LLM-Based Auto-Categorization of Android Applications

Pierangelo Loi[1,*], Diego Soi[1], Leonardo Regano[1] and Giorgio Giacinto[1,2]

[1]University of Cagliari, Italy
[2]Consorzio Interuniversitario Nazionale per l'Informatica (CINI), Italy

Abstract

Many anomaly-based malware detectors implicitly depend on the automatic categorisation of Android apps. Such tools first group apps by their declared functionality and then learn what constitutes "normal" use of sensitive APIs within each group. If this categorisation is wrong or overly coarse, the entire pipeline suffers: benign apps can be flagged as malicious, while malicious or grayware apps can hide among poorly matched neighbours. This highlights the need for precise, robust, and scalable categorisation methods that operate directly on app stores' metadata and can be plugged into security pipelines. In this paper, we present a *free* and fully automatic description-based framework for fine-grained categorisation of Android applications, explicitly designed as a drop-in replacement for the categorisation stage in anomaly-based systems. Our pipeline embeds Google Play Store descriptions with a sentence-transformer model, reduces the embeddings with Uniform Manifold Approximation and Projection, and clusters them using K-means while automatically selecting the number of clusters via the Mean Silhouette Coefficient. Finally, it employs a lightweight Large Language Model to generate concise, human-readable labels for each discovered cluster.

We evaluate our approach on AndroCatSet, a manually curated ground-truth dataset of 5000 benign apps organised into 50 fine-grained classes. The resulting categorisation component yields semantically coherent and interpretable functional groups, which can be readily integrated into security pipelines to strengthen the detection of miscategorised, malicious, and grayware apps whose actual behaviour diverges from their declared purpose.

Keywords

Android, App Categorisation, LLM, Clustering, Embeddings

1. Introduction

The Android ecosystem has grown into a vast and heterogeneous marketplace where millions of applications compete for users' attention. To organise this space, app stores such as Google Play Store (GPlay) assign each application to one or more high-level categories (e.g., TOOLS, PRODUCTIVITY, HEALTH). While these categories are primarily designed to improve search and recommendation, they are also implicitly used as a coarse proxy for *expected behaviour* in many security analyses. For example, a "flashlight" or "alarm clock" app is not expected to access contact lists and SMS messages, whereas a "messaging" app is.

Unfortunately, real-world categorisation is often too broad, inconsistent, and, in some cases, misleading. For instance, GPlay's categories, such as TOOLS or HOUSE & HOME, routinely mix apps with very different functionalities, such as weather widgets, file managers, smart-home controllers, and real-estate portals, making it difficult to characterise what constitutes normal behaviour within a category. Prior work [1, 2, 3] has also documented concrete misalignments between categories, descriptions and implementations. A well-known example is in the work of Gorla *et al.* [4], where the `SlideIT Free Keyboard` app was incorrectly grouped with "language" apps because its description emphasises language support, leading their framework CHABADA [4] to learn an inaccurate model of the category and wrongly flag `SlideIT` as suspicious despite it being benign. In this work, we aim to provide a practical

Joint National Conference on Cybersecurity (ITASEC & SERICS 2026), February 09-13, 2026, Cagliari, IT
*Corresponding author.
pierangelo.loi@unica.it (P. Loi); diego.soi@unica.it (D. Soi); leonardo.regano@unica.it (L. Regano); giorgio.giacinto@unica.it (G. Giacinto)
0009-0001-5302-8184 (P. Loi); 0009-0009-0092-9067 (D. Soi); 0000-0002-9259-5157 (L. Regano); 0000-0002-5759-3017 (G. Giacinto)

improvement of Android apps categorisation pipelines by designing a fully automatic, description-based categorisation framework that relies solely on *free* and *open source* models. Our intuition is that modern sentence-transformer embeddings already capture enough semantic information from GPlay descriptions to recover fine-grained functional groupings, and that small instruction-tuned LLMs can be used to attach meaningful labels to those groupings automatically. Such a component can then be plugged into anomaly detection pipelines, where it serves as a precise functional "lens" through which behavioural deviations, such as unusual permissions or data flows, can be identified.

