Ranking Mobile Applications by Energy Efficiency

João Rocha
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
up201806261@edu.up.pt

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

5th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

6th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—In recent years, technological advancements have enabled the development of intelligent mobile phones, also known as smartphones, and their applications (apps) that help complete everyday tasks in a simple, fast, and mobile manner.**

According to statistics, it is estimated that 6.92 billion smartphones are being used across the World, and this number is expected to grow to 7.33 billion by 2025. The average user has 35 apps installed on his smartphone.

The enormous number of mobile applications being run raises concerns about the management of the batteries of mobile devices. Batteries have limited power, so they must be used efficiently to make them last longer without charging. Another issue is the negative environmental impact due to the amount of energy running these apps uses. Consequently, it is necessary to consider Energy Efficiency in mobile application development to minimise this impact.

The leading app stores do not provide information about the energy efficiency of the applications they distribute. With this, discussions about what we can do to alert developers and companies to be more cautious about the energy efficiency of their products arise.

We searched for related literature, analysed the approaches found to address the lack of energy labels and agreed that they could be improved.

This work focuses on designing and developing a framework that analyses Android applications and labels them based on their energy efficiency. The resulting labels from this project are obtained by combining the analysis of different anti-pattern detection tools that complement each other.

We believe that providing these energy labels can serve as an incentive for developers to be more energy efficiency aware of their applications, as well as help users choose more energy-efficient applications.

***Index Terms*—energy labelling, energy efficiency, mobile applications, apps**

I. Introduction

In today's world, we use smartphones regularly in our everyday lives. We use these devices to keep track of the time, stay up to date with our friends and family, have fun while playing games and much more.

In 2023 the number of smartphones worldwide has increased to 6.92 billion, which is expected to grow to 7.33 billion by 2025. This means that around 86.41% of all people nowadays have access to a smartphone. And these numbers are not expected to slow down any time soon.

Each smartphone is packed with a lot of apps with different functionalities. The mobile app industry has a vast market and is thriving. Mobile apps are expected to generate over $935 billion in revenue in 2023. When we look at the statistics about app usage, we can see that 49% of people open an app 11+ times per day and that 21% of Millennials do it 50+ times daily. The average user uses ten apps daily and has around 35 apps installed on their device.

The conventional distribution channels for mobile applications are App stores. The most famous ones are Google Play Store, developed by Google and Apple App Store, made by Apple. Another app store is Aptoide, created in Portugal and the focal distribution store referenced in this report. In 2023, 2.87 million and 1.96 million apps are available for download and installation in these App stores.

Both Apple App Store and Google Play Store do a similar job but are used in different operational systems (OS). While Apple App Store is only used on iOS devices (also developed by Apple), Google Play Store has a significant market share of Android devices. The market share of Mobile Operating Systems has two leading contenders. Android has 71.8% of the market share, while iOS has 27.6%.

With the abundance of smartphones worldwide using all kinds of mobile apps and using a sizable amount of energy, we need to start thinking about Energy Efficiency in this context [8], [11].

Mobile applications are becoming more complex and demanding more computational power than ever. Although we see development in software and hardware, batteries are not evolving at the same pace. If apps are not energy efficient, they contribute to increased energy consumption, thereby adversely impacting the global environment. These apps will also consume users' smartphone batteries faster than usual.

Previous studies have reported that many pages in app stores have comments referencing apps' energy efficiency [12].

However, no mobile application store provides information about the energy efficiency of an app.

Researchers have pursued various approaches to address energy efficiency in mobile app development. One such method involves the creation of guidelines and code refactors aimed at enhancing battery consumption [2]–[7].

II. Background

A. Previous Solutions

Some of the solutions proposed in other research papers take advantage of an energy profiler to directly measure or approximate the energy usage of a mobile application. Although these solutions can have good results, they are not scalable and need different test cases to analyse different applications. Other proposed solutions are based on anti-pattern detection and resource leaks but only use a few tools in the analysis or just compare apps with similar functionality.

III. Analysis and Labelling Tool

The solution proposed in this paper can be divided into three different and separate steps: Decompilation, Analysis and Classification. Figure 1 represents the architecture of the developed project.

