




# Analysis of Energy Consumption on Android Devices for Developers: A Systematic Mapping Study

Edwin Monteiro   [Federal University of Amazonas | [edwin@icomp.ufam.edu.br](mailto:edwin@icomp.ufam.edu.br)]

Helena Cavalcante  [Federal University of Amazonas | [helena.cavalcante@icomp.ufam.edu.br](mailto:helena.cavalcante@icomp.ufam.edu.br)]

Raimundo Barreto  [Federal University of Amazonas | [rbarreto@icomp.ufam.edu.br](mailto:rbarreto@icomp.ufam.edu.br)]

Rosiane de Freitas  [Federal University of Amazonas | [rosiane@icomp.ufam.edu.br](mailto:rosiane@icomp.ufam.edu.br)]

 *Institute of Computing, Universidade Federal do Amazonas, Av. Rodrigo Otávio, n° 6200, Coroado I, Campus Universitário Senador Arthur Virgílio Filho, Setor Norte, CEP 69080-900, Manaus/AM Brazil.*

**Received:** 22 February December 2022 • **Accepted:** 15 June 2023 • **Published:** 11 December 2023

**Abstract** Reducing energy consumption is a major challenge that mobile computing must deal with. Smartphones are constantly evolving to match traditional computers in many aspects, especially in processing and memory. However, user experience is severely impacted by the rapid discharge of batteries in smartphones. This article aims to identify which metrics and assessment techniques are applied to evaluate energy consumption on Android smartphones, by summarizing factors that cause high energy consumption through a Systematic Mapping Study (SMS). The methodology of this SMS consisted of performing searches on the digital libraries ACM, IEEE, and Scopus. Sixty articles were obtained, of which 17 were identified as relevant for this study. Among the main methods identified, energy consumption is collected at time intervals based on information on battery voltage/current or on information from features such as Wi-Fi, cellular networks, screen brightness, screen duration on, Bluetooth usage, and others. Regarding the tools and applications that collect such information, there are Android batterystats, applications developed specially for each research like BatteryHub. The data is mainly analyzed by techniques such as clustering (17.65%), covariance (17.65%), Bayesian classification, and decision trees (11.76%). From these techniques, it was identified that user profile is the main factor affecting battery performance, being present in 23.53% of the articles, followed by mobile networks and Wi-Fi (17.65%), in addition to applications and services in background present at 11.76%. Finally, two articles in this SMS provide recommendations to reduce consumption based on users' usage profiles. In summary, the analysis has shown a significant correlation between user habits and energy consumption. As a result, it is recommended that developers prioritize the exploration of artificial intelligence techniques to automatically adjust smartphone usage settings based on usage context. This approach can lead to significant battery power savings.

**Keywords:** Android apps. Android smartphone. Battery Saving. Energy consumption. Energy Efficiency. Mobile Computing.

## 1 Introduction

This Systematic Mapping Study (SMS) focuses on smartphones with Android Operating System, since it is the operating system (OS) widely used in mobile computing and which has gained prominence since its launch by Google and the Open Handset Alliance in 2007. Android is a Linux kernel-based open-source system that allows interaction with users from touches and gestures on the smartphone screen. In more recent statistics taking into account the 2009-2021 interval, it is possible to notice from Statcounter [2021] that Android is present in 39.75% of smartphones in terms of the market.

Despite the rising popularity of smartphones, new challenges have arrived. Processing power, as well as application complexity, evolves rapidly each year. However, despite a greater battery capacity, energy autonomy has not evolved at the same speed as the features introduced or improved in smartphones. Analysis of energy consumption in applications is critical to decision-making seeking to increase battery life, not only in terms of improving user comfort but also seeking to reduce the environmental costs of disposing of batteries and e-waste. The focus of this mapping study

is smartphones with the Android operating system, as this is the operating system that occupies the largest fraction of the market, aside from being an open-source OS that allows many developers to contribute.

In a study by Pang *et al.* [2016], only 18% of a population of 122 app developers interviewed consider energy savings during development, and only 10% of programmers measure the energy consumption of their software. These low rates are largely due to the lack of knowledge of programmers regarding techniques, methods, softwares, and frameworks for the analysis of such an important performance metric of embedded systems. This systematic mapping study seeks to synthesize and evaluate the different approaches in the current literature, to provide developers with a comprehensive view of the state of the art on energy consumption of Android smartphones. Therefore, we considered the most recent publications that present the evaluation and reduction of energy consumption of Android devices.

This paper also identifies the main techniques and methods used in the analysis of energy consumption recommendations for application developers in order to reduce the battery drain on Android smartphones, by identifying the fac-

tors with the greatest impact on battery life. An examination of literature publications reveals a prominent trend among developers towards customized assessments of energy usage. This trend considers the impact of user behavior and decision-making in the development of energy-efficient solutions. There is a noteworthy interest among developers in providing personalized recommendations for individual user profiles, aimed at enhancing decision-making processes and promoting efficient battery usage on smartphones.

Thus, the research questions (RQ) that guide the development of this mapping study, prepared in accordance with the guidelines provided by Kitchenham and Charters [2007], are:

- **RQ1:** “What techniques and methods are currently used to investigate the factors that impact Android smartphone battery consumption?”;
- **RQ2:** “What factors have the greatest impact on Android smartphone battery consumption?”;
- **RQ3:** “What are the main recommendations to reduce the battery consumption of Android smartphones?”.

The main contributions of this work are: (i) to provide an overview of the state of the art, and; (ii) to identify gaps in the literature on energy consumption in Android smartphones. Through a global and comprehensive view of the leading methods, techniques, tools, and experiments described in the literature, this work aims to be a guide to: (i) researchers, suggesting new and promising research paths; and (ii) developers, adding new ways to create apps that drain less energy.

This paper is divided as follows: Section 2 details the SMS protocol and describes the search strategies, search strings, and inclusion and exclusion criteria. Section 3 presents the results identified in the reading and extraction of the evaluated articles. The discussion of these results is detailed in Section 4 and the research questions are answered in Section 5, and finally, Section 6 discusses the limitations of the works, gaps to be evaluated and closes with the final considerations.

## 2 Methodology

This section describes the protocol of the mapping study carried out, whose purpose is to establish the necessary foundations to guide the conduction of the review, in order to identify the leading methods, techniques, tools, and approaches that deal with energy consumption on Android smartphones.

### 2.1 Search strategies

The query for articles using search expressions named “search strings” was performed in the Scopus indexing database. This database aggregates articles from other digital databases such as Elsevier, Springer, and IEEE. In addition to Scopus, the digital bases of IEEE and ACM were also consulted. In terms of filters, only papers published in English from 2017 to October 2021 were considered. The documentation and conduction of the systematic review were carried out with the aid of StArt software (State of Art through Systematic Review), distributed by Federal University of São Carlos [UFSCar, 2021].

The PIO methodology (Population, Intervention, Outcomes) described by Kitchenham and Charters [2007] was applied to define the search strings. This methodology consists of using a conjunction of disjunctions of three-word blocks. These blocks contain words and synonyms associated, respectively, with population, intervention, and desired results of the systematic review. Items for each category of PIO methodology are specified in **Table 1**.

**Table 1.** PIO Methodology (Population, Intervention, and Outcome) applied in the context of this SLR.

PIO	Description
<b>Population</b>	Android Smartphones.
<b>Intervention</b>	From the identification of consumption, charging, discharging and battery saving
<b>Outcome</b>	Techniques, frameworks, databases, metrics, architectures and analysis related to energy consumption in Android smartphones

The definition of the words and synonyms present in the composition of the string arose from an iterative process divided into two stages: (i) carrying out an ad-hoc search in the digital databases of Scopus, ACM, and IEEE based on key terms extracted from the control papers and; (ii) extraction of terms from the IEEE, ACM, and Scopus digital databases. Iterations were controlled through the number of relevant articles returned. After this step, a search was conducted on the same databases through the string elaborated and presented in **Table 2**. In total, 342 articles were returned, 59 of which were duplicates. This high number of duplicate papers can be attributed to the fact that Scopus also indexes articles from IEEE and ACM. Of the 342 articles under review, 246 were obtained from Scopus, 78 from IEEE, six from ACM, and 12 were manually entered.

### 2.2 Inclusion and Exclusion Criteria

The title, abstracts and keywords of the 283 papers, a number obtained from the difference of all 342 articles returned by 59 papers identified as duplicates, were systematically read during the first filter of the selection stage and subsequently accepted or rejected according to the criteria defined in the Table 3 and Table 4.

**Table 2.** Final search expression compatible with the three databases under study performed on May 1, 2021.

Indexing Base	String
ACM, IEEE and Scopus	( “android device*” OR “android smartphon*” OR “android app*” OR “android servic*” ) AND ( “battery charg*” OR “battery discharg*” OR “battery usage” OR “battery consumption” OR “energy consumption” OR “energy behavior” OR “energy efficiency” OR “battery powered” OR “energy saving” OR “saving energy” OR “power saving” OR “power consuming” ) AND ( approach* OR analy* OR metric OR measurement OR methodolog* OR model* OR framework* OR recommend* OR techniqu* OR dataset OR database )

For acceptance, the papers of interest must address energy consumption on Android smartphones as its main subject as written in inclusion criteria 1 (IC1), present techniques, metrics, approaches, methodologies or recommendations focused on the research topic (IC2) or use/produce databases with energy metrics related to a specified population (IC3).

For nonacceptance, the papers must not be related to any inclusion criteria and fit with some criteria in **Table 4**. In summary, according to this table, one paper is excluded if: it refers to procedures not allowing to identify, through the title, keywords, or abstract, the agreement with the objective of this mapping; it is not available for full reading; it deals with power consumption in Android apps, but their focus is not on the device, for example, it focuses on web applications; it features programming-level intervention, but there is no replication for Android, for example, use/modification of some Java library that is not part of the standard Android API and was not written specifically for Android and the study does not perform tests on the mobile system; All these papers are discarded.

To validate the first selection step, carried out by two researchers, Kappa criterion of agreement between pairs was calculated. The methodology detailed by Perez [2020] served as the basis for calculating Cohen's Kappa coefficient. The criteria refinement was carried out in three iterations with 15 samples each, totaling a sample of 45 papers. The researchers classified the articles individually and, after each iteration, held a meeting to adjust the criteria. The third and last iteration resulted in a Kappa coefficient of 0.815, which represents, according to Perez [2020], an almost perfect agreement among researchers.

A total of 60 papers met the inclusion criteria corresponding to the first filter of the selection of papers that considered only the titles, abstracts, and keywords. Currently, the number of articles has increased to 60 with the inclusion of six papers suggested by the reviewers. In this first filter, it is possible that a number considered high is accepted since a deep reading is not performed for this step. After that, the second filter of the selection step was performed, comprising a full read of the introduction, methodology, and conclusion sections of these papers. Out of the 60 works, only 30 kept on meeting the inclusion criteria. It then proceeded to the full reading and analysis of papers in the extraction step, which still implied the rejection of 13 papers, leaving 17 as primary studies. **Figure 1** shows a schematic of the article selection process.

