACM Trans. Softw. Eng. Methodol.* 31, 2, Article 20 (nov 2021), 30 pages. https://doi.org/10.1145/3474827

[19] Wenhao Fan, Yeh g Chen, Yuan'an Liu, and Fan Wu. 2019. DroidARA: Android Application Automatic Categorization Based on API Relationship Analysis. *IEEE Access* 7 (2019), 157987–157996.

[20] E. B. Fowlkes and C. L. Mallows. 1983. A Method for Comparing Two Hierarchical Clusterings. *J. Amer. Statist. Assoc.* 78, 383 (1983), 553–569. https://doi.org/10.1080/01621459.1983.10478008

[21] Brendan J. Frey and Delbert Dueck. 2007. Clustering by Passing Messages Between Data Points. *Science* 315, 5814 (2007), 972–976. https://doi.org/10.1126/science.1136800 arXiv:https://www.science.org/doi/pdf/10.1126/science.1136800

[22] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking App Behavior Against App Descriptions. In *ICSE '14: Proceedings of the 2014 International Conference on Software Engineering* (Hyderabad, India). ACM Press, 292–302.

[23] Nils Gruschka, Luigi Lo Iacono, and Jan Tolsdorf. 2018. Classification of Android App Permissions.

[24] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of Classification* 2, 1 (Dec 1985), 193–218. https://doi.org/10.1007/BF01908075 Company: Springer Distributor: Springer Institution: Springer Label: Springer number: 1 publisher: Springer-Verlag.

[25] IEEEXplore. 2023. https://ieeexplore.ieee.org/Xplore/home.jsp.

[26] Ian T Jolliffe and Jorge Cadima. 2016. Principal component analysis: a review and recent developments. *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences* 374, 2065 (2016), 20150202.

[27] Konstantin Kuznetsov, Vitalii Avdiienko, Alessandra Gorla, and Andreas Zeller. 2016. Checking App User Interfaces against App Descriptions. In *Proceedings of the International Workshop on App Market Analytics* (Seattle, WA, USA) *(WAMA 2016)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/2993259.2993265

[28] Anran Li, Shuangshuang Xue, Xiang-Yang Li, Lan Zhang, and Jianwei Qian. 2022. AppDNA: Profiling App Behavior via Deep-Learning Function Call Graphs. *IEEE Transactions on Emerging Topics in Computing* 10, 1 (2022), 414–427. https://doi.org/10.1109/TETC.2020.3026335

[29] ACM Digital Library. 2023. https://dl.acm.org/.

[30] S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137. https://doi.org/10.1109/TIT.1982.1056489

[31] Arvind Mahindru and Amrit Sangal. 2021. SemiDroid: a behavioral malware detector based on unsupervised machine learning techniques using feature selection approaches. *International Journal of Machine Learning and Cybernetics* 12 (05 2021). https://doi.org/10.1007/s13042-020-01238-9

[32] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2017. A Survey of App Store Analysis for Software Engineering. *IEEE Transactions on Software Engineering* 43, 9 (2017), 817–847. https://doi.org/10.1109/TSE.2016.2630689

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger (Eds.), Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

[34] A. Mohammad Ebrahimi, M. Saber Gholami, Saeedeh Momtazi, M. R. Meybodi, and A. Abdollahzadeh Barforoush. 2020. Correlation Analysis of Applications' Features: A Case Study on Google Play. In *Data Science: From Research to Application*, Mahdi Bohlouli, Bahram Sadeghi Bigham, Zahra Narimani, Mahdi Vasighi, and Ebrahim Ansari (Eds.). Springer International Publishing, Cham, 202–216.

[35] Annamalai Narayanan, Charlie Soh, Lihui Chen, Yang Liu, and Lipo Wang. 2018. Apk2vec: Semi-Supervised Multi-view Representation Learning for Profiling Android Applications. In *2018 IEEE International Conference on Data Mining (ICDM)*. 357–366. https://doi.org/10.1109/ICDM.2018.00051

[36] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, Johannes Heidecke, Pranav Shyam, Boris Power, Tyna Eloundou Nekoul, Girish Sastry, Gretchen Krueger, David Schnurr, Felipe Petroski Such, Kenny Hsu, Madeleine Thompson, Tabarak Khan, Toki Sherbakov, Joanne Jang, Peter Welinder, and Lilian Weng. 2022. Text and Code Embeddings by Contrastive Pre-Training. arXiv:2201.10005 [cs.CL]

[37] Robin Nix and Jian Zhang. 2017. Classification of Android apps and malware using deep neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*. 1871–1878. https://doi.org/10.1109/IJCNN.2017.7966078

[38] Babatunde Olabenjo. 2016. Applying Naive Bayes Classification to Google Play Apps Categorization. arXiv:1608.08574 [cs.LG]

[39] OpenAI. 2022. https://openai.com/blog/introducing-text-and-code-embeddings.