Concretely, we make the following contributions:

- We design a fully automatic pipeline that embeds app descriptions with a sentence-transformer model, reduces them with Uniform Manifold Approximation and Projection (UMAP), and clusters them using K-means while automatically selecting the number of clusters via the Mean Silhouette Coefficient (MSC), thus avoiding any manual choice of the number of categories.
- We integrate a lightweight, open source Large Language Model (LLM) to generate concise human-readable labels for each discovered cluster from a small set of prototypical descriptions, enforcing a constrained output format that makes the labels directly usable in security tools.
- We evaluate our framework on AndroCatSet, a curated dataset of 5,000 apps and 50 classes, and show that it achieves an Adjusted Rand Index (ARI) [5] of 0.95[1] improving over the state-of-the-art G-CatA approach, which relies on a paid embedding API and a manually fixed number of clusters.

The rest of the paper is organised as follows. Section 2 reviews related work on app categorisation and its use in security. Section 3 details our methodology, from text preprocessing and embedding to clustering and LLM-based label generation. Section 4 presents our experimental results on AndroCatSet and compares our approach against existing methods. Section 5 concludes the paper and outlines directions for future work, in particular, the integration of our framework into broader malware and grayware detection pipelines.

2. Related Works

Automatic categorisation of mobile applications has been studied both as an independent problem, to improve search and discoverability [2, 3], and as a key building block for security tools such as malware detectors and anomaly detectors [4, 6].

One of the earliest and most influential description-based approaches in Android security is CHABADA by Gorla *et al.* [4], which targets apps whose implemented behaviour does not match what is advertised in their GPlay descriptions. CHABADA first collects around 32K apps from GPlay, retains 22,521 after filtering, and applies standard NLP preprocessing on descriptions (language detection to keep only English texts, stop-word removal, stemming, removal of non-text tokens). It then uses Latent Dirichlet Allocation (LDA) to mine 30 topics from the preprocessed descriptions and assigns each app up to four topics whose probability exceeds 5%. Instead of using store categories, CHABADA clusters apps based on their topic distributions, so that applications with similar textual descriptions end up in the same group. The authors show that such clusters are semantically coherent, for instance via word clouds where navigation apps share terms like map, GPS, route, while personalisation apps emphasise theme, wallpaper, launcher and so on. Within each description-based cluster, CHABADA statically extracts all sensitive API calls (those governed by Android permissions) from the APK and builds feature vectors that count how often each sensitive API is used. One-Class Support Vector Machine (SVM) is then trained per cluster to detect outliers whose API usage is atypical for that functional group. This combination of description-level clustering and API-level anomaly detection allows CHABADA to flag apps whose behaviour is inconsistent with their stated purpose, such as a "navigation" app that unexpectedly reads user accounts or SMS logs. In their evaluation, CHABADA detects 56% of previously unseen malware samples with a false-positive rate of 16%.

[1]The ARI ranges between -1 and 1, where 1 means perfect alignment with the ground truth and -1 complete disagreement.

Ma *et al.* [7] extend this work by formulating malware detection as an active semi-supervised classification task, rather than an unsupervised anomaly detection problem over benign apps. Their method derives latent semantic topics from app descriptions using LDA, with model parameters tuned via a Genetic Algorithm (GA), and extracts static behavioural features from APKs based on sensitive Android API invocations. Apps are encoded as high-dimensional vectors of topic–API co-occurrence features, enabling the detection of inconsistencies between stated functionality and observed behaviour. To jointly exploit labelled malware samples, large volumes of unlabeled apps, and limited expert annotation, the authors employ Ensured Collaborative Active and Semi-Supervised Labelling (ECASSL) on top of SVM classifiers.

Another similar approach has been proposed by Yang *et al.* [6], which used adaptive LDA with GA to extract 34 semantic categories from app descriptions and cluster them based on the most relevant topics. Then, topic-specific signatures have been generated from sensitive source-sink flows extracted via FlowDroid and leveraged to detect malicious behaviour.