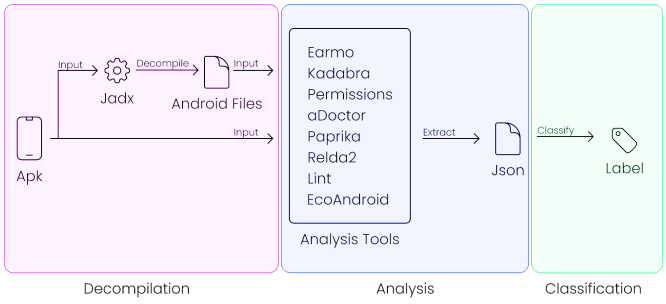

Fig. 1. Architecture Diagram

The following sections will describe in more detail each one of the steps.

A. Decompilation

This is the first step and where all the inputs for the analysis tools come from. While some tools only require the Android Package file (APK), others require the source code to work. To get the source code from the APK file, we need to run an Android decompiler, and for that purpose, Jadx was chosen. There are several decompilers, such as Dex2Jar, but Jadx has better documentation, and the decompilation results were more appropriate to the kind of inputs the analysis tools needed.

B. Analysis

The Analysis is the second and the most time-consuming step. Here is where the app will be analysed. This solution has the following analysis tools integrated:

Although these are implemented, we can easily add more analysis tools to this pool.

Tools
EARMO
Kadabra
Permissions Analyzer
aDoctor
Paprika
Relda2
Android Lint

TABLE I
Analysis Tools

This stage results in a JSON file with the number of detections and time of execution of each tool. This file also contains information about the app, such as the name, size, number of Java files, and categories.

C. Classification

This is the final step, but where the final label will come from.

Firstly, thresholds for each analysis tool are calculated after analysing a dataset of different apps. These thresholds are also calculated for each category to compare each category differently.

After calculating the thresholds, we compute a classification per tool for each mobile application. Then, we calculate the mean value of these classifications to get a single classification from 1 to 5 for each app. Lastly, we calculate new thresholds based only on the last values but this time not considering the apps' categories.

At this point, we will have thresholds for classifications from 1 to 7. The last step is to convert these numbers into labels.

D. Requirements and How to Run

To run the analysis and labelling tool, there are a few requirements:

Python 3.10 or above
Java 20.0.1 or above
Python Packages
Polars
Json
Multiprocessing
Tools
Jadx - Version 1.4.7
Kadabra - Build 20230210-1521
aDoctor
Paprika
Relda2
Android Lint

TABLE II
Program Requirements

This is the command to run the tool:

```
python main.py [-h] -categories
CATEGORIES [CATEGORIES ...]
[-analyzers [ANALYZERS ...]]
[-force]
```

```
[-fdroid FDROIDPACKAGENAME]
ApkPath
```

The following list shows and explains the mandatory and optional program arguments:

- ApkPath – The path to the APK file of the app to be analyzed. (Mandatory)
- -categories / -c – A list with the names of the categories of the app. (Mandatory)
- -analyzers / -a – A list with the names of the analyzers to run. Default: Earmo Kadabra AndroidManifestAnalyzer Lint ADoctor Paprika Relda2. (Optional)
- -force / -f – A flag that, if present, will force the execution of all or some analyzers discarding previously saved values. (Optional)
- -fdroid – The package name of an app from F-Droid. The program will try to download the app from F-Droid if a package name is given. (Optional)
- -aptoide – The package name of an app from Aptoide. The program will try downloading the app from Aptoide if a package name is given. (Optional)

The following code shows an example of how to run the tool:

```
python main.py -c Test Example example.ap
```

With these arguments, the tool will execute all analysis tools if there are no saved previous results and associate the app with the Test and Example categories. All analysis tools classifications and the final label will be calculated based on these categories and others that previously could have been associated with the app.

E. Further Details

The developed program can easily be modified to add or change any analysis tools with minor changes. Since the beginning of the development of the project, a possible modular solution was always in mind to ease future changes.