[40] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[41] OpenAI. 2023. https://github.com/openai/openai-cookbook/blob/main/examples/How_to_count_tokens_with_tiktoken.ipynb.

[42] OpenAI. 2023. https://openai.com/blog/new-and-improved-embedding-model.

[43] OpenAI. 2023. https://platform.openai.com/docs/guides/embeddings.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[45] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1532–1543. https://doi.org/10.3115/v1/D14-1162

[46] Jathushan Rajasegaran, Naveen Karunanayake, Ashanie Gunathillake, Suranga Seneviratne, and Guillaume Jourjon. 2019. A Multi-Modal Neural Embeddings Approach for Detecting Mobile Counterfeit Apps. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 3165–3171. https://doi.org/10.1145/3308558.3313427

[47] William M. Rand. 1971. Objective Criteria for the Evaluation of Clustering Methods. *J. Amer. Statist. Assoc.* 66, 336 (1971), 846–850. https://doi.org/10.1080/01621459.1971.10482356

[48] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

[49] Mukund Rungta, Praneet Prabhakar Sherki, Mehak Preet Dhaliwal, Hemant Tiwari, and Vanraj Vala. 2020. Two-Phase Multimodal Neural Network for App Categorization using APK Resources. In *2020 IEEE 14th International Conference on Semantic Computing (ICSC)*. 162–165. https://doi.org/10.1109/ICSC.2020.00032

[50] Google Scholar. 2023. https://scholar.google.com/.

[51] Sevil Sen and Burcu Can. 2021. Android Security using NLP Techniques: A Review. arXiv:2107.03072 [cs.CR]

[52] Md. Shamsujjoha, John Grundy, Li Li, Hourieh Khalajzadeh, and Qinghua Lu. 2021. Checking App Behavior Against App Descriptions: What If There are No App Descriptions?. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. 422–432. https://doi.org/10.1109/ICPC52881.2021.00050

[53] Tiezhu Sun, Weiguo Pian, Nadia Daoudi, Kevin Allix, Tegawendé F. Bissyandé, and Jacques Klein. 2023. LaFiCMIL: Rethinking Large File Classification from the Perspective of Correlated Multiple Instance Learning. arXiv:2308.01413 [cs.CL]

[54] Wenqi Sun, Songyang Wu, and Zhi Xue. 2020. Clustering Mobile Apps based on Design and Manufacturing Genre. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*. 1956–1960. https://doi.org/10.1109/ICCC51575.2020.9344944

[55] Didi Surian, Suranga Seneviratne, Aruna Seneviratne, and Sanjay Chawla. 2017. App Miscategorization Detection: A Case Study on Google Play. *IEEE Transactions on Knowledge and Data Engineering* 29, 8 (2017), 1591–1604. https://doi.org/10.1109/TKDE.2017.2686851

[56] Virus Total. 2020. *Virus total free online virus, malware and url scanner.* https://www.virustotal.com/en

[57] Saarland University. 2014. https://www.st.cs.uni-saarland.de/appmining/.

[58] Wei Wang, Yuanyuan Li, Xing Wang, Jiqiang Liu, and Xiangliang Zhang. 2018. Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers. *Future Generation Computer Systems* 78 (2018), 987–994. https://doi.org/10.1016/j.future.2017.01.019

[59] Ryszard Wiśniewski. 2016. https://ibotpeaches.github.io/Apktool/.

[60] Xinli Yang, David Lo, Li Li, Xin Xia, Tegawendé F. Bissyandé, and Jacques Klein. 2017. Characterizing malicious Android apps by mining topic-specific data flow signatures. *Information and Software Technology* 90 (2017), 27–39. https://doi.org/10.1016/j.infsof.2017.04.007

[61] Chengpeng Zhang, Haoyu Wang, Ran Wang, Yao Guo, and Guoai Xu. 2018. Rechecking App Behavior against App Description in the Context of Third-party Libraries. 665–710. https://doi.org/10.18293/SEKE2018-180