This line of work demonstrates that semantically coherent categories can enhance security analyses. However, the quality of the anomaly detectors highly depends on the quality and granularity of the underlying categorisation, and classic topic modelling techniques such as LDA are very sensitive to vague or misleading descriptions. Coarse or noisy clusters can blur what "normal" behaviour is and lead to both false positives and false negatives. Another issue arises from the manual choice of the number of topics, e.g. 30 in the case of CHABADA, which may lead to overly broad categories, conflating heterogeneous apps (e.g., car rental, car games, car magazines; sports scores, sports betting, live streaming), increasing the risk of mislabeling benign behavioural variations as outliers. This limitation is particularly evident in cases such as SlideIT Free Keyboard, which CHABADA groups with "language" apps due to the wording of the description, and then incorrectly flags as anomalous.

Alecci *et al.* [1] explicitly revisit Android app categorisation and show that a similar issue affects GPLAY's own categories that tend to group apps that only share a very broad theme (e.g., "TOOLS", "HOUSE & HOME" and others), which makes them unsuitable as ground truth for evaluating categorisation algorithms. To address this, they construct AndroCatSet, a ground-truth dataset of 5,000 benign Android apps organised into 50 fine-grained classes with 100 manually chosen apps in each class. The dataset is built using a keyword-based scraping from GPlay, filtering against AndroZoo and VirusTotal to keep only apps with zero antivirus detections, and finally intensive manual verification of each app's purpose using name, description, screenshots, reviews and, when needed, installing them. In their work, they also propose G-CatA, a new description-based pipeline that uses OpenAI's *text-embedding-ada-002* model to embed descriptions, followed by K-means clustering in the embedding space with the number of clusters manually set to 50, to match the AndroCatSet classes. On AndroCatSet, G-CatA reaches an ARI Score of 0.91, outperforming all the other tested approaches. Furthermore, when plugged into a re-implementation of CHABADA, replacing the original LDA-based categorisation, G-CatA improves anomaly detection performance, illustrating how better categorisation directly benefits downstream security tools. However, G-CatA still has significant drawbacks from a practical perspective. It relies on a paid, proprietary API for embeddings; the number of clusters is fixed a priori to match the ground truth, rather than being inferred from the data, and it does not explore label generation or hierarchical structures. These limitations leave room for fully automatic, open, and more flexible pipelines such as the one we propose.

Beyond Android-specific security scenarios, Ebrahimi *et al.* [8] study mobile app classification more broadly and propose a supervised approach based on word embeddings for app descriptions. Their motivation is that traditional bag-of-words and topic-modelling techniques (e.g., LDA, Term Frequency-Inverse Document Frequency (TFIDF) vectors with BM25) struggle with short, sparse descriptions and that they require substantial manual tuning of hyperparameters such as the number of topics or clusters. In their framework, app descriptions are represented by aggregating pre-trained word embeddings (Word2Vec, GloVe, fastText) into document-level vectors, which are then fed into off-the-shelf classifiers such as SVM, Logistic Regression (LR), and Random Forest (RF). The main evaluation uses 600 apps sampled from the Education, Health & Fitness, and Medical categories, for which independent domain experts had previously defined fine-grained categories. They show that the best performance

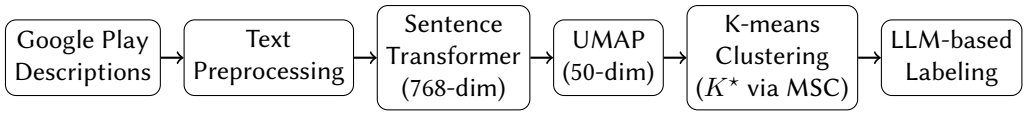

Figure 1: Overview of the proposed pipeline. Descriptions are cleaned, embedded with a sentence transformer, reduced with UMAP, clustered with K-means (with automatic selection of K via the Mean Silhouette Coefficient), and finally labelled using a lightweight LLM.

is obtained when descriptions are vectorised with GloVe embeddings and classified using SVMs. This combination produces labellings that are strongly aligned with expert categorisations; additional validation on a separate dataset of Sharing-Economy apps confirms that the resulting clusters match human judgments collected from 12 participants. At the same time, the authors highlight several limitations of existing classification pipelines. For instance, topic-model–based representations like LDA require a substantial amount of data, careful hyperparameter tuning, and perform poorly on short texts such as app descriptions. Ground truths are also often ad-hoc, researcher-defined label sets rather than expert-verified taxonomies. Their work demonstrates the benefits of semantically rich embeddings for supervised app classification, but it still assumes the availability of expert labels and focuses on relatively small datasets.