Another important detail is that we can use any category. This means that the categories used in this dissertation can be changed into any set of categories. But it is not restricted to a category. A company can, for example, create a new category with the name of the company to track and compare all its Android applications by their energy efficiency.

After the analysis and label computation, the program cleans all the directories used from useless post-analysis files. The only files that are not deleted are the log files, a text report and a JSON file with the results. The number of detections from each tool and the respective classifications are contained in this JSON file along with other stats.

If the **-fdroid** or the **-aptoide** argument is set, the program will try to download the app with the given package name from the respective distributor. If it is successful, it will analyze and label it and then delete the app from the system.

All the code developed in this project is available on Github: https://github.com/JayRx/Labelling-Android-Apps-by-Energy-Efficiency.

IV. Dataset

The dataset used to compute and calibrate the values of the labels was composed of several Android applications taken from F-Droid, a catalogue of free and open-source apps.

The apps were selected randomly and have various sizes, numbers of files and uses. To achieve this, a Python script was developed to Web scrape the F-Droid website and get all the apps in its database with the respective categories.

Then, a new Python script was made to select 50 mobile applications of each category randomly. The output of this code is a JSON file per category with the package names of the selected apps.

As shown in Figure 2, the dataset used has diverse kinds and sizes of apps represented, which can positively affect the results of the thresholds and labels when analysing apps in scenarios outside of testing. Figure 3 displays the number of Android applications from each category. As we can see, the lowest number of apps from the dataset in a category is 43.

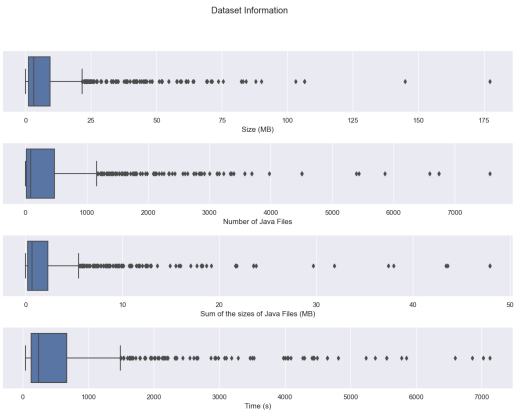

Fig. 2. Dataset Information

As shown in Figure 4, we calculated the Spearman rank correlation coefficients between some of the dataset characteristics. With the results we got, we noticed a strong correlation between the time of the analysis and the size of Java files of an app. This way, we can approximate the time an application is going to take to be analysed.

Each app can have one or more categories. In theory, any category name can be used in the app, but to calibrate and test the developed model, we split the dataset apps into 17 separate categories. We did not make these categories. We took advantage of the ones already created by F-Droid.

The categories are the following:

V. Label Calibration

To correctly calculate thresholds to help us decide which label to attribute to each application, we calculated separate

