A Study on the Energy Consumption and Performance of Single-Activity Android Apps

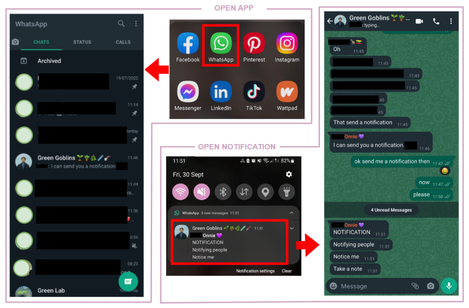
Carolina Neves, ChiaYu Lin, Srishti Nigam, Daumantas Patapas, Ander Eguiluz, Tanjina Islam, Ivano Malavolta
Vrije Universiteit Amsterdam, The Netherlands
{c.oliveiraneves | c.y2.lin | s.nigam | d.patapas | a.eguiluz}@student.vu.nl, {t.islam | i.malavolta}@vu.nl

***Abstract*—*Context*. A new architecture for Android apps has been recently introduced. It is called single-activity app architecture and it results in Android apps with a completely different structure and runtime behaviour when transitions among screens.**
***Goal*. The goal of this research is to assess the impact of the new single-activity architecture on the energy consumption, CPU usage, and memory usage of Android apps.**
***Method*. We developed four Android apps: two apps following the single-activity architecture (with basic and advanced complexity) and two apps following the traditional architecture (with basic and advanced complexity). Then, we measure all of them in terms of energy consumption, CPU usage, and memory usage and statistically analyse the collected measures.**
***Results*. Our results show a significant difference between single-activity and multiple-activity architecture for the dataset of Energy consumption (advanced complexity app) and Memory usage (basic complexity app) measurements.**
***Conclusions*. This study provides evidence that there is a significant difference in energy consumption between the single-activity and multiple-activity architecture on both basic and advanced Android apps.**

I. Introduction

An application's life cycle begins with a requirement list. It starts with having predefined functionalities that the application will have provided by either research, by the request of a customer, or by other means. But with the fast-paced world that we are in, each application needs to keep up and adapt or it will quickly become irrelevant and be exchanged for a different one with new technology and more functionality. With the introduction of the new functionality comes the task of managing the application's speed, network, stability, reliability, etc. The mobile app experience differs from the desktop counterpart, in that, a user's interaction with the app does not always begin in the same place. Instead, the journey begins non-deterministically. For instance, if you open a social media app such as Whatsapp, as shown in Fig. 1, you are shown the main screen with the recent activities contrast (on the left "Open App"), if you receive a push notification onto your phone about an event and you click on it you are sent directly onto that event's page (on the right "Open Notification").

That results in even more possible problems that a developer must oversee as they are developing the mobile app. Consumer opening their application expects that it will load fast, will not drain the battery quickly, and will not make their phone overheat. To keep up with users' expectations, new technologies get introduced with ideas as to how to better manage these issues and provide your application with the reliability the consumer expects. That brings about the introduction of app architectures such as multiple activities and single activities. But to understand the similarities and differences between both of these infrastructures it is important to first understand the technologies that surround them. Android activity is an application component that gives a user interface where the interaction occurs. When one app invokes another, the calling app invokes an activity in the other app. The activity serves as the entry point for a mobile app's interaction with the user. The window typically fills the screen, but could also be smaller than the screen and float on top of other windows.



Green Goblins
[redacted]: I can send you a notification [redacted]

11:51
Fri, 30 Sept
WhatsApp 3 new messages 11:51
Green Goblins 11:51
[redacted] Onnie
NOTIFICATION
Notifying people
Notice me

Green Goblins
[redacted] typing...
[redacted]
Oh 11:45
[redacted] 11:45
[redacted] 11:45
[redacted] 11:45
[redacted] 11:45
[redacted] 11:45
That send a notification 11:45
[redacted] Onnie
I can send you a notification [redacted] 11:46
ok send me a notification then 11:47
now 11:47
please 11:50
4 Unread Messages
[redacted] Onnie
NOTIFICATION 11:51
Notifying people 11:51
Notice me 11:51
Take a note 11:51

Fig. 1: Launching VS Opening a notification on WhatsApp

Most mobile apps contain multiple screens, which means they are comprised of multiple activities. Typically one activity in an app is specified as the main activity which first appears when the user launches the mobile app. Each activity after that may start another activity to perform different actions. As an example, the main activity is a social media application that provides a screen that shows recent activities. From there, the main activity might launch other activities that provide screens for tasks such as writing a comment. Fig. 2 shows the process of launching an activity from the main activity. On the left is the main activity - Facebook feed - and by pressing the "What's on your mind?" it launches the second activity - Create a Facebook post.