3. Methodology

Given the limitations of the previously mentioned works, our goal is to improve the state-of-the-art by creating a pipeline that automatically discovers fine-grained categories of Android applications solely from their GPlay descriptions and assigns a human-readable label to each discovered category, utilising free and open-source models. Figure 1 summarises our pipeline. At a high level, we i) clean and normalise app descriptions, ii) obtain sentence-level semantic embeddings using a free sentence transformer model, iii) perform dimensionality reduction with UMAP, iv) discover categories via K-means clustering with an automatic selection of the number of clusters, and v) use a lightweight LLM to generate concise labels for each cluster.

3.1. Description Preprocessing

For each app in the dataset, we extract its Google Play description. We keep the original text as provided on the store and apply a lightweight normalisation pipeline designed to reduce noise while preserving as much semantic content as possible. More specifically, the preprocessing steps are the following:

- Remove HTML tags, URLs, emojis and punctuation;
- Normalise Unicode characters;
- Tokenise the description using the standard tokenizer NLTK.

We experimented with stopword removal but observed a consistent degradation of clustering quality; thus, in the final pipeline, we do not remove them.

3.2. Description Vectorization

After preprocessing, to obtain a dense semantic representation of each description, we embedded them using the free and publicly available pretrained SentenceTransformers' model "*all-mpnet-base-v2*"[2], which maps sentences and paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search, balancing computational efficiency with semantic complexity capture. This model is widely adopted in the state-of-the-art for semantic similarity tasks across various works and contexts, including automated journal recommendation [9], reducing the workload associated

[2]https://huggingface.co/sentence-transformers/all-mpnet-base-v2

Input: Reduced embeddings $Z \in \mathbb{R}^{N \times d}$, candidate range $\mathcal{K} = \{K_{\min}, \ldots, K_{\max}\}$
Output: Selected number of clusters $K^\star$ and assignments $\mathbf{c}^{(K^\star)}$
$bestScore \leftarrow -\infty$
$K^\star \leftarrow K_{\min}$
$\mathbf{c}^{(K^\star)} \leftarrow \emptyset$
for $K \in \mathcal{K}$ **do**
  Run K-means on Z with K clusters, obtain assignments $\mathbf{c}^{(K)}$
  Compute silhouette scores $s_i^{(K)}$ for all $i = 1, \ldots, N$
  $MSC(K) \leftarrow \frac{1}{N} \sum_{i=1}^{N} s_i^{(K)}$
  if $MSC(K) > bestScore$ **then**
    $bestScore \leftarrow MSC(K)$
    $K^\star \leftarrow K$
    $\mathbf{c}^{(K^\star)} \leftarrow \mathbf{c}^{(K)}$
  end
end
return $K^\star$, $\mathbf{c}^{(K^\star)}$

Algorithm 1: Automatic selection of K via the Mean Silhouette Coefficient

with Graphical User Interfaces (GUIs) development [10], LLM-assisted threat intelligence [11], and improving medical information retrieval [12]. Directly using 768 dimensions to cluster the descriptions is computationally expensive, so we applied a dimensionality reduction step employing UMAP with the number of components set to 50 and the cosine distance as the similarity metric. We empirically explored several target dimensions between 10 and 256 and found that 50 dimensions provide a good trade-off between preserving local neighbourhoods and enabling effective clustering. Additionally, we chose UMAP after testing other techniques, e.g. Principal Component Analysis (PCA), which all provided worse results.

3.3. Automatic Category Generation

In a real-world scenario, where we might have massive datasets of uncategorised or poorly categorised applications, manually labelling samples becomes a very challenging task in terms of required time and effort. Not only that, but also understanding the right number of categories in the dataset and assigning them meaningful names is also very difficult and time-consuming. For this reason, we implemented a method to automatically determine the optimal number of clusters based on the
MSC: , which measures the similarity of samples to their own cluster compared to other clusters. The silhouette value ranges from -1 to +1, where a high value indicates that objects are well-matched to their own group and poorly matched to neighbouring groups. As depicted in Algorithm 1, we loop over a range of possible cluster numbers, e.g., from 10 to 150, actually computing clusterings and measuring the
MSC: for each iteration. After all iterations, we can tell which number of clusters provides the highest *MSC:* , and we proceed with that number. This way, we automatically obtain the most suitable number of clusters, allowing the framework to adapt to different datasets without any prior knowledge of the "true" number of categories.