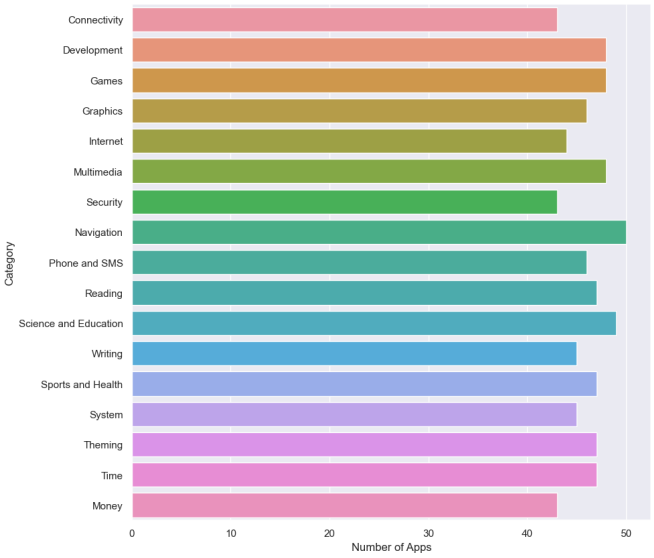

Fig. 3. Number of Apps per Category in the Dataset

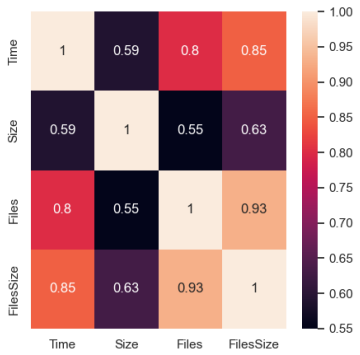

Fig. 4. Correlation Matrix between some characteristics of the dataset

Category Name		
Connectivity	Development	Games
Graphics	Internet	Money
Multimedia	Navigation	Phone and SMS
Reading	Science and Education	Security
Sports and Health	System	Theming
Time	Writing	

TABLE III
CATEGORIES

thresholds for each category and each analysis tool. This is because apps in different categories and with other purposes will need less or more energy. For example, a game will be more computationally expensive than a note-taking app. Thus we need to classify them differently.

We split the results file into separate datasets by category to achieve this. Then, for each of these, calculated the thresholds of 5 different classification levels for each tool depending on the number of detections. These levels are also characterized by a number of stars, being five stars the best level and one star the worst one.

To calculate the different thresholds, we followed a method based on making a density function based on weights [1].

VI. Results

Each Android application from the 738 different apps in the dataset was labelled following the previously described process. Figure 5 illustrates that the labels obtained are distributed across all energy labels. The dataset has many more apps with fewer anti-pattern detections, and bigger apps generally have more detections. This makes the labelling results of the dataset more skewed to the labels A and B. Table **??** shows the final classification and respective label of each mobile application in the dataset.

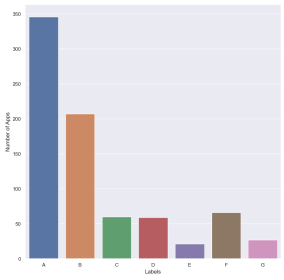

Fig. 5. Final Label Results

We compared the results from six different apps when running our labelling tool with Trepn [9] and EcoDroid to validate our classification and labelling method. Trepn is an energy profiler that measures the actual energy consumption of apps. The results of both Trepn and EcoDroid come from EcoDroid's scientific article [10]. Figure 7 illustrates that comparison and, as we can see, the results are very similar. The metric in the figure is a relative classification between the results of the six apps. Equation 1 shows how these values are calculated.

$$RelativeClassification(\alpha) = \frac{Classification(\alpha)}{MaxClassification} \quad (1)$$

We calculated the Spearman rank correlation coefficients using these values, which showed a 0.75 correlation between our and Trep results. This means that there is some correlation between the two values.