Although activities work together to form a cohesive user experience in a mobile app, each activity is only loosely bound to other activities and may even be start-up activities belonging to other apps.

Fragments are the reusable components that are attached and displayed within the activities. A fragment is a piece of an activity that enables a more modular activity design. It defines and manages its own layout, has its own lifecycle,

979-8-3503-1238-6/23/$31.00 ©2023 IEEE
DOI 10.1109/GREENS59328.2023.00008

and can handle its own input events. Fragments cannot live on their own and must be hosted by activity or another fragment [1]. These introduce modularity and reusability into the activities' User Interface (UI) as that allows to divide it into discrete chunks. Considering an app that responds to various screen sizes. On larger screens, the app should display a static navigation drawer and a list in a grid layout. On a smaller screen, the app should display a bottom navigation bar and a list in a linear layout. Managing all these variations in the activity can become close to impossible. Separating the navigation elements from the content can make this process more manageable. The activity is then responsible for displaying the correct navigation UI, while the fragment displays the list with the proper layout.

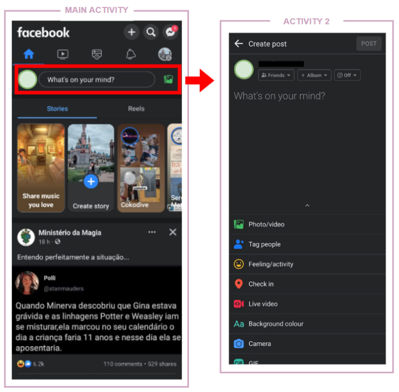


Fig. 2: Main activity with a side activity

Dividing the UI into fragments makes it easier to modify the activity's appearance at run-time. When an activity is running, fragments can be added, replaced, or removed.

Fig. 3 shows the difference between the architectures in question. Multiple-activity architecture has multiple activities where the data is shared using a singleton data holder and each activity has its own fragments. Whereas in a single activity architecture, one activity generates fragments that are shared using a shared view model.

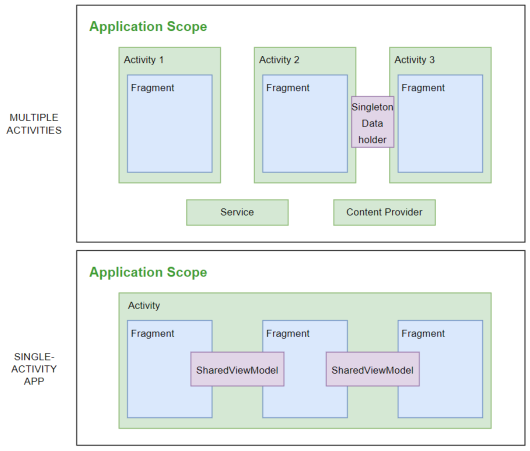


Fig. 3: Two architectures

The purpose of this research is to analyze the benefits and drawbacks of having either a multiple-activity mobile application or a single activity mobile application and evaluate which architecture design is more beneficial in terms of Central Processing Unit (CPU) usage, memory allocation, and energy consumption. We will conduct a statistical analysis on the collected data for the energy, CPU, and memory usage, from the applications designed by us, based on the two architecture types, single and multiple architecture, and the two complexity types basic and advanced, explained in more details in the upcoming sections. The research aims to benefit the developers when choosing the app's architecture.

II. Related Work

The impact of energy consumption on single-activity multiple fragments and multiple-activity Android apps has been given little attention. Therefore in this section, we summarize the literature relevant to Android fragments and activities, mentioning how they may relate to our experiment on energy consumption, CPU usage, and memory allocation.

Yongfeng et al. [2] compared the life cycle of Android activities and fragments. They found that the fragment's life cycle has an effect on data flow analysis on Android apps. They also pointed out that the fragments could cause memory leakage if their life cycle is not considered properly. Based on their experiment in our study, we aim to investigate the memory consumption of single activity multiple fragments architecture over multiple activity architecture on Android apps.