To generate labels for each category, we first need to understand their context. To do so, for each discovered cluster, we compute its centroid and the cosine distance between each of its points and the centroid, selecting the closest ones. We call these points "*prototypical descriptions*", and we consider them to be the most representative descriptions of that cluster. We empirically evaluated different numbers of prototypical descriptions, ranging from 5 to 30, and found that using 10 descriptions yields the most accurate and consistent labels. This subset of descriptions is then used as input to the pre-trained LLM model "*google/flan-t5-large*" for label generation. We enforce a strict output format to ensure that the

generated labels can be programmatically consumed as category names.

Listing 1 presents the prompt used for label generation.

Other models have also been tested: LLAMA2 ("meta-llama/Llama-2-7b-chat-hf")[3] was not able to generate correctly formatted output for every description. In contrast, BART ("facebook/bart-large-cnn")[4] was not able to process long descriptions. In both cases, "google/flan-t5-large" was the fastest.

```
Instruction: You are given several Android app descriptions.
Your task is to generate a concise, meaningful category label for these apps.
Requirements:

- Output must be 1-3 words, in Title Case (e.g. "Photo Editor").
- item Do not include punctuation, quotes, or extra text, only the label.
- item Focus on the core theme or functionality shared by all examples.

Descriptions: {descriptions}
Category Label:
```

Listing 1: Prompt used with google/flan-t5-large for automatic category label generation.

4. Experimental Results

In this section, we evaluate the effectiveness of our framework in identifying fine-grained functional categories of Android applications and generating meaningful human-readable labels. We follow the evaluation protocol of Alecci *et al.* [1] and use AndroCatSet as the ground truth.

4.1. Dataset

Our experiments are conducted on the full AndroCatSet [1] dataset[5] of benign Android apps, for which both the APKs and their corresponding store metadata are available. It consists of 100 labelled samples for 50 handcrafted categories, totalling 5000 labelled applications. Each sample in the dataset is represented by the following information:

- its sha256 identifier;
- its package name;
- its label, which is the category that the dataset creators have manually assigned to the sample;
- GPlay category, which is the category under which the app appears in the Google Play Store;
- GPlay description.

Since GPlay categories are too broad and often imprecise, many applications are labelled differently from their GPlay category. As an example, we may find many applications whose GPlay's category is "TOOLS", "EDUCATION", "FINANCE" or even "PRODUCTIVITY", labelled as "Calculator". Of all this information, we were interested only in the labels and app descriptions; thus, we extracted this information and used it as a ground truth. Labels are used exclusively for evaluation.

4.2. Evaluation Setup

We instantiate the pipeline with the following configuration:

- the sentence-transformer `all-mpnet-base-v2` to obtain 768-dimensional embeddings for each preprocessed description;
- UMAP with cosine distance to reduce embeddings to 50 dimensions;
- K-means clustering in the reduced space;

[3] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[4] https://huggingface.co/facebook/bart-large-cnn
[5] https://github.com/Trustworthy-Software/Revisiting-Android-App-Categorization

- automatic selection of the number of clusters K by scanning a range of candidate values and choosing the one that maximises the MSC;
- the `google/flan-t5-large` model for cluster label generation, fed with the 10 descriptions closest to each of the cluster centroids.

To quantify the agreement between the discovered clusters and the ground-truth classes, we use the same evaluation metric of Alecci *et al.* [1], the Adjusted Rand Index (ARI) score, a standard clustering metric introduced by Hubert *et al.* [5]. Intuitively, ARI looks at *pairs of apps* and checks whether our clustering and the ground-truth labels treat each pair in the same way:

- a pair is *consistent* if both partitions put the two apps in the same group, or both put them in different groups;
- a pair is *inconsistent* if one partition says they belong together and the other says they do not.

The more pairs are treated consistently, the higher the ARI. A score of **1** means the clustering and the ground truth are identical, a score close to **0** implies that the agreement is no better than random, and a score close to **-1** means almost completely different clusterings.