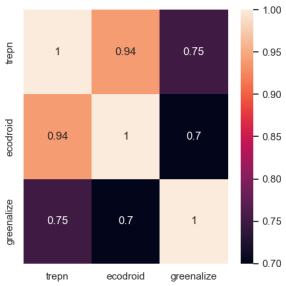

Fig. 6. Correlation Matrix between results

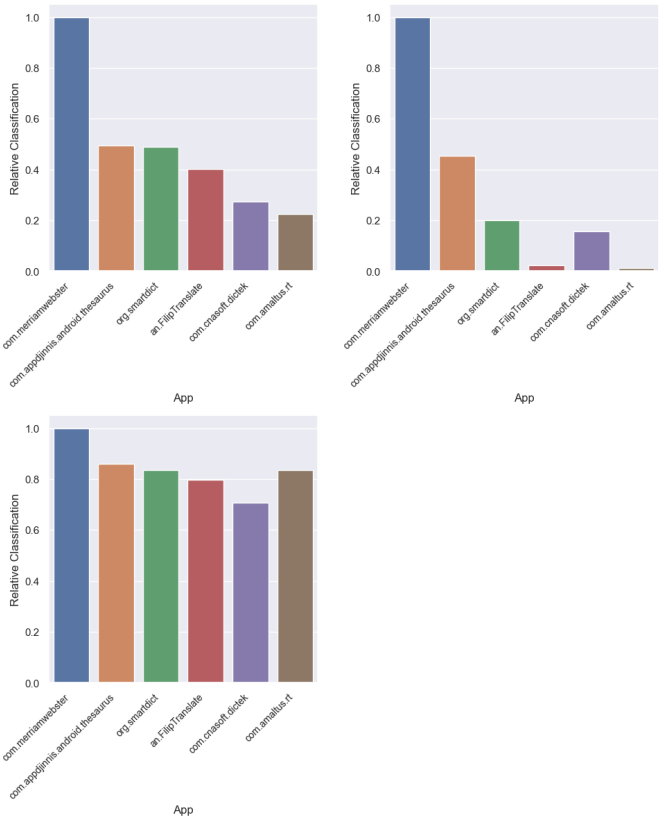

Fig. 7. Trepn, EcoDroid and Developed Tool Results Comparison

VII. Conclusion

As we have seen, there are billions of smartphones, each having, on average, 35 apps. With this, we must start thinking about energy efficiency applied to mobile applications. More efficient applications can bring advantages to users and the World. Since there is currently no app store showing information about apps' energy efficiency, this report contributes with a labelling solution to solve this problem.

To have a starting point, some initial research was done in which many solutions and tools were found. These solutions had different problems and some improvements that could be implemented. After this essential step, a solution was proposed using the tools researched that combats all limitations found in previous works.

The results obtained from this labelling method were good. The apps were compared with the data from different tools to get multiple points of view. The threshold calculation shows how comparing different apps regarding their types is possible. This way, apps that require more computational power and consume more power are not affected by their category, and we will have a more fair overall ranking.

Acknowledgment

This project was financed by FEDER (Fundo Europeu de Desenvolvimento Regional), from the European Union through CENTRO 2020 (Programa Operacional Regional do Centro), under project CENTRO-01-0247-FEDER-047256 – GreenStamp: Mobile Energy Efficiency Services.

References

[1] Tiago L. Alves, Christiaan Ypma, and Joost Visser. Deriving metric thresholds from benchmark data. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10, 2010.

[2] Abhijeet Banerjee, Lee Kee Chong, Clément Ballabriga, and Abhik Roychoudhury. Energypatch: Repairing resource leaks to improve energy-efficiency of android apps. *IEEE Transactions on Software Engineering*, 44(5):470–490, 2018.

[3] Huaqian Cai, Ying Zhang, Zhi Jin, Xuanzhe Liu, and Gang Huang. Delaydroid: Reducing tail-time energy by refactoring android apps. In *Proceedings of the 7th Asia-Pacific Symposium on Internetware*, Internetware '15, page 1–10, New York, NY, USA, 2015. Association for Computing Machinery.

[4] Marco Couto, João Saraiva, and João Paulo Fernandes. Energy refactorings for android in the large and in the wild. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 217–228, 2020.

[5] Luís Miranda da Cruz. Tools and techniques for energy-efficient mobile application development, 2019.

[6] Iffat Fatima, Hina Anwar, Dietmar Pfahl, and Usman Qamar. Detection and correction of android-specific code smells and energy bugs: An android lint extension, 2020.

[7] Daniel Feitosa, Luís Cruz, Rui Abreu, João Paulo Fernandes, Marco Couto, and João Saraiva. *Patterns and Energy Consumption: Design, Implementation, Studies, and Stories*, pages 89–121. Springer International Publishing, Cham, 2021.

[8] John Harris. Our phones and gadgets are now endangering the planet — john harris, Jul 2018.

[9] Qualcomm Technologies Incorporated. When mobile apps use too much power a developer guide for android app performance, 2013.

[10] Reyhaneh Jabbarvand, Alireza Sadeghi, Joshua Garcia, Sam Malek, and Paul Ammann. Ecodroid: An approach for energy-based ranking of android apps. pages 8–14. Institute of Electrical and Electronics Engineers Inc., 7 2015.

[11] Murray G Patterson. What is energy efficiency?: Concepts, indicators and methodological issues. *Energy Policy*, 24(5):377–390, 1996.

[12] Claas Wilke, Sebastian Richly, Sebastian Götz, Christian Piechnick, and Uwe Aßmann. Energy consumption and efficiency in mobile applications: A user feedback study. pages 134–141, 2013.