Jun et al. [3] developed an automatic tool LeakDAF for detecting memory leakage caused by fragments on Android apps. In contrast to their experiment, we aim to analyze whether memory leakage from fragments has an effect when a single activity is used compared to multiple activity architecture.

Jue Wang et al. [4] focused on optimizing and updating the existing GreenDroid tool that diagnoses energy inefficiency in Android apps by supporting the features of fragments. In contrast to our experiment, the authors focused on the energy consumption issues of Android applications with and without fragments.

Xianfeng Li et al. [5] analyzed the reduction of redundant rendering in Android applications. The authors shed a dim light on the possibilities of increasing energy efficiency in GUI rendering by changing between single and multiple-activity architectures in specific Android applications. However, they do not provide a general overview of the neat energy-efficiency difference in using one architecture or the other in rendering.

III. Experiment Definition

The goal of this study is to *analyze mobile architecture designs for the purpose of evaluation with respect to their energy consumption and performance from the point of view of a developer in the context of Android mobile applications.*

Our goal is refined into the following RQ:

[RQ1] *How does single activity compare to multiple-activity in terms of energy consumption and performance?* To answer this question we will consider two apps with two levels of complexity, basic and advanced. The different levels of complexity will be analyzed for both architecture types. This means that for each app type, there will be two versions, one using single activity architecture and another, using multiple activity architecture. Then we will analyze the performance based on the CPU usage (%), memory usage (Bytes), and energy consumption (J) of each of these applications.

We identify the following two sub-questions related to RQ1:

[RQ1.1] *How does single activity compare to multiple-activity in terms of energy consumption?* To answer this question, we consider the energy consumption in Joules of the applications.

[RQ1.2] *To what extent does single activity compare to multiple-activity in terms of performance?* To answer this question, we consider the average CPU usage (%) and memory usage in Bytes (B) of the applications.

IV. Experiment Planning

The goal of this research is to compare and contrast the energy consumption and performance of multiple activity apps and single activity apps. This section includes the plan for achieving our research goal, the subjects, and variables involved in the experiment, and the hypotheses defined. Finally, the data analysis method will be described in later sections.

A. Subjects Selection

The subjects of this research are the mobile applications used for measuring energy consumption. Before starting the subject selection process, the research team first needed to determine which Android mobile device to use for the subjects to execute. According to Open Signal, the founder of the mobile signal monitoring app, there are over 24,000 unique Android devices as of June 2015 [6]. Since it is impossible for us to obtain and test such a large number of devices, our research team decided to concentrate on the devices we already have. The experiment device that the research team used is Google Pixel 5 with Android version 11, which is a relatively new mobile phone released in September 2020.

Another critical decision is the selection of the subjects, which are the Android applications that would be analyzed during the experiment. The research team first delved into the pre-existing apps that used single-activity and multiple-activity architectures. However, without looking at the original source code, it is difficult to tell which architecture this app is based on. Second, it was impossible to find two identical apps developed using these two different architecture types. Therefore, the Android applications were developed from scratch to ensure the validity of the experiment. In total, the research team developed four apps, two basic ones, and two advanced ones. Both the basic and advanced apps are implemented twice, one for each type of architecture, single and multiple-activity. The main difference between the advanced and basic apps is the complexity of the app elements used and the amount of data exchanged inside the app. The basic applications consist of 4 pages the onboarding, sign-up, sign-in, and the homepage, whereas the advanced applications have an additional 2 pages of video selection page and the video page. A more detailed explanation of the design and the pages for basic and advanced apps can be found in IV-D.

B. Experimental Variables

To conduct the experiment of this project, the independent variable is the architecture of the app, it will be one of the two treatments: *single-activity* or *multiple-activity* architecture.

The dependent variables, in turn, are the following:

- Energy Consumption (J): The overall energy consumption by the single/multiple-activity app architecture running in the smartphone.
- CPU usage (%): The average CPU usage of the application, measured in real-time while interacting with the app.
- Memory usage (Bytes): The overall memory usage while the app is running.

These dependent variables correspond to the research questions specified in the GQM. This can be viewed in the table below:

C. Experimental Hypotheses

The goal of this research is to investigate the impact both single and multiple activity architecture have on the dependent variables described in Section IV-B. As the complexity of the application can affect the dependent variables, we analyze the impact of the basic and advanced apps separately.