More precisely, assume we have n elements and two partitions of these elements:

$$X = \{X_1, X_2, \ldots, X_r\} \quad \text{and} \quad Y = \{Y_1, Y_2, \ldots, Y_s\},$$

where X represents the clustering produced by a given approach and Y represents the ground truth. The overlap between X and Y can be summarized in a contingency table (n_{ij}) as follows:

$X \backslash Y$	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	

where each entry $n_{ij} = |X_i \cap Y_j|$ is the number of elements that appear simultaneously in cluster X_i and class Y_j, $a_i = \sum_j n_{ij}$ is the size of cluster X_i, and $b_j = \sum_i n_{ij}$ is the size of class Y_j [1].

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

$$\mathrm{ARI}(X,Y) = \frac{\displaystyle\sum_{i,j}\binom{n_{ij}}{2} - \frac{\displaystyle\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}}{\binom{n}{2}}}{\displaystyle\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \frac{\displaystyle\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}}{\binom{n}{2}}}. \tag{1}$$

In the remainder of this section, we report ARI as our main quantitative metric: higher ARI values indicate that our automatically discovered clusters are more closely aligned with the manually defined classes in AndroCatSet.

4.3. Quantitative Comparison with Existing Methods

We evaluate our framework in two settings. In the first setting, we directly fix the number of clusters to $K = 50$, matching the number of classes in AndroCatSet. In the second setting, we let the framework automatically select K by maximising the MSC over a predefined range of candidate values.

Table 1
ARI scores for each cited methodology

Paper	Methodology	ARI Score	Fixed Clusters	Free
CHABADA [4]	LDA topic extraction	0.70	Yes	Yes
Yang *et al.* [6]	LDA + GA topic extraction	0.68	Yes	Yes
Ebrahimi *et al.* [8]	GloVe embeddings + SVMs	0.59	Yes	Yes
G-CatA [1]	LLM (text-embedding-ada-00)	0.91	Yes	No
Our work	Transformer (all-mpnet-base-v2)	0.95	No	Yes

In the **fixed-K** setting ($K = 50$), we run K-means on the 50-dimensional UMAP embeddings and compute the ARI between the resulting cluster assignments and the AndroCatSet labels. We obtain an ARI of **0.95**, indicating that our purely description-based clustering is almost perfectly aligned with the manually defined classes.

In the **automatic-K** setting, we vary K over a range of 150, run K-means for each candidate, and compute the MSC. The maximum MSC is reached at $K^\star = 50$ with an MSC of 0.86, meaning that the intrinsic structure of the embedding space itself suggests the same number of categories as the ground truth. Using this automatically selected $K^\star$, we demonstrate that our procedure can recover both the *number* of categories and their *membership* from unlabeled data.

Table 1 compares our ARI scores against those reported by Alecci *et al.* [1] for several existing methods on AndroCatSet. Topic-model-based approaches such as CHABADA [4] and Yang *et al.* [6] achieve ARI scores around 0.70 and 0.68, respectively. The supervised word-embedding classifier by Ebrahimi *et al.* [8] reaches an ARI of about 0.59. The G-CatA pipeline, based on a proprietary embedding API and K-means with $K = 50$, attains an ARI of 0.91 and is reported as the best-performing method in their study. Our framework improves on this result by reaching an ARI of 0.95, while relying solely on free models and without requiring prior knowledge of the number of categories. G-CatA is claimed to improve anomaly detection performance in already existing frameworks, illustrating how better categorisation directly benefits downstream security tools, thus we expect to see even better results with the implementation of our pipeline.

4.4. Quality of Automatically Generated Labels

Beyond clustering quality, our framework also generates a concise, human-readable label for each discovered cluster using LLM-based methods. To assess the quality of these labels, we compare the names produced by our method with the manually defined class names associated with AndroCatSet.

Table 2 reports the resulting pairs. In most cases, the generated labels are either identical or very close to the handcrafted ones. For example, our method correctly names clusters as *Alarm Clock*, *Barcode Scanner*, *Translator*, *Weather*, *VPN*, or *Music Player*, matching the semantics of the corresponding classes. In a few cases, the label is slightly more creative but still clearly understandable, such as *Pocket Guide* for the *TravelGuide* class, or *Kitchen Stories* for *Recipes*. There are also isolated instances where the label is less informative, such as a label referencing a specific app name in a streaming cluster. However, these are rare and can be trivially corrected by a human analyst if needed.