### 3 Summary of Results

Among the various articles analyzed, there is a trend in experiments aimed at generating databases or using available databases in order to apply metrics aimed at energy consumption through data mining. Two of these databases, described in Pereira *et al.* [2021] and Rua *et al.* [2019], are open to the general public.

In the papers read, three possible approaches were identified to estimate the energy consumption of an application. The first is through hardware measurement. This approach returns more accurate results. The analyzed papers make use

of hardware external to the smartphone components, which demands its acquisition and the need for specialized knowledge. Also, it does not help to discriminate the causes of power consumption within a device. The second is through the construction of energy models. This method is easier to implement. However, its main disadvantage is in the calibration of parameters, which is not always an easy task. The third approach is through software, although simple to implement, it presents results of low precision. The paper by Oliveira *et al.* [2019] mentions the RAPL (Running Average Power Limit) feature available on Intel processors to analyze power consumption. However, RAPL is not present as an approach in this article due to the focus on desktop/laptop processors.

Among studied publications, the predominant type of analysis is dynamic analysis, present in twelve of the fourteen articles. Dynamic analysis is understood as simulations, controlled experiments, or data collection aimed at analyzing energy consumption at runtime, that is, while the user interacts with the smartphone. Five papers, Guo *et al.* [2017], Pereira *et al.* [2021], Dai *et al.* [2020], Almasri and Sameh [2019] and Linares-Vásquez *et al.* [2018], also use static analysis and in four, Dai *et al.* [2020], Pereira *et al.* [2021], Oliveira *et al.* [2017] and Oliveira *et al.* [2019] it is done in conjunction with dynamic analysis. The most common features collected are Wi-Fi status, network connection (3G/4G), Bluetooth status, CPU usage, baud rates in kBs, and screen brightness. The numbers of these features in the primary studies are described in **Table 6** and **Figure 2**. The **Table 6** organizes features into nine groups. For example, the foreground and background applications group refers to the technique of collecting the names of applications in use at a given time interval such as one second. Battery level corresponds to the available battery charge range (1%-100%). The Screen status group refers to the state of the screen, that is, whether it is on or off. Wi-Fi and Cellular network describe the status of these features as on or off and Transmission Rates refer to the sending and receiving rates of these features. To calculate battery consumption and link this consumption to an app, information is usually collected from apps running in the foreground or background, battery level, and in some cases, battery voltage and electrical current.

Some of the main softwares and libraries for energy consumption collection are Treprn Profiler, Android TreprnLib, PowerTutor, and BatteryStats, as shown in **Table 7**, in which the exact numbers are detailed. Treprn Profiler is an Android application for collecting data related to smartphone performance, such as GPU frequency, consumed power, memory, and network interface, as per information from Qualcomm [2017]. PowerTutor is an Android application for energy consumption analysis, according to Dick [2011]. Although Treprn and PowerTutor are present in seven papers of **Table 7**, these apps have been discontinued, which affects their use in recent versions of Android like 10 and the newest. Batterystats is a tool included in the Android framework responsible for collecting battery data from smartphones [Google, 2021]. It shows the energy consumed by the main smartphone components. The most used software and libraries for comparative studies to determine precision in frameworks are BatteryStats Plugin and, in most studies, Monsoon Power Moni-

**Table 3.** Inclusion criteria defined for the selection of mapping articles.

Criteria id	Description of Inclusion Criteria (IC)
IC1	Papers or book chapters in which the main study is energy consumption in Android smartphones.
IC2	Papers or book chapters that describe techniques, metrics, approaches, methodologies, models, recommendations applied to energy consumption in Android smartphones.
IC3	Papers or book chapters that use/provide databases related to energy consumption on Android smartphones.

**Table 4.** Exclusion criteria defined for the selection of mapping articles

Criteria id	Description of Exclusion Criteria (IC)
EC1	Papers derived from the same research without inserting new ideas (duplicates).
EC2	Papers dealing with power consumption in Android apps, however the focus is not on the device.
EC3	Papers focused on web apps or applications that do not deal with energy consumption on Android smartphones.
EC4	Papers or book chapters that are not available for full reading
EC5	Papers or collection of papers referring to the proceedings that do not allow to identify, through the title, keywords or abstract, the agreement with the objective of this mapping.
EC6	Papers or book chapters in which the central theme involves energy/battery consumption in Android smartphones whose study is not intended to propose improvements.
EC7	Papers that present evaluations without showing the method used.
EC8	Papers dealing with power consumption on Android smartphones at the programming or hardware level without the purpose of offering improvements.

tor, a hardware device to estimate power dissipated in smartphones with voltage and current measurements. Some studies also carry out comparative studies with their own electronic instrumentation. Monkey is the most used software to perform automated tests in Android applications.

The analyzed publications fall into four categories organized into subsections, as shown in **Table 5**: four articles, 23.53%, report controlled studies to determine the pattern of energy consumption – Subsection 3.1; five articles address the development of frameworks for analysis and management of energy consumption – Subsection 3.2, corresponding to 29.41% of the total articles; two articles, 11.76%, cover studies of patterns based on user behavior – Subsection 3.3; and, finally, six papers, 35.29%, detail techniques to increase battery life – Subsection 3.4.

### 3.1 Controlled studies to determine energy consumption patterns in Android apps

In order to reduce the environmental impact of smartphone battery disposal, which cannot be reused or recycled, Elliot *et al.* [2017] sought to experimentally determine and analyze the energy consumed by certain multimedia applications and conduct interviews to document user behavior and its impact on battery consumption. Using Treppn Profiler application (refer to Qualcomm [2017]), which is a resource monitor that allows the monitoring of features such as screen brightness, Wi-Fi status and GPS, Elliot *et al.* [2017] collected CPU and temperature information. The experiment was conducted in

two stages. The first was related to the use of a smartphone of the Samsung Galaxy Note model to collect data about smartphone features such as hardware settings, temperature, screen brightness, network (Wi-Fi and cellular), installed applications and user behavior.

Results were grouped into the categories of audio applications, video/streaming applications and chat. In addition, each group was structured in terms of the Wi-Fi network and mobile networks. Manufacturer's native applications such as Samsung music and Samsung video performed better when using both Wi-Fi and mobile networks. For example, in audio applications the difference in energy consumption was 0.49 J/s for Samsung Music when using Wi-Fi. Regarding CPU usage, it was close to the average with 46%. For video applications while the YouTube and VLC Player applications consumed around 1.24 J/s, the native Samsung video application consumed around 0.57 J/s with only 49% CPU usage being below average usage, around of 54%. It is worth noting that there were noticeable variations for use over Wi-Fi and use over cellular networks. The Youtube and Samsung video apps had a slight drop with a difference of 0.42 J/s and 0.02 J/s respectively for energy consumption and a drop of 2% and 6% for CPU usage. Regarding the VLC application, switching from Wi-Fi to mobile data resulted in a considerable increase in energy consumption, jumping from 1.25 J/s to 10.45 J/s. Finally, to conclude this analysis by category, the chat had different behavior between the Viber and WhatsApp applications depending on the network used. For Wi-Fi, the apps had one of the lowest energy consump-

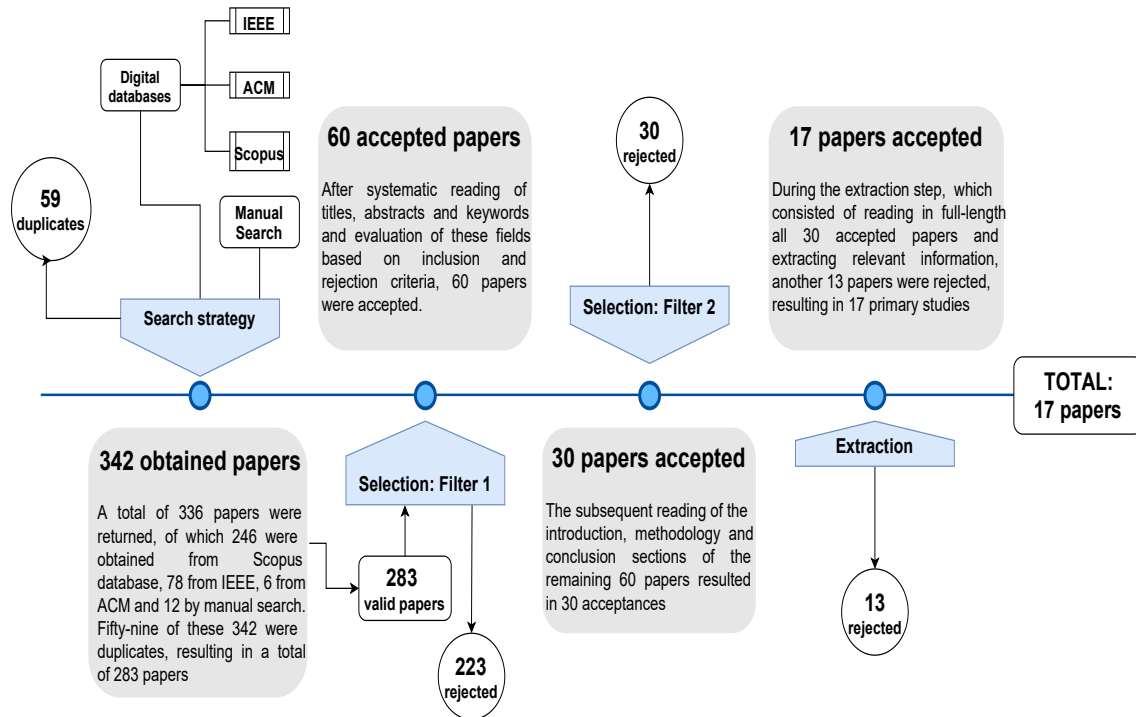


Figure 1. Number of papers accepted at each stage of the systematic review process.

tions, around 0.49 J/s and 0.46 J/s. Regarding the mobile network, Viber had 0.29 J/s, in this case a decrease, while WhatsApp recorded 0.62% indicating an increase. CPU usage decreased by 10% compared to that identified for Wi-Fi. Overall, the biggest energy consumption was in the media apps with Google music and VLC Player being the main villain. It was not possible to define which network drains more battery as the high consumption switched between the two network types. Although there is a clear difference between Wi-Fi and cellular network, it concerns specific experimental scenarios and not in a general way to say which consumes more energy, thus reiterating that there is no consensus.

Finally, a questionnaire applied at the end of the experiment tried to gather information that could help to understand the collected data. Among the questions were “What activities do you do on your smartphone?”, “How much time do you spend on video apps daily?” and “How much time do you spend on social media apps?”. The results obtained from the questionnaire are in line with the findings in the experiment, more precisely about video applications that tend to drain more energy than others such as audio applications. Social media apps have the same behavior. What can be noticed is that there is not a single trend among users of the experiment given the usage profile, as there are reports of users who charged the smartphone twice a day while other users did not have this need. Thus, the results were not conclusive.