From the experiment, we calculate population means μ_{ijk} with $i \in \{e, c, l\}$ each of the dependent variables energy consumption (e), CPU usage (c), memory usage (l) respectively, $j \in \{b, a\}$ each level of complexity basic (b) and advanced (a) respectively, $k \in \{s, m\}$ indicating single activity (s) and multiple-activity (m) architecture respectively.

For the energy consumption variable e, it is unclear whether the single activity architecture will perform better than the multiple activity architecture. To this end, we execute two-sided statistical tests on the hypothesis.

$$\begin{aligned} H_{0,ej} &: \mu_{ejs} = \mu_{ejm}, \forall j \in \{b, a\} \\ H_{\alpha,ej} &: \mu_{ejs} \neq \mu_{ejm}, \forall j \in \{b, a\} \end{aligned} \tag{1}$$

For the performance variables c and l, we cannot assume which architecture single or multiple-activity will perform better. To this end, we execute two-sided statistical tests on the hypotheses.

$$\begin{aligned} H_{0,ij} &: \mu_{ijs} = \mu_{ijm}, \forall i \in \{c, l\}, \forall j \in \{b, a\} \\ H_{\alpha,ij} &: \mu_{ijs} \neq \mu_{ijm}, \forall i \in \{c, l\}, \forall j \in \{b, a\} \end{aligned} \tag{2}$$

D. Experiment Design

The main point of this experiment is the design of the subjects, which are the four mobile applications used to measure energy consumption (two advanced apps and two basic apps). To compare, one advanced app and one basic app were built using the multiple activity architecture, while the other advanced app and basic app were built using the single activity architecture. The primary difference between advanced and basic apps is (1) **the number of pages designed**, which affects the amount of data passing between the pages. The second difference between basic and advanced apps is (2) **the complexity of app elements used**. The two main design differences would be described in this section:

Pages: Basic apps, Fig. 4a, contain only four pages: the on-Boarding page, with a sign-up button that leads the user to the Sign-Up page, the Sign-up page, where users provide their basic information, the Sign-in page, where users input their username and password, and lastly the Home Page, which displays the users profile information. User information like username, password, age, and address will be exchanged between these four pages.

For advanced apps, Fig. 4b, six pages were designed. The pages include an on-Boarding Page, a Sign-in Page, and a Sign-up Page which contains more data columns such as gender, email, and phone number for the user to fill in, along with a location API, a Home Page which listed more user information, a Video Selection Page where user can select one video to view and lastly the Video Page where the videos are played. Compared to the basic apps, more user information will be exchanged between these six pages.

Elements: In basic apps, Fig. 4a, the pages consist of data columns [7] for users to type in their information. In advanced apps, Fig. 4b, except for the data columns [7], Android Location [8] is used to track the user's current location. Additionally, a ListView [9] and a Video Player [10] are utilized to display the collection of videos and play the videos, respectively. The elements in the advanced apps are more complex and more energy consumption comparing to the basic ones.

The summary of the basic and advanced app is shown in Table I. Also, a prototype for the apps is drawn in Fig. 4a and Fig. 4b.

TABLE I: Summary of basic and advanced apps

	Pages	**Elements**
Basic	Home Page Sign-in Page Sign-up Page On-Boarding Page	Data Columns
Advanced	Home Page Sign-in Page Sign-up Page on-Boarding Page Video Selection Page Video Page	Data Columns List Android Location API Video Player

During the experiment, the research team navigated through the app pages in a fixed order to measure energy consumption. The navigation order is also drawn in Fig. 4a and Fig. 4b. We count a tour of page navigation as a run, since each run only takes a little amount of time, it would be difficult to measure the energy consumption. Therefore, the research team performed 10 turns in a single trial. The energy consumption may vary over the time the app has been running, for example, the battery consumption may drop faster from 75% to 74% than 100% to 99%. In order to increase the accuracy of the experiment, initially we were planning to perform each trial 10 times with a 5 min resting period in between them to ensure the phone charges back to 100% and has enough time to cool down. However, in order to have a bigger dataset with more data points, we decided to increase the size of the trials from 10 to 40 runs, thus reducing the waiting time from 5 to 1 min, due to time constraints for running the complex apps multiple times.

V. Experiment Execution

The execution of our experiment is divided into several steps. First, we prepared the subjects and the setup of the hardware and tools required for the measurement. Then, we collect the data, and finally we test the hypotheses and perform the data analysis. A complete **replication package** is available for verifying the results of this study[1].