Overall, the automatically generated labels provide a concise and interpretable summary of each cluster's functionality, which is crucial for security analysts who need to quickly understand what type of apps are grouped before inspecting behavioural anomalies.

4.5. Discussion

The experimental results highlight three main points. First, modern sentence-transformer embeddings, even without task-specific fine-tuning, are sufficient to recover very fine-grained functional classes from raw app descriptions. Second, the silhouette-based selection of K is effective in discovering the appropriate number of categories without any supervision, making the method suitable for new datasets

Table 2
Comparison of AndroCatSet categories vs ours

AndroCatSet	OURS	AndroCatSet	OURS
Alarm	Alarm Clock	Keyboard	Emoticon Keyboard
Airlines	Travel	LanguageLearning	Language Learning App
Antivirus	Virus Cleaner	Launcher	OS13 Launcher
Astrology	Horoscopes	Messenger	Messages
BarcodeAndQRcodeScanner	QR Code Scanner	MusicProduction	Music Maker
BikeAndScooterSharing	Electric Scooter	Navigator	Offline Maps
Banking	Mobile Banking	News	Breaking News
BooksReader	Audiobooks	Notepad	Notepad
Browser	Web Browser	OnlineTravelAgency	Hotel Discounts
BuyAndRentHome	Property	PhotoEditor	Photo Editor
Calendar	Calendar	PromoAndDeals	ClicFlyer
CarBuying	Car	PublicTransit	Transit
Calculator	Calculators	Radio	Radio
Dating	Dating	Recipes	Kitchen Stories
Dialer	Contacts	RemoteControl	TV Remote Control App
Drawing	Learn Drawing App	Shopping	Fashion App
Email	Email App	SmartHome	Smart Home
ExpenseTracker	Expense Tracker	Streaming	MAXstream
FileManager	File Explorer	Translator	Translator
FoodDelivery	Just Eat	TravelGuide	Pocket Guide
FoodDiaryAndCalorieCounter	Nutrition Coach	VideoPlayer	Video Player
HikingAndTrekking	Hiking	Vpn	VPN
Insurance	Insurance	Weather	Weather
Investment	Investing	Wallpaper	Wallpaper
JobSearch	Job Seekers	WorkoutAndTraining	Fitness

where no ground truth is available. Third, lightweight LLMs can reliably generate meaningful labels from a handful of representative descriptions, reducing or eliminating the need for manual naming.

From a security perspective, the resulting clusters and labels can serve as the functional backbone of anomaly detection frameworks: once apps are grouped into semantically coherent categories, one can learn what "normal" permission usage, data flows, or network behaviour look like for each category and flag apps that deviate from these patterns, including malicious and grayware applications whose implementation does not match their stated purpose.

5. Conclusions and Future Work

In this paper, we presented a fully automatic, description-based framework for fine-grained categorisation of Android applications, designed with security applications in mind. Our approach leverages free and open source models at every stage: we embed GPlay descriptions with a sentence-transformer, reduce them via UMAP, and cluster them with K-means while automatically selecting the number of clusters using the MSC. We then employ a lightweight LLM to generate short, human-readable labels for each discovered category from a small set of prototypical descriptions.

Using the AndroCatSet [1] dataset as an external ground truth, our framework achieves an ARI of 0.95, outperforming prior description-based methods, including the state-of-the-art G-CatA [1] pipeline, which relies on a paid embedding API and a manually fixed number of clusters. Qualitative analysis of the automatically generated labels further indicates that the resulting categories are not only accurate but also interpretable, closely matching handcrafted class names in the vast majority of cases.

These results suggest that modern sentence-transformers and small LLMs can be combined into a practical categorisation component that is both accurate and easy to deploy in real-world settings. From a security standpoint, our framework provides precise functional groupings that can serve as a

foundation for detecting apps whose behaviour is inconsistent with their declared purpose, thereby supporting the identification of malicious and grayware applications.

This work represents one building block of a broader research effort on Android security, rather than a complete detection pipeline. In future work, we plan to integrate our categorisation framework into end-to-end systems for malware and grayware detection. In particular, we expect that CHABADA-like anomaly detection systems may improve their performance as they are based on a first categorisation step. Our automatic, precise, and coherent categorisation approach could help reduce false positives and negatives, and enable testing on larger-scale datasets due to its higher automation.