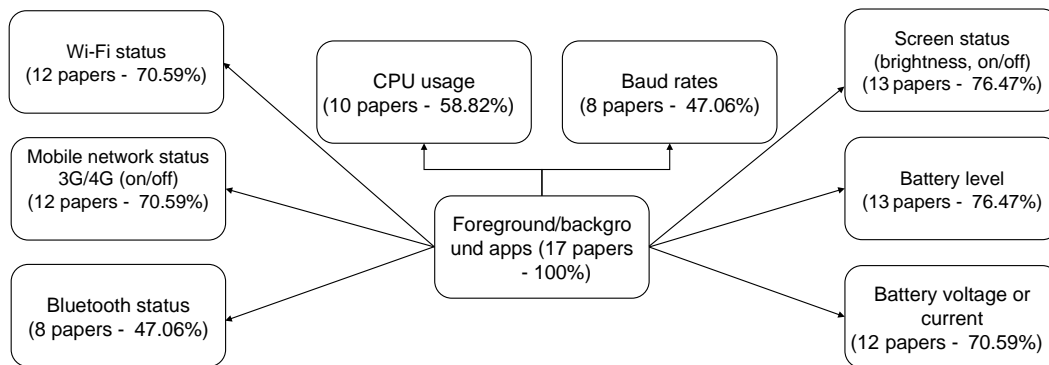
In order to study the energy consumption of applications, Almasri and Sameh [2019] propose a method that combines static and dynamic analysis for rating applications in regards to their energy consumption, on a scale of 1 to 6 stars. They present a simple and intuitive way to show, at the time of an app installation, its impact on battery life. The method investigates the different types of permissions required for applications to be installed and run correctly. Among the

various permissions we can mention “Allows applications to access information about networks” and others such as “Allows an app to search precise location from location sources such as GPS, cell towers, and Wi-Fi”. There are two ways to identify the required permissions, the first is to inspect the source code of each application through the “AndroidManifest.xml” file, the second is to collect it directly from the Android application store, the Play Store. The second option is identified as the most viable, because although it is not as accurate and does not reflect all the commands as in a source code, it is always updated, because as soon as an application is updated in the store, the permission information is also updated. From this information, each application is evaluated in terms of star from the point of view of energy consumption. In summary, the applications in the store are organized into 12 categories, and each group has permissions in common, so there is a filtering of permissions to extract only those associated with energy consumption. Finally, the profiling technique is applied through the PowerTutor application<sup>1</sup>, an app similar to the previously mentioned Trepn, to collect information from smartphone features, within 1 second, in order to measure the amount of energy consumption of components such as processor, cellular network, screen, vibration, cameras, flashlight, speakers and microphone. Given this information, Almasri and Sameh [2019] defined a scale to evaluate the collected features with respect to energy consumption. The highest consumption measured was ~25 mAh and the lowest was ~5 mAh. Therefore, the consumption scale is in the range between 1 and 30 mAh where one star means low power consumption and six means high power consumption. The top five categories of Play Store applications in terms of energy consumption are: (i) Social with five stars on Wi-Fi (~26 mAh) and six stars on cellular net-

<sup>1</sup><http://ziyang.eecs.umich.edu/projects/powertutor/>

**Table 5.** Classification of publications analyzed in this review.

Type of Study	Studies	Amount	Percentage
Controlled studies to determine energy consumption patterns in Android apps	Elliot <i>et al.</i> [2017]	4	23.53%
	Mehrotra <i>et al.</i> [2021]		
	Almasri and Sameh [2019]		
	Barreto Neto <i>et al.</i> [2020]		
Frameworks for analysis and energy consumption management	Rua <i>et al.</i> [2019]	5	29.41%
	Di Nucci <i>et al.</i> [2017]		
	Wang <i>et al.</i> [2017]		
	Duan <i>et al.</i> [2017]		
	Malavolta <i>et al.</i> [2020]		
Usage patterns of mobile devices	Guo <i>et al.</i> [2017]	2	11.76%
	Pereira <i>et al.</i> [2021]		
	Zhang <i>et al.</i> [2019]		
Techniques for reducing energy consumption	Cañete <i>et al.</i> [2020]	6	35.29%
	Harihar and Sukumaran [2018]		
	Oliveira <i>et al.</i> [2017]		
	Oliveira <i>et al.</i> [2019]		
	Linares-Vásquez <i>et al.</i> [2018]		



**Figure 2.** Most commonly collected device and system features for energy consumption analysis.

**Table 6.** Groups of most collected features among the analyzed papers.

Features Collected	No. Articles	Percentage
Applications in foreground and background	17	100%
Battery level	13	76.47%
Screen status	13	76.47%
Wi-Fi	12	70.59%
Cellular network	12	70.59%
Battery Voltage and/or current	12	70.59%
CPU	10	58.82%
Transmission Rates	8	47.06%
Bluetooth	8	47.06%

work (30mAh); (ii) Tools with five stars in both Wi-Fi (~26 mAh) and mobile network (~26 mAh); (iii) Communication with five stars both on Wi-Fi (~25 mAh) and on the cellular network (~28mAh); (iv) Personalization with four stars on Wi-Fi (~20mAh) and five stars on cellular network (~22 mAh); and finally, (v) Lifestyle with four stars in both Wi-Fi (~21 mAh) and cellular network (~21 mAh). Almasri and Sameh [2019] concluded that social category consumes the most energy.

Seeking to determine how energy consumption patterns vary regarding apps and respective system settings, Mehrotra *et al.* [2021] developed a multi-class classification algorithm that separates mobile apps into three categories in terms of energy consumption: low, medium and high. The objective is to map the energy consumption from the consumption pattern of the characteristics of smartphones. For this, Mehrotra *et al.* [2021] collected the energy consumption of 90 different applications in order to build a dataset based on 10 predictor attributes: total energy consumption, LCD, processor, Wi-Fi network, cellular network, RAM memory and others. like name and app category. As mentioned, the data were organized into the categories of natives, tools, entertainment, social media, education and advertising. The data collected by the profiling technique through the Trepro Profiler application were generated for the paper’s experimentation, so there is no label for this database, which makes it difficult to apply supervised machine-learning models.

Mehrotra *et al.* [2021] uses unsupervised learning to label the data. To this end, the K-means cluster algorithm was used to organize the data into three clusters defined by the authors as low, medium and high, which will be the labels of the database. Applications related to applications of the tools category consume low power and are found in cluster one and two. Applications related to advertising demand high power consumption generating high contribution to the cluster la-

**Table 7.** Identification of most used software in data collection for energy consumption analysis.

Software category	Power consumption collection software	Estudos	Number	Percentage
Android data collection application	Trepn Profiler	Rua <i>et al.</i> [2019]	5	29.41%
		Cañete <i>et al.</i> [2020]		
		Zhang <i>et al.</i> [2019]		
		Malavolta <i>et al.</i> [2020]		
		Elliot <i>et al.</i> [2017]		
Several Profilers	Others	Wang <i>et al.</i> [2017]	7	41.18%
		Barreto Neto <i>et al.</i> [2020]		
		Cañete <i>et al.</i> [2020]		
		Zhang <i>et al.</i> [2019]		
		Duan <i>et al.</i> [2017]		
		Oliveira <i>et al.</i> [2019]		
Android data collection application	PowerTutor	Almasri and Sameh [2019]	2	11.76%
		Mehrotra <i>et al.</i> [2021]		
Tool for command line data collection	BatteryStats	Pereira <i>et al.</i> [2021] Malavolta <i>et al.</i> [2020]	2	11.76%

beled as high and minimal contribution to the cluster labeled as low. The educational applications reside in the cluster labeled as medium. Supervised learning is applied to labeled data using algorithms such as Decision Tree, Random Forest, K-Nearest Neighbors, Rule Induction and Naive Bayes. In general, the Random Forest algorithm performed better than the others, with an accuracy of 97% and a Kappa coefficient of 95%. In the second and third positions are the Decision Tree and Naive Bayes with 93% and 82% accuracy respectively. When analyzing among the positive classifications how many were classified correctly using the precision metric, Mehrotra *et al.* [2021] identified that the Random Forest hit 100% of the instances whose label is high, 96% for the medium cluster and 94% for the low cluster.

In a study that also investigates patterns in the data, Barreto Neto *et al.* [2020] aim to automatically build machine-learning models to estimate energy consumption based on the usage pattern of each user. A total of 30 smartphone characteristics were collected through a specific application, such as screen brightness, RAM memory, processor, current, voltage, Wi-Fi and cellular network, among others. This was made possible by using Android's own methods such as the Battery Manager API through Linux's `procfs` command. The calculation to obtain the value of energy consumption around the power was based on the multiplication of the current and voltage values considering the time of one second to allow the use of this definition. The accurate current values were obtained by the virtual file `batt_current_uA_now` and the instantaneous voltage by the property `extra_voltage`. The automatic model construction methodology is organized in five steps: (i) reading the database and removing outliers using the Local Outlier Factor technique; (ii) choosing the best features to train the model – done by iterating on the 30 features performing a cross-validation division of 10 on each one to evaluate the error estimate. The algorithms used are Random Forest, Support Vector Regressor, Stochastic Gradient Descent Regressor, Neural Nets, Deep Multilayer Perceptron and Long short-term memory. The characteristics are ranked through the error of the medians produced

by each one; (iii) construction of 30 models using time series cross-validation – cross-validation with time series split is applied. Then the 25th percentile and 75th percentile of the 30 test scores are calculated. The interquartile range (iiq) (given by 75th - 25th) will determine a cut-off point such that test scores below  $median - 1.5 \times iiq$  and above  $median + 1.5 \times iiq$ ; finally in (iv) the most appropriate model is chosen among the 30 generated through the adjusted training base for the samples filtered in the previous step. For this, the Nemenyi test is used to rank the scores and choose the one with the minimum error.

### 3.2 Frameworks for analysis and energy consumption management

The work presented by Duan *et al.* [2017] details a framework for energy management based on smartphone usage patterns. The collector consists of three components: (i) Data Collector, (ii) Data Analyzer, and (iii) Decision Maker. In total, five features are monitored and collected by the Data Collector: the number of screen switches, duration of the screen on, number of foreground application switches, the volume of data transfer, and battery level. An application was developed to collect the features every minute. The power consumption (P) metric is obtained from the battery level (N) measured in mW as  $P = \frac{v \times c \times N}{100 \times t}$  with V being the average voltage during the time T in seconds and C the battery capacity. Duan *et al.* [2017] analyze the frequency of use on devices by the number of screen changes and foreground applications. Thus, users are classified as active and non-active depending on the period of change. An active user is expected to consume more energy than a non-active user.

A set of 22 usage patterns was collected for 24 hours, half being generated by the Samsung Galaxy S II smartphone and the other half by the Samsung Galaxy Note III. During data analysis, the K-means algorithm was used to divide the data into K clusters. In the experiments, users were divided into active and non-active, while smartphones were classified into low-power and high-power consumption. Half of

each group was classified as active and non-active. However, there are two exceptions. Active user #10 has less power consumption than all non-active users for the same device. Non-active user #15 has higher power consumption than users #17 and #21 on the same device. There are two reasons for this: the smartphone's energy efficiency is determined by the manufacturer and; the user's usage profile. In a last experiment to validate the hypothesis that non-active users consume more energy than expected, three test cases were analyzed: Case one – All background user services were interrupted; Case two – The same as Case one, but with the execution of tests changing the screen every 30 seconds for five minutes of duration; Case three and Case four consisted of installing, respectively, a messaging application and an email client based on cases one and two. For example, the power consumption on Samsung Galaxy S II for Case four is 483mW which is 5% more than Case three, 57% more than Case one on the same device, and up to 160% more than in Case four on the latest Galaxy Note III smartphone. The conclusion is that systems with services in the background consume more energy than without the execution of these services and the type of smartphone has a great influence on energy consumption. Finally, based on user type and power consumption, decision-making can notify the non-active user with high power consumption on the device. Non-active users can receive recommendations such as turning off any unused network interfaces, adjusting the screen brightness to an acceptable level, and stopping background services when the device remains in a sleep state. If an old device is the reason, it is advisable to replace it with an energy-efficient device or install a new battery.

Rua *et al.* [2019] present the GreenSource infrastructure to analyze energy consumption in Android applications. The infrastructure contains (i) a database of open source applications for Android; (ii) AnaDroid – a framework for instrumenting code and monitoring power consumption and; (iii) a repository with metrics obtained from AnaDroid. The database contains 609 Android applications obtained from the MUSE repository, Lopes *et al.* [2017], by filtering projects that contain classes with specific libraries for Android. A fundamental part of monitoring energy consumption is the application of the profiling technique. For this, Rua *et al.* [2019] use the Trepp Profiler application. The AnaDroid framework initially deals with the source code of applications in which a code analysis is performed to generate metrics such as the use of APIs by methods and classes, the number of declared variables, functions, and their respective parameters. Then AnaDroid integrates Trepp into the source code through instrumentation. This is done to allow control of application execution through API calls to monitor power consumption at runtime. With the fully instrumented source code, the app installer generation starts in the APK file format. Then, tests in 20 pre-defined usage scenarios were performed to exercise the applications. The test uses the Android Application Exerciser Monkey to allow automatic testing in any application. Finally, the results of the tests performed generate metrics that are stored in a database. GreenSource database has 281.811 static and dynamic metrics, of which 39.375 are test metrics, 241.128 are method metrics, and 1.308 are class metrics.

On the other hand, the work of Di Nucci *et al.* [2017] pro-

poses PETrA, a framework to calculate energy consumption with the ease of software implementation and the accuracy of hardware implementation. Its methodology is divided into two stages: simulation and tool operation.

For simulation, PETrA uses the Android Monkey program to simulate user interaction with applications, in the form of clicks, taps, and gestures. To collect battery consumption data, PETrA uses tools such as *dmtracedump*, *Batterystats*, and *SysTrace*, all available through Google's Project Volta. The framework depends on the installation of its own Android app on the smartphone where data collection will be applied. This application performs all simulations and collects interactions. At the end of execution, Android Monkey stores the collected data and test information on a CSV file. Finally, the framework, previously installed on a computer, summarizes the results. Additionally, users can view the top five most power-consuming methods and visualize the energy profile of the analyzed app in the form of graphs.

To compare performance and determine the framework accuracy, a comparative study was done by measuring the energy consumption of the same app, in the same conditions, with PETrA and hardware power monitor Monsoon. The average estimation error was 4% in relation to the values measured by the Monsoon monitor. In 95% of the errors, the difference between the real value and the value produced by PETrA was at most 5%.

Another framework, E-Spector, is presented in Wang *et al.* [2017]. This online framework for energy consumption analysis can estimate the runtime consumption of the application under test (AUT) for both foreground and background applications, requiring only the installation of an Android application. It also creates a power curve for the AUT and shows all processes and services running at any point in the application's energy curve. The app collects information such as network traffic data, screen brightness, battery, CPU usage. These collections are used to calculate the energy dissipation of the smartphone through the V-Edge Energy model of Xu *et al.* [2013] based on voltage and a regression method is used to build the model for energy estimation. To determine the framework accuracy, an experiment was carried out with Monsoon Power Monitor, in which results obtained by the monitor were compared to the results measured by the E-Spector. The E-Spector presented an accuracy of 10%, with an approximate overhead of 4%.

For the purpose of performing multiple tests, Malavolta *et al.* [2020] present Android Runner (AR), a framework to facilitate the execution of experiments with native Android apps and web apps. Android Runner is a framework that allows replicability and customization of different experiments to analyze different features and metrics at runtime, such as power consumption, CPU, and memory usage. Among the advantages of using AR are the fully automatic, customizable and replicable execution of the framework. Furthermore, AR is independent of the type of profilers used, which can be either hardware (Monsoon Power Monitor) or software (Trepp). It is possible to combine multiple profilers in a single experiment. AR demands in its input configuration scripts for the execution of the experiment such as repetitions, duration time, list of apps, and the list of devices; the application installer is in the APK format. The framework out-



puts the collections and measurements of energy consumption measured by the *batterystats* Android plugin, as well as logs on the execution of the experiment. In terms of accuracy for AR, a set containing 27 benchmarking applications was created, each one performing a stress test on an Android feature such as CPU, screen, and GPS. The application ran for three minutes, and each round was repeated 30 times with a two-minute interval between successive collections. The established accuracy was 95%.

### 3.3 Usage patterns of mobile devices

Regarding the study of patterns, Guo *et al.* [2017] present a study with data from 80,000 smartphone users, carried out through the Energy Saver battery manager application, which collects metrics related to how the battery decays according to the use of the applications. One of the first known works dealing with this type of collection was Oliner *et al.* [2013] which later served as inspiration for the collection method used in Pereira *et al.* [2021] described in this mapping. It is the first large-scale study in its category. So far in the literature, only small or medium-scale studies had been carried out. The application architecture has two main components, *EventListener* and *EventServer*. The first collects data every time a significant event occurs on the smartphone, such as battery decay, screen brightness change, or an application opening. The second transfers the collected data to a remote server.

After filtering the data, the program calculates the total usage time of each application and the total battery decay during the analyzed interval. Based on the usage time, it calculates the energy consumption of each application. The metrics used to calculate battery life are the average battery discharge rate given by dividing the total battery level drain divided by the total discharge time, and battery life given as a percentage, obtained by dividing 100% by the average battery discharge rate. Regarding the consumption rate for applications, the calculation considers that the average rate of energy consumption is given by the total battery consumption divided by the total time of use. The results were validated by means of an experimental setup with the Monsoon Power Tool software. The results show that there are users with a heavy pattern of use – an average of 8.6 hours per day – while users with a light pattern of usage spend less than one hour. The average usage time for all users in the study corresponds to 3.79 hours per day. It was also found that the standby state uses far more energy than expected, accounting for 45% of the smartphone's total power consumption in one day.

In a similar line of reasoning, Pereira *et al.* [2021] aims to understand how applications, operating systems, hardware, and usage habits influence battery consumption. The authors developed an app, BatteryHub, and made it publicly available on the official Android app store. App user data was periodically sent to a central unit, helping to generate a database with more than 700 million process entries and energy-related data. Among the data collected, there is information about the applications such as name, version, and package; battery details like the voltage, temperature, health, and charging status; CPU status including utilization, uptime,

and sleep time; general device information such as manufacturer, model, and operating system version. The database is public and has 23,600,501 records until the last update reported in the paper. In addition to the database, two other contributions are Farmer – a REST API to perform quick database queries through JSON files that contain fields related to database features and; Lumberjack – a command-line application that allows you to flexibly query the database through on-demand queries to allow rapid support in prototyping queries by applying different filters and parameters.

This database was submitted to metric analysis that revealed trends in characteristics related to energy consumption, as well as new and promising lines of research. Trends pointed out that: smartphones in underdeveloped countries (Equatorial Guinea, Zimbabwe, Botswana, Burundi, Benin) tend to discharge quickly and charge slowly. One exception was Greenland, where smartphones charge and discharge slowly. A possible cause pointed out by the authors was low ambient temperatures. Pereira *et al.* [2021] also found that the difference between smartphones of distinct models is greater than the difference between smartphones of different brands and also that, in general, there is a trend towards more energy efficiency for newer versions of Android, reaching peak efficiency in Android Oreo operating system.

Apps with location and video services such as Google Maps, YouTube, and VidMate have higher consumption trends, while apps such as Facebook, Messenger, and Chrome have lower consumption trends. An important paper finding was that Facebook Lite consumes more battery than Facebook's main app. This happens when running Facebook Lite takes place on weaker smartphone models. Also according to Pereira *et al.* [2021], activating or deactivating location and Bluetooth makes no difference, and the energy-saving mode is only effective when the smartphone is charging. The authors have found that the best network connection is Bluetooth tethering and that there is no difference between Wi-Fi and mobile network. Pereira *et al.* [2021] complement that the metrics did not indicate an exact energy consumption for each sample, application, or process, only trends to be investigated. Furthermore, it is important to mention that these results were obtained based on the typical use of these smartphones. They do not mean that there is no difference in a lab environment. Considering the ways smartphones are normally used, there is not much difference between these two approaches because other factors such as battery quality, smartphone age, and context of use can have a much greater weight on energy consumption.

### 3.4 Techniques for reducing energy consumption

Among the studies that deal with techniques to reduce energy consumption there is the PIFA, an intelligent CPU frequency tuning framework introduced by Zhang *et al.* [2019], which regulates CPU frequency according to the app execution phase. The phases can be characterized as different stages of execution of an application. For games, there are usually two phases, according to Zhang *et al.* [2019], the phase operation menu and the actual phase when playing. Applications can have multiple phases performing different func-

tionality in addition to different latencies for performance metrics. The PIFA approach combines at first offline analysis where previously collected data on energy consumption are fed to the clustering algorithm (identification phase) to separate the data into clusters so the result is a classification model; in a second moment, online analysis is used to perform a decision according to the monitoring of smartphone characteristics such as CPU, GPU and memories to adjust the frequency. With phase identification, PIFA can adjust the voltage and frequency of the core at which the application is scheduled to run according to the “sweet frequency”. In an experiment conducted with 18 apps for real use in the world where 13 were downloaded from the official Android store (Play Store), and five were obtained from open-source stores, the analysis of the results shows that PIFA is more effective in reducing energy consumption, more than 30% reduction for most apps, in addition to keeping latency at a desired level of performance with considerably small overhead, less than 5%.