Another promising direction is the detection of *grayware* and misrepresentor apps, which are not overtly malicious but breach user expectations by collecting unnecessary data, exhibiting aggressive advertising, or performing hidden tracking. Our description-based categories capture what apps claim to do from a user's perspective, and by combining them with behavioural analyses, it may become possible to identify systematic mismatches between declared and actual functionality and to surface them at scale.

Finally, we intend to extend our study beyond AndroCatSet by considering multilingual descriptions, additional app stores, and larger, mixed benign–malicious corpora. We also plan to explore hierarchical and incremental variants of our approach, where categories can be refined over time as new apps and behaviours emerge. All these directions converge towards a modular, scalable security pipeline in which our auto-categorisation framework serves as a reusable, high-quality functional lens for subsequent detection components.

Acknowledgments

This work was partially supported by the SERICS 1527 project (PE00000014) under the NRRP MUR program, funded by the EU-NGEU.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly to check grammar and spelling, paraphrase, and reword. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. Alecci, J. Samhi, T. F. Bissyandé, J. Klein, Revisiting android app categorization, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1–12.

[2] A. Aminordin, M. F. Abdollah, R. Yusof, R. Ahmad, Preliminary findings: Revising developer guideline using word frequency for identifying apps miscategorization, in: R. Saian, M. A. Abbas (Eds.), Proceedings of the Second International Conference on the Future of ASEAN (ICoFA) 2017 – Volume 2, Springer Singapore, Singapore, 2018, pp. 123–131. URL: https://doi.org/10.1007/978-981-10-8471-3_12.

[3] D. Surian, S. Seneviratne, A. Seneviratne, S. Chawla, App miscategorization detection: A case study on google play, IEEE Transactions on Knowledge and Data Engineering 29 (2017) 1591–1604. doi:10.1109/TKDE.2017.2686851.

[4] A. Gorla, I. Tavecchia, F. Gross, A. Zeller, Checking app behavior against app descriptions, in: Proceedings of the 36th international conference on software engineering, 2014, pp. 1025–1035.

[5] L. Hubert, P. Arabie, Comparing partitions, Journal of classification 2 (1985) 193–218. doi:10.1007/BF01908075.

[6] X. Yang, D. Lo, L. Li, X. Xia, T. F. Bissyandé, J. Klein, Characterizing malicious android apps by mining topic-specific data flow signatures, Information and Software Technology 90 (2017) 27–39. doi:10.1016/j.infsof.2017.04.007.

[7] S. Ma, S. Wang, D. Lo, R. H. Deng, C. Sun, Active semi-supervised approach for checking app behavior against its description, in: 2015 IEEE 39th Annual Computer Software and Applications Conference, volume 2, 2015, pp. 179–184. doi:10.1109/COMPSAC.2015.93.

[8] F. Ebrahimi, M. Tushev, A. Mahmoud, Classifying mobile applications using word embeddings, ACM Trans. Softw. Eng. Methodol. 31 (2021). doi:10.1145/3474827.

[9] M. T. Colangelo, M. Meleti, S. Guizzardi, E. Calciolari, C. Galli, A comparative analysis of sentence transformer models for automated journal recommendation using pubmed metadata, Big Data and Cognitive Computing 9 (2025). doi:10.3390/bdcc9030067.

[10] S. A. Hussain, R. Kohli, S. Zahoor, S. A. Sofi, Transforming the gui landscape: Harnessing the power of mpnet base v2 sentence transformers, Procedia Computer Science 259 (2025) 1809–1816. doi:https://doi.org/10.1016/j.procs.2025.04.136, sixth International Conference on Futuristic Trends in Networks and Computing Technologies (FTNCT06), held in Uttarakhand, India.

[11] S. Paul, F. Alemi, R. Macwan, Llm-assisted proactive threat intelligence for automated reasoning, arXiv preprint arXiv:2504.00428 (2025).

[12] R. Shetty, Enhancing medical text search: A study on small language models and retrieval-augmented techniques, Authorea Preprints (2025). doi:10.36227/techrxiv.174138500.08452275/v1.