Another work that aims to make adjustments to some smartphone features, now in real-time applications, is described by Cañete *et al.* [2020]. The work deals with reducing energy consumption based on user behavior by dynamically adapting running applications. The focus of the article, in addition to measuring energy consumption, aims to ensure that the consumption of the adaptation mechanism does not increase when compared to the benefits of adaptation, as the mechanism can reduce battery drain by up to 20%. One of the key techniques employed is dynamic proxies – wrappers that allow adding or modifying functionalities, allowing the adaptation of applications. The interest is in the reconfiguration of the base functionality of the applications and not just in reusing functionalities such as method authentication and algorithm encryption.

The architecture of the proposed solution is divided into (i) *handlers*: to intercept calls to objects; (ii) *monitors*: to obtain context information and control a specific set of variables from apps; (iii) *context*: to manage information about the app’s current context; (iv) *analysis*: to make decisions about the most appropriate functionality based on the application context and; (v) *Adaptation rules*: contains a set of expressions (the rules themselves) that define how the adaptation mechanism should act based on the current context. The validation step used the Trepp Profile tool (mentioned previously) and the GreenScaler which works mainly in collecting data such as the number of CPU jiffies<sup>2</sup> and the colors used on the screen.

The experiment was carried out in two scenarios. In the first one, the amount of information displayed on the screen and the number of requests for data/functionalities were adapted to show information compatible with the user’s profile. For example, in this adaptation, the WhatsApp application loads, and when starting, all the conversations of contacts are considered friends. This is justified because the user frequently chats with a sample of the contact population which allows for reducing the number of chat requests which can save energy without affecting the user experience. In the

second scenario, the adaptation consists of adding new functionality and being able to modify an existing one, according to the user’s profile. For example, data compression can be applied to photos, videos, or pdf in communication applications such as Gmail to send a smaller file which can result in battery savings.

Cañete *et al.* [2020] obtained a benefit above 20%, from 168 J to 134 J, using a solution based on Internal and External Proxies. The introduced overhead is minimal representing a percentage between 0.58% and 2.51%, an increase from 1 J to 4 J of energy consumed. In the second scenario, a video chosen by the user was compressed and sent over the network. The overhead produced was between 0.43% and 1.39% and produced an increase between 3 J and 9 J of energy consumption, which meant an increase of only 8.87 J in the worst case.

Contrary to the papers cited in this subsection that try to reduce energy consumption locally, Harihar and Sukumaran [2018] transfers part of the local computing to the cloud through the Mobile Cloud Computing (MCC) framework. It dynamically selects the code suitable for offloading. Therefore, it is necessary to analyze whether the cost of transferring code data during offloading does not consume more battery than keeping the code in the device. Moreover, the authors sought to collect runtime and data transfer parameters.

Within that framework, runtime and data transferred through invocation methods are analyzed for all public method calls. To evaluate the impact of the framework on the transfer of classes, the size of input objects for each class is measured using the *ObjectOutputStream* and *ByteArrayOutputStream* classes. These calculations are written to a binary file and sent to the cloud when an internet connection is detected. Data is then mined and used to predict class execution parameters, serving as the basis for code partitioning. Through a time series decomposition, data is also divided into three categories: (i) seasonal data, tied to cyclical variations of data, (ii) trend, which shows the increasing or decreasing trend of the data, and (iii) residuals. This decomposition can reveal important patterns of usage habits.

To test the framework, a custom app called Go was created and used for 30 days by an unknown number of volunteers. A reduction in battery consumption of 3.5% was estimated. Furthermore, the work demonstrated the superiority of dynamic methods over static methods to estimate battery gain. It was also possible to verify the existence of user behavior patterns, keeping an extra processing load due to instrumentation of only 0.12%. All MCC processing is done in the cloud, saving battery power.

In a study with a different approach from those presented in this study, but with significant results, Linares-Vásquez *et al.* [2018] present an approach to reduce energy consumption by optimizing the color palette used in the graphical user interface (GUI). The Gui Energy Multi-objective optimization for Android apps (GEMMA) approach generates color palettes using the multi-objective optimization technique to produce color compositions that optimize energy consumption and contrast while using colors that are consistent with respect to the original color palette. To validate the quality of the color schemes recommended by GEMMA, optimized GUIs were generated for 27 applications: 22 free applica-

<sup>2</sup>Basically every time the CPU timer interrupt occurs the value of the variable ‘jiffies’ is incremented

tions available on Google Play and five commercial applications developed by Italian companies. The visual attractiveness of the derived GUIs for the 27 apps was evaluated by 104 mobile app users in an online survey. In addition, developers and managers from three Italian companies were asked to provide feedback on the GUIs generated by GEMMA for their applications. In the experiment with 104 participants, the color perception was evaluated. For this, three questions were elaborated: (i) “What is your current position?” (ii) “How often do you use mobile apps?” (iii) “Is GUI color rendering important to you in the overall judgment/rating of a mobile app?”. Eighty-six participants reported using apps frequently and 18 participants reported using them occasionally. Regarding the importance of GUI color compositions in the overall judgment of a mobile app, 29 participants reported that it is “very important”, 57 responded that it is “important”, and only 18 participants think it is “not very important”. Participants were also asked about the importance of color compositions to ensure they care about color compositions and provided helpful responses. None of the participants answered “nothing important”. Linares-Vázquez *et al.* [2018] employed energy monitors to measure the actual energy saved by adopting the GUIs generated by GEMMA. Overall, GEMMA was rated on four dimensions: (i) the ability to optimize the three objectives, (ii) the actual energy saved by adopting the generated GUIs, (iii) the extent to which potential users consider the color choices sufficiently acceptable, and (iv) the extent to which the original developers of commercial applications would be willing to take the GEMMA recommendations into account. Regarding the applications optimized by GEMMA, in terms of battery life gain, the original application and the lowest power solution recommended by GEMMA were compared. Among the apps analyzed are Privacy Friendly Dicer, Learn Music notes, Simple Deadlines, Tasks, and Tasks: Astrid To-Do List. The results indicate that there is a clear reduction in energy consumption when using the color palette recommended by GEMMA. Per-app reduction percentages are Privacy Friendly Dicer (33.48%), Tasks (32.44%), Tasks: Astrid To-Do List (32.13%), Simple Deadlines (29-34%) and Learn Music Notes (21.74%). The application performance improvement is not marginal and varies between +28.22% and +50.27%.

Oliveira *et al.* [2017] investigate the impact of some of the most popular development approaches on the energy consumption of Android apps. This study uses a testbed of 33 different benchmarks and 3 applications on 5 different devices to compare the energy efficiency and performance of the most commonly used approaches to develop applications on Android: Java, JavaScript, and C/C++ (Native Development Kit – NDK). Oliveira *et al.* [2017] compared the power consumption and performance of 33 benchmarks developed by several authors of the Rosetta Code and The Computer Language Benchmark Game (TCLBG) to compare the use of different languages in the development of Android applications. Multiple versions of each benchmark were run on several different mobile devices, measuring runtime and power consumption. It was found that for 26 of the 33 benchmarks analyzed, the JavaScript versions exhibited lower power consumption than their Java counterparts; and only six Java

versions of these benchmarks outperformed their JavaScript counterparts. This indicates that Java may not be the most appropriate language considering energy efficiency. For the TCLBG benchmarks, the comparison took into account the Java, JavaScript, and C++ versions, running them on five devices with different characteristics, obtaining similar results. As a result, it was found that although there are some slight variations in performance, the power consumption relationship between Java, JavaScript, and C++ remains similar across devices, with JavaScript having an advantage over other approaches in terms of power consumption. It can be seen that using the NDK did not improve performance in this scenario.

In another experiment, Oliveira *et al.* [2017] tried to reduce energy consumption by transforming a native application into a hybrid. Four open-source apps were redesigned. Each application was written in Java and the authors made parts of them run in JavaScript and C++. The purpose of the experimentation was to analyze whether using these approaches together with Java impacted performance and power consumption. Different models were analyzed to invoke Javascript and C++ snippets using Java code and measured power consumption in all cases. The results indicate that it is possible to save energy using this hybrid approach - for one of the applications, a hybrid version using a combination of Java and C++ consumed 0.37 J, under a given workload, while the original version written entirely in Java consumed 32.92 J. In terms of results, there is no superior programming language considering all aspects evaluated. The results indicate that JavaScript saves more energy but is slower compared to Java and C/C++. Therefore, the hybrid approach emerges as an alternative to deal with energy-saving and execution performance dilemma to optimize the performance of Android applications.

In a later work, Oliveira *et al.* [2019] investigates in the Java language which collections have consumed more energy and which alternative collections can replace the original ones to obtain gains in energy consumption. For this Oliveira *et al.* [2019] developed the CT+ tool that implements an energy-aware approach capable of producing recommendations for energy savings through the use of alternative implementations for Java collections in different parts of the system, whether for desktop or mobile. The CT+ tool works in two steps. First, it automatically runs multiple micro-benchmarks independently of the application for the Java collections available in the runtime. With the data from these micro-benchmarks, energy consumption profiles are built for the implementations of these collections. A power profile provides a grade that can be used to compare the power consumption of different implementations. After building the energy profiles, the second step consists of performing a static analysis on the system to be optimized, in order to estimate the frequency of use of multiple collection operations. Finally, the most efficient implementation for each energy variation is recommended. So CT+ can optionally apply the recommended changes automatically. The experimentation considering Android smartphones was performed on Samsung S8, Samsung J7, and Motorola G2 models in which 4 applications were tested: Commons Math, Fast-Search, Google Gson, and PasswordGen. The effectiveness

of the recommendation varied among the analyzed devices. Modified versions of the PasswordGen app on the Samsung S8 and J7 smartphones have seen significant improvements. The original versions consumed 4.7% and 17.34% more energy than the modified versions. The recommendation for this device is to replace ArrayList with FastList. For the Motorola G2 device, there were no recommendations, so consumption remained the same. The Gson app had a significant improvement of 5.03% on the J7 smartphone. For the Samsung S8, the improvement was only 0.95%. With respect to J7, the ArrayList collection can be exchanged for FastList or NodeCachingLinkedList. The Commons Math app had the most inconsistent results, as the original app consumed 11.31% more power on the Samsung S8, however for the G2 and J7 smartphones less power than the modified versions, 1.2% and 0.33% respectively.

## 4 Discussion

This mapping study identified the leading techniques and methods used when it is necessary to analyze the impact on battery consumption in smartphones. It was divided into: (i) collection of battery-only information such as voltage and/or electrical current that is easy to access. The collected values allow for obtaining other metrics such as the power and total energy consumed by the smartphone; (ii) metric collected by tools that, for this purpose, are called *profilers* such as Trepn, BatteryStats, and PowerTutor which provide more comprehensive captures such as CPU usage, Wi-Fi signal level, functionalities in use such as GPS, Bluetooth, mobile data, the screen on; and (iii) Artificial Intelligence techniques such as decision tree, random forest, multilayer perception, and others focused on analyzing data through predictive models capable of determining device usage behavior and even reducing energy consumption.

Therefore, it is possible to list several limitations applied in the scope of the studies that this review understands as a potential gap to be investigated. Amongst the limitations are a low amount of information and analysis about applications running in the background, and controlled experiments without the presence of real users as in static analysis. Furthermore, access to the source code of applications is often necessary for certain research studies. However, this requirement is rarely met due to the closed-source code nature of the most popular applications developed by major companies in the fields of instant messaging, streaming, and gaming. The next paragraphs describe the limitations and future work of the analyzed papers with a concise analysis based on the four types of studies presented in **Table 5**.

### A. Controlled studies to determine energy consumption patterns in Android apps

Almasri and Sameh [2019] discussed a star-based scoring system to indicate which apps can consume more power based on characteristics extracted from the manifest file present in Android apps. The paper uses a metric scale based on energy consumption to provide the stars. As a proposal for future work, an algorithm such as k-nearest neighbors can be added to associate an application with a star based on the characteristics extracted from the manifest file and the cur-

rent apps considered neighbors by the algorithm.

In contrast to Almasri and Sameh [2019], the subsequent papers, although presenting different approaches, deal with the application of machine learning to analyze energy consumption. Mehrotra *et al.* [2021] discussed the use of machine-learning algorithms such as decision trees, k-means, and k-nearest neighbors for multiclass classification. A predominant factor in this type of method is to establish the label of the problem. While the absence of a label in the authors' collected database was resolved using unsupervised k-means, the database-created label is a pseudo-label. An alternative approach to avoid a pseudo-label is to employ the binning technique, which defines the label based on the energy consumption values gathered. Furthermore, given that k-means were applied in the labeling stage, it would be interesting to consider it in the multiclass classification stage along with the previously defined algorithms. Another limiting factor is the rules of the decision tree which are obtained to represent the smartphone usage profile. These rules can be used to add value to a study that aims to identify possible actions to be taken by an application, for example, to manage energy consumption.

During the preprocessing stage, Barreto Neto *et al.* [2020] identified potential outliers and determined the most significant features of the database to utilize in their experiment. In terms of outlier detection and smoothing, the Hampel filter technique is worth exploring. As for feature selection, the Boruta algorithm based on Random Forest is highly relevant. Moreover, for data division, the one-step time series cross-validation approach was employed. However, conducting new experiments using the multi-step approach would be interesting, as it would consider k-set-ahead steps in the future instead of only one step.

Elliot *et al.* [2017] examine the correlation between energy consumption and media application usage, particularly in regard to audio, video, and social media applications. To improve the study, conducting additional experiments with a larger sample size could help to transform the interview questionnaire into an effective tool for identifying the user profile associated with energy consumption. One potential approach to achieve this is to adapt the interview questionnaire into a dataset suitable for machine-learning algorithms, such as k-nearest neighbors.

In general, papers such as Mehrotra *et al.* [2021], Almasri and Sameh [2019], and Elliot *et al.* [2017] used PowerTutor or Trepn apps that are currently discontinued. A recommendation for future work, in line with the trend of more recent papers, is to develop a specific collection app for the survey, in addition to complementing the data through the use of Android's batterystats.

### B. Frameworks for analysis and energy consumption management

The study conducted by Di Nucci *et al.* [2017] involves the analysis of individual energy consumption in applications through the use of the PETRA tool, which estimates power for Android. The analysis considers the methods invoked, the elapsed time in seconds, and the energy consumption in Joules. In the end, a file in CSV format is generated with the analyzed data. Although the framework requires some configurations to carry out the experiments, there are some

limitations in the execution of the experimentation and in the supply of results, which must be improved. An instance of a limitation in the experiment is the constraint of executing a single application at any given time. It would be good to have a configurable pipeline that helps experiment with multiple apps (not necessarily in parallel). As mentioned by the authors, the batterystats program is embedded in the core of PETRA, allowing for the generation of data that includes a wealth of energy consumption information. This includes additional details on the application's resource usage, such as mobile data, Bluetooth connectivity, and the frequency of CPU core usage, which can enhance the CSV file's content.

One research conducted by Rua *et al.* [2019] introduced GreenSource, a framework designed to analyze energy consumption in Android applications. The infrastructure comprises three components: (i) an Android open-source application database, (ii) a benchmarking tool called AnaDroid, and (iii) a repository that contains the primary metrics utilized in the literature. GreenSource contains a limitation identified during the application code modification step. In this step, the insertion of methods that allow starting and stopping the collection of information associated with energy consumption uses the Trepan application. However, the application has been discontinued, which makes it difficult to reproduce the experiment on newer versions of Android. There is an additional constraint that the app is only assured to be compatible with smartphones featuring Qualcomm-branded processors utilizing the Snapdragon chipset. To overcome these limitations, a possible direction for future work is to develop a specialized application for data collection.

Wang *et al.* [2017] introduced E-spector, which is another tool that relies on code instrumentation. One distinctive feature of this tool is its ability to perform real-time analysis of energy consumption. Nevertheless, to accomplish this goal, the analyzed application must be open-source to allow for code instrumentation, which facilitates code insertion and method detection. The collection of features related to energy consumption is developed by the authors themselves, which can avoid legacy or discontinued app problems compared to the works already mentioned. Although the analysis is done in real-time and can be accessed in a web environment, the entire experiment only works locally. Potential future work should explore the option of running experiments remotely through ADB. Additionally, there is potential for analyzing energy consumption in proprietary applications, which may involve a hybrid approach using Android's batterystats tool for offline analysis. Finally, the power estimation model may be extended to encompass a broader range of smartphone models.

The framework presented by Duan *et al.* [2017] adopts a software-based approach for analyzing energy consumption, which consists of two stages: (i) data collection; and (ii) unsupervised learning-based offline analysis. Step (i) occurs by running an application for periodically capturing records. In step (ii), the k-means algorithm separates the data into clusters to generate a bi-class classification of the data. There is a classification based on users with two possibilities: active or not active; and a classification based on power consumption which can be low or high. If the energy consumption of a user belonging to the non-active group is high, then there

are indications of abnormal smartphone behavior. A limitation of this approach is the identification of such anomalies, as this takes place offline. In future work, the models trained on each smartphone can be used to perform real-time analysis, via online machine learning, in order to perform some action on the Android system to correct any abnormal behavior.

Malavolta *et al.* [2020] distinguish themselves from previous frameworks by offering the option to choose the preferred profiler application in their Android Runner (AR) tool. Different applications, native or web-based, can be configured to run in different contexts of experimentation. As there is no instrumentation to modify the source code, proprietary applications can be tested as long as the installer is in APK format. For future work, compatibility with tools based on hardware other than Monsoon can be extended and experiments can be performed with the researchers' own energy consumption collection application or another application compatible with newer versions of Android.

### C. Usage patterns of mobile devices

Regarding usage patterns, Guo *et al.* [2017] created an application that collected data from 80,000 Android devices over four weeks to calculate the energy consumption rate. For validation, the collected data were compared with the Monsoon Power Monitor tool so that the difference between the values measured by the app and the values obtained by the tool had a difference of less than 10%, proving that the method is accurate with a large scale data. However, an obvious limitation is the amount of data. If too little data is collected, the method can become inaccurate for certain devices. One possible solution to this issue is to gather data from the batterystats tool in conjunction with the existing data collection process. This tool, which is a native Android feature, provides measurements that are more accurate and can isolate energy consumption for specific features, thereby enhancing the effectiveness of the data collection.

Pereira *et al.* [2021] introduced a collaboratively constructed dataset consisting of approximately 23 million records from over 1,600 Android devices. Among the analyzes of the generated base, the work presents the PPM (percentage-per-minute) metric to obtain trends of which factors can impact energy consumption. The database can be applied to facilitate the use of the PPM metric by collecting which applications are in the foreground and background of each record collected. With this, one can have stronger evidence of what has caused the high battery consumption of Android smartphones. Moreover, machine learning models can leverage current as a new feature to produce energy consumption estimates. By multiplying current with voltage, the power associated with energy consumption can be derived when the collection interval is set to 1 second.

### D. Techniques for reducing energy consumption

In terms of techniques to reduce energy consumption, Harihar and Sukumaran [2018] minimized energy usage in their experiments by transferring application code snippets to cloud processing, thereby avoiding the overhead on smartphones. The proposed approach uses techniques such as code instrumentation, data collection related to energy consumption, and time series. An obvious limitation is in the exclusive parsing of public methods restricted to a single thread.

Measurements can generate inaccurate values. It is possible to use the Android logcat tool to monitor method calls to try to workaroud this situation, in addition to allowing a study aimed at identifying bugs between method calls. With respect to partitioning apps into two groups, as it is an NP-hard problem, heuristics based on genetic algorithms are a promising approach. Finally, the time-series prediction of the bandwidth can be inserted without too many problems in the experiments.

Oliveira *et al.* [2019] propose the reduction of energy consumption via modifications in Java language collections performed in the CT+ tool using the static analysis technique. The experiments performed on both desktops and Android smartphones differ slightly in design on each operating system guiding to some nuances as the use of jRAPL on the desktop platform. Using RAPL allows for accurate resource analysis, but is limited to Intel processors. To ensure portability, the energy consumption capture approach can be adapted to a software approach.

A previous study conducted by Oliveira *et al.* [2017] introduced hybrid methodologies for mobile application development as a means of minimizing energy consumption. The results show that there is no common language for power consumption and faster performance/processing scenarios. The analysis was limited to collecting energy and runtime data. The experiment did not address other metrics that can influence energy consumption, such as CPU usage, core frequency, network usage, and others. There is a problem of performance identified in different implementation approaches of some benchmarks depending on the adopted language. A curation would be ideal to validate and fix the database when needed. Using a smartphone differently from the others in the experiment is another crucial factor to avoid. Conducting a new experiment with adjustments to the identified problems may yield significant results for the research.

In conclusion, Linares-Vásquez *et al.* [2018] took a distinct approach to optimize applications by making modifications to the color palette of graphical interfaces. In the case of OLED screens, the object of study, the energy consumption of the pixels depends on the level of the color component rendered via the combination of red, green, and blue subpixels at different intensities in the RGB color model. Consumption becomes a double summation of the RGB intensities based on the row and column coordinates of the pixel matrix. Changes to the color palette of apps occur through the GEMMA (Gui Energy Multi-objective optiMization for Android apps) approach. A new color standard is defined by taking screenshots from the apps. The accuracy of power consumption is validated using Monsoon hardware. One aspect that requires further investigation is the method used to define colors in the approach. Choosing colors carefully is crucial to avoid hindering the reading of texts and losing the identification of GUI elements such as buttons. To improve this approach in future work, a more precise method could be applied to application GUIs by reverse engineering, taking into consideration the appropriate choice of colors for GUI elements like text fields and buttons.

## 5 Answers to Research Questions

A. *What methods and techniques are currently used to investigate the factors that impact Android smartphone battery consumption?*

Given the complexity of determining which resources directly influence energy inefficiency, whether, through the real-time collection or hardware instrumentation that requires the purchase of extra hardware or through static analysis that requires access to application source code, which is not always possible, it is observed in the literature that these options tend to be complex in terms of how to isolate and analyze a single feature and estimate its impact on the battery. Due to this, what is identified in most works as in Barreto Neto *et al.* [2020], Pereira *et al.* [2021] and Linares-Vásquez *et al.* [2018] is not isolating the features, but rather carrying out studies taking energy efficiency into account in a total. In Pereira *et al.* [2021], for example, the term “trend” is adopted when analyzing the Percentage per Minute (PPM) metric when suggesting which feature could consume more battery power. Metrics such as power acquired through current and voltage in Barreto Neto *et al.* [2020], separation of clustered features for low, medium, and high consumption as identified in Mehrotra *et al.* [2021] or adjustments to the color palette as shown in Linares-Vásquez *et al.* [2018] are presented.

Amongst 17 analyzed papers, 13 measure direct energy consumption. To know this consumption, three approaches are used. Initially, battery voltage and/or current information is gathered, which is then utilized to calculate the dissipated power and the total energy consumed during a specific time period. This time interval is typically determined by computing the difference between the timestamps of the collected samples. The power equation (Equation 1) and energy equation (Equation 2) are the most frequently employed equations. One paper refers to battery capacity in mAh and percentage of charge, while another paper computes it using automated models that rely on network baud rates, screen brightness, and CPU usage.

$$P = V * I \quad (1)$$

$$E = \int P dt \quad (2)$$

The definition of timestamps varies. In papers such as Barreto Neto *et al.* [2020], Guo *et al.* [2017], Almasri and Sameh [2019], Di Nucci *et al.* [2017], Harihar and Sukumaran [2018], it is defined as the shortest time in which there is no change in smartphone components or variations of this rule. Barreto Neto *et al.* [2020] define a set of rules for the definition of an interval slice: i) the sample is from the same device, ii) the changes in the battery are in the same direction, either charging or discharging and iii) always within a variation limit [-2, 2].

The second approach, used in seven papers (see Table 7), is to collect energy consumption information through specialized software. For this purpose, five used the Trepp Profiler app and two used the PowerTutor app; BatteryStats, Battery Linux profcs, Android Power Profiler, Project Volta, GreenScaler, and DroidWalker are also cited in the total of the seventeen articles.

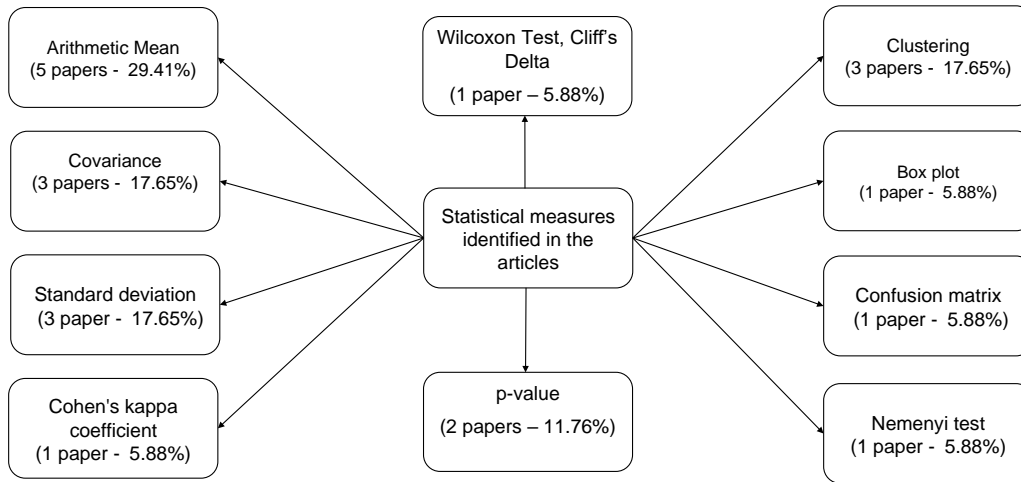


Figure 3. Most currently used statistical analysis techniques.

Although recent works like Malavolta *et al.* [2020], Rua *et al.* [2019], Almasri and Sameh [2019] use Treprn or PowerTutor as a data capture mechanism, it is important to highlight problems when using these applications, PowerTutor is from mid-2011 and mid-2015 Treprn, which makes compatibility and smooth operation difficult on smartphones with more modern Android hardware and system. Articles like Di Nucci *et al.* [2017] make use of tools from Project Volta such as Batterystats, which is interesting because Google keeps these tools up to date for current versions of Android. Other works develop their own applications, see Barreto Neto *et al.* [2020] and Pereira *et al.* [2021], making use of Android libraries that allow the monitoring of functionalities to perform the collection. What can be seen are two paths for real-time collection: using a collection application published in the literature although it is not updated or developing an application to perform data collection. The second way is more effective and accurate to collect data. Details on how to monitor and collect each resource are available in works by Barreto Neto *et al.* [2020] and Pereira *et al.* [2021].

There is also a significant number of statistical techniques applied to battery consumption analysis. For the collection and processing of battery-related information, the arithmetic mean is widely used. Almasri and Sameh [2019] and Dai *et al.* [2020] also use variance and standard deviation for large-scale generalization of battery traces. Oliveira *et al.* [2019] use standard deviation, but also evaluate p-value and mean to compare application performance. In a study that involved improving power consumption through color adjustments in the GUI, Linares-Vásquez *et al.* [2018] made use of techniques like the Wilcoxon Test and Cliff's Delta. Studies involving machine-learning algorithms also perform statistical analyses, the main one being the clustering of samples by similarities and clustering by K-means. Barreto Neto *et al.* [2020] uses the Nemenyi Test. Pereira *et al.* [2021], for selection of better consumption modeling algorithms, use box plot charts and confusion matrices. The results are presented in detail in Figure 3.

Regarding machine-learning algorithms, several models were used as shown in Table 8. The most common technique was clustering, present in 17.65% of the works, followed by Bayesian Classification and Decision Tree algorithm. The

Table 8. Identification of Machine-Learning techniques in the evaluated articles.

AI Techniques	Percentage of Occurrences
Clustering	17.65%
Bayesian classification	11.76%
Decision tree	11.76%
LSTM	5.88%
Rule Induction	5.88%
Support Vector Regressor (SVR)	5.88%
Multilayer Perceptron Neural	5.88%

authors in Pereira *et al.* [2021] employ both J48 and Random Forest algorithms, while in Barreto Neto *et al.* [2020], only the Random Forest algorithm is utilized, and a Multilayer Perceptron Neural model algorithm plays a crucial role. Furthermore, crucial artificial intelligence techniques applied include the rule induction algorithm and Support Vector Regressor (SVR). The AI techniques are shown in Table 8 and Figure 4.

B. What are the biggest impact factors on Android smartphone battery consumption?

Eight of the analyzed papers performed a controlled study or data analysis of real users. Table 9 shows obtained results. The biggest impact factor found in the studies is the user profile itself. Barreto Neto *et al.* [2020] carried out an experiment in which two different videos were played on the same device under the same conditions. The result was different energy consumption. Duan *et al.* [2017] showed, also through experiments, that active usage time is one of the main factors for the increase or reduction of battery life. These experiments characterize the relevance of user behavior in battery consumption.

Additionally, the authors also cite background applications and services as a primary cause of battery drain. Despite the importance of this factor, only two studies carried out experiments focused on it, which may reveal a literature gap. Elliot *et al.* [2017] performed an experiment that demonstrated the high power consumption of background services by running the same application with four different levels of services active in the background. The level with more ac-

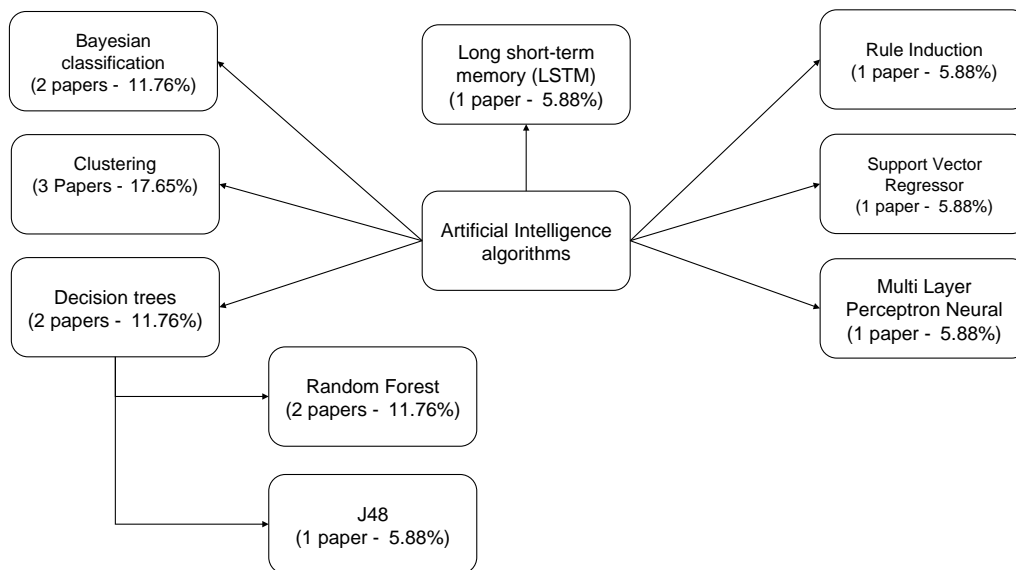


Figure 4. Artificial intelligence algorithms implemented in the studied publications.

tive services had a much higher consumption than the level with less active services. Also mentioned as causes of battery drain are the inefficiency of old hardware in Guo *et al.* [2017], Di Nucci *et al.* [2017]; and standby mode. In an experiment carried out by Dai *et al.* [2020], it was shown that after the user turns the smartphone off, a delay happens of about six seconds until the current level decreases, explaining the continuity of background services for some time, even after the smartphone enters standby mode. Furthermore, according to Dai *et al.* [2020], some applications keep the smartphone on, even if the screen is off, draining the battery.

Table 9. Summary of factors that affect the performance of devices and their occurrence in evaluated papers.

Factors of greatest impact	Percentage of Occurrence
User profile	23.53%
Mobile network and Wi-Fi	17.65%
Applications and services in background	11.76%
Old hardware	11.76%
Standby mode	11.76%

C. What are the top recommendations for reducing Android smartphone battery consumption?

Only two articles provide a direct recommendation on how to reduce Android smartphone battery consumption. Other studies, despite carrying out analyses on high-impact battery elements, do not provide direct recommendations or are limited to identifying the factor with the greatest energy consumption. The papers’ conclusions identify the following indirect recommendations as significant: (i) optimizing and investigating background activities during standby mode; (ii) advising researchers to focus their research on the small fraction of frequently used applications that consume more energy; and (iii) turning off unused network interfaces, reducing screen brightness, and stopping background services when the smartphone is in standby mode. Another suggestion is not to use smartphones with old and ineffi-

cient hardware and buy new devices regularly. For this to be feasible, the manufacturer could inform the user about these requirements. Regarding the development environment used by professionals, it is recommended to update the hardware/software platform, avoid inefficient hardware platforms and applications, and optimize system settings.

Furthermore, it is crucial for developers to enhance communication with users about ways to conserve battery life and protect the environment. In addition, developers should always make an effort to incorporate green code practices to reduce battery consumption. Another important step in the decrease of smartphone energy consumption is to carry out research intending to keep the evolution of battery technologies in sync with the evolution of smartphones.

## 6 Conclusion

This paper presented a Systematic Mapping Study aimed at identifying metrics, approaches and recommendations to reduce energy consumption on Android smartphones. All steps that constitute a review, such as elaboration of the protocol, conduction, extraction and, finally, the summarization were addressed in order to allow the replicability of this SLR and collaborate in future research.

To validate the first stage of article selection, and thus avoid the introduction of biases, the Kappa criterion of pairs agreement was adopted through the calculation of Cohen’s Kappa coefficient, whose value for this review was 0.815, which represents an almost perfect agreement among researchers [Perez, 2020].

It was noted that user habits tend to be the most responsible for energy consumption, as time of use, preferred apps and custom settings are determining factors in battery drain. There is consensus in the literature about the diversity of smartphone user options and their energy impact. In this context, the exploration of future works should maintain focus on user profile analysis, especially research with large databases collected in use. The analysis of more recent pub-



lications made it possible to notice an increase in the application of artificial intelligence techniques to automatically adapt apps to the context of use in order to save energy. These techniques are also being used for mining databases with energy metrics, in order to seek correlations between factors and battery level for optimization possibilities. The main focus of this work is, in fact, identifying and summarizing results aiming not only to answer the three elaborated research questions, but also to allow the discovery of gaps to be investigated in the research field of energy consumption in Android smartphones.

## Declarations

## Acknowledgements

This research was carried out with the support of the Coordination for the Improvement of Higher Education Personnel – Brazil (CAPES) – Funding Code 001 and, as provided for in Arts. 21 and 22 of Decree No. 10.521/2020, was partially financed by Motorola Mobility Comércio de Produtos Eletrônicos Ltda and Flextronics da Amazônia Ltda, pursuant to Federal Law No. 8.387/1991, through agreement No. 004/2021, signed with ICOMP/UFAM.

## Authors' Contributions

The authors confirm their contribution to the paper as follows: study conception and design: Edwin Monteiro, Helena Cavalcante, Raimundo Barreto, and Rosiane de Freitas; data collection, analysis, and interpretation of results: Edwin Monteiro and Helena Cavalcante; draft manuscript preparation: Edwin Monteiro. All authors reviewed the results and approved the final version of the manuscript.

## Competing interests

The authors declare that they have no competing interests.

## Availability of data and materials

Data can be made available upon request.

## References

- Almasri, A. and Sameh, A. (2019). Rating google-play apps' energy consumption on android smartphones. In *2019 2nd IEEE Middle East and North Africa COMMUNICATIONS Conference (MENACOMM)*, pages 1–6. DOI: 10.1109/MENACOMM46666.2019.8988554.
- Barreto Neto, A. C. S., Farias, F., Mialaret, M. A. T., Cartaxo, B., Lima, P. A., and Maciel, P. R. M. (2020). Building energy consumption models based on smartphone user's usage patterns. *CoRR*, abs/2012.10246. DOI: 10.1016/j.knosys.2020.106680.
- Cañete, A., Horcas, J.-M., Ayala, I., and Fuentes, L. (2020). Energy efficient adaptation engines for android applications. *Information and Software Technology*, 118:106220. DOI: 10.1016/j.infsof.2019.106220.
- Dai, Z., Wang, W., and Wu, Y. (2020). Static energy consumption analysis for android applications. *IOP Conference Series: Earth and Environmental Science*, 512:012011. DOI: 10.1088/1755-1315/512/1/012011.
- Di Nucci, D., Palomba, F., Prota, A., Panichella, A., Zaidman, A., and Lucia, A. (2017). Petra: A software-based tool for estimating the energy profile of android applications. *International Conference on Software Engineering*. DOI: 10.1109/ICSE-C.2017.18.
- Dick, Zhuoqing Morley Mao, L. Y. (2011). Power tutor description. Available at: <https://ziyang.eecs.umich.edu/projects/powertutor/> Accessed: 2021-11-24.
- Duan, L., Lawo, M., Rügge, I., and Yu, X. (2017). *Power Management of Smartphones Based on Device Usage Patterns*, pages 197–207. Springer. DOI: 10.1007/978-3-319-45117-6\_8.
- Elliot, J., Kor, a.-l., and Omotosho, O. (2017). Energy consumption in smartphones: An investigation of battery and energy consumption of media related applications on android smartphones. *International SEEDS Conference 2017*. Available at: [https://www.researchgate.net/publication/319954899\\_Energy\\_Consumption\\_in\\_Smartphones\\_An\\_Investigation\\_of\\_Battery\\_and\\_Energy\\_Consumption\\_of\\_Media\\_Related\\_Applications\\_on\\_Android\\_Smartphones](https://www.researchgate.net/publication/319954899_Energy_Consumption_in_Smartphones_An_Investigation_of_Battery_and_Energy_Consumption_of_Media_Related_Applications_on_Android_Smartphones).
- Google (2021). Battery stats description. Available at: <https://developer.android.com/topic/performance/power/setup-battery-historian> Accessed: 2021-11-24.
- Guo, Y., Wang, C., and Chen, X. (2017). Understanding application-battery interactions on smartphones: A large-scale empirical study. *IEEE Access*, 5:13387–13400. DOI: 10.1109/ACCESS.2017.2728620.
- Harihar, V. K. and Sukumaran, S. (2018). Behaviour comprehension and prediction using time series analysis of data for code offloading in mobile cloud computing. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 367–375. DOI: 10.1109/RTE-ICT42901.2018.9012270.
- Kitchenham, B. A. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report. Available at: [https://www.elsevier.com/\\_data/promis\\_misc/525444systematicreviewsguide.pdf](https://www.elsevier.com/_data/promis_misc/525444systematicreviewsguide.pdf).
- Linares-Vásquez, M., Bavota, G., Bernal-Cárdenas, C., Penta, M. D., Oliveto, R., and Poshyvanyk, D. (2018). Multi-objective optimization of energy consumption of guis in android apps. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(3):1–47. DOI: 10.1145/3241742.
- Lopes, C. V., Maj, P., Martins, P., Saini, V., Yang, D., Zitny, J., Sajjani, H., and Vitek, J. (2017). Déjàvu: A map of code duplicates on github. *Proc. ACM Program. Lang.*, 1(OOPSLA). DOI: 10.1145/3133908.
- Malavolta, I., Grua, E. M., Lam, C.-Y., de Vries, R., Tan, F., Zielinski, E., Peters, M., and Kaandorp, L.

- (2020). A framework for the automatic execution of measurement-based experiments on android devices. *2020 35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. DOI: 10.1145/3417113.3422184.
- Mehrotra, D., Srivastava, R., Nagpal, R., and Nagpal, D. (2021). Multiclass classification of mobile applications as per energy consumption. *Journal of King Saud University - Computer and Information Sciences*, 33(6):719–727. DOI: 10.1016/j.jksuci.2018.05.007.
- Oliner, A. J., Iyer, A. P., Stoica, I., Lagerspetz, E., and Tarkoma, S. (2013). Carat: Collaborative energy diagnosis for mobile devices. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems, SenSys '13*, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2517351.2517354.
- Oliveira, W., Oliveira, R., and Castor, F. (2017). A study on the energy consumption of android app development approaches. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 42–52. DOI: 10.1109/MSR.2017.66.
- Oliveira, W., Oliveira, R., Castor, F., Fernandes, B., and Pinto, G. (2019). Recommending energy-efficient java collections. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 160–170. DOI: 10.1109/MSR.2019.00033.
- Pang, C., Hindle, A., Adams, B., and Hassan, A. E. (2016). What do programmers know about software energy consumption? *IEEE Software*, 33(03):83–89. DOI: 10.1109/MS.2015.83.
- Pereira, R., Matalonga, H., Couto, M., Castor, F., Cabral, B., Carvalho, P., Sousa, S., and Fernandes, J. (2021). Greenhub: a large-scale collaborative dataset to battery consumption analysis of android devices. *Empirical Software Engineering*, 26. DOI: 10.1007/s10664-020-09925-5.
- Perez, J Diaz, B. T. (2020). Systematic literature reviews in software engineering—enhancement of the study selection process using cohen’s kappa statistic. *Journal of Systems and Software*, 168:110657. DOI: 10.1016/j.jss.2020.110657.
- Qualcomm (2017). Trepn description. Available at: <https://www.qualcomm.com/news/onq/2015/04/introducing-trepn-profiler-60> Accessed: 2021-11-24.
- Rua, R., Couto, M., and Saraiva, J. (2019). Greensource: A large-scale collection of android code, tests and energy metrics. *IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 176–180. DOI: 10.1109/MSR.2019.00035.
- Statcounter (2021). Operating system market share worldwide. Available at: <https://gs.statcounter.com/os-market-share#monthly-200901-202111> Accessed: 25/10/2021.
- UFSCar (2021). Start-lapes-ufscar. Available at: <https://www.lapes.ufscar.br/resources/tools-1/start-1> Accessed: 25/10/2021.
- Wang, C., Guo, Y., Shen, P., and Chen, X. (2017). E-spector: Online energy inspection for android applications. *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. DOI: 10.1109/ISLPED.2017.8009207.
- Xu, F., Liu, Y., Li, Q., and Zhang, Y. (2013). V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 43–55, Lombard, IL. USENIX Association. Available at: [https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/xu\\_fengyuan](https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/xu_fengyuan).
- Zhang, X., Xiao, X., He, L., Ma, Y., Huang, Y., Liu, X., Xu, W., and Liu, C. (2019). Pifa: An intelligent phase identification and frequency adjustment framework for time-sensitive mobile computing. *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 54–64. <https://par.nsf.gov/servlets/purl/10127975>. DOI: 10.1109/RTAS.2019.00013.