A. Preparation

Android applications for the research project were created using Kotlin programming language version 1.7.0 and an Android Studio integrated development environment. Android Runner, in turn, is being used for collecting data related to the dependent variables. This framework was selected because it allows the developers to get different metrics and set up a tailored experiment in a compact and centralized way.

With the Android application created, in order to be able to check how it looks and behaves on a device, the application needs to be built and run as a release app. To use the Android Runner [11], first we set up the environment and the dependencies, and then install the framework. Finally, the interaction with the app will be automatized using a Python script. MonkeyRunner [12] will be used to detect and reproduce all the events related to button selection, navigation, and text input. Therefore the related library for MonkeyRunner must be downloaded and properly configured since it has been deprecated for the latest version of Ubuntu.

B. Setup

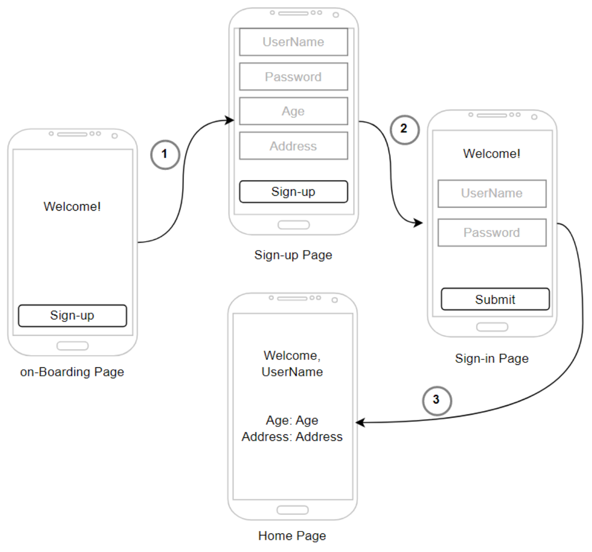
The measurement process is managed by the Android Runner framework [11]. To utilize Android Runner, we would need to first make the device accessible through the ADB [13] utility, which makes possible the connection between the device and the laptop. Hence, we connect our Android device to our personal computer using a USB C cable. The device will stay connected and unmoved, and the debugging tools option will be enabled while the device is connected to the system. This will allow the ADB utility to identify and operate with the Android device.

Next, we can record the required interaction to perform the tests within the four apps with MonkeyRunner [12]. This will generate a textual file containing the performed actions that we will later transfer to the `interaction.py` file, which will take care of performing the same actions for every run[2]. After that, we prepare a configuration file describing the experiment, which then will be used by Android Runner for executing the actual experiment.

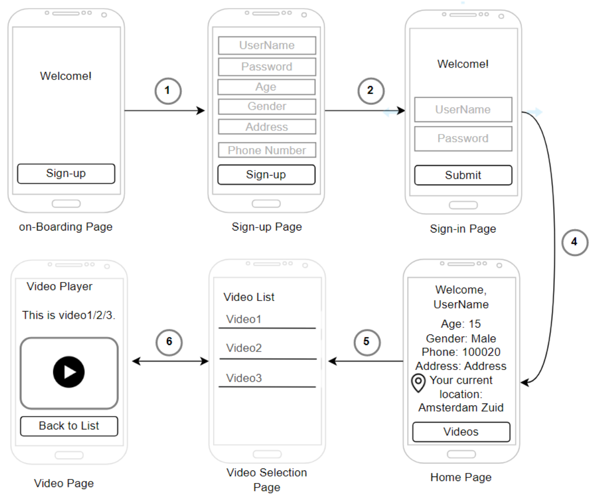
- Add the pertinent plugins (Android Profiler [14] and BatteryStats [15]).
- Specify the module name of the corresponding app.

[1]https://github.com/S2-group/greens-2023-single-activity-android-rep-pkg

[2]The duration of the usage scenarios is about 20 seconds for the basic app and about 33 seconds for the advanced one.



(a) Basic App Prototype



(b) Advanced App Prototype

Fig. 4: Basic App Prototype (a) vs Advanced App Prototype (b)

- The number of runs required.
- Time between runs for cooldown.
- Scripts to be run during the experiment, including the interaction.py file, and other configuration details.

For our experiment, we prepared four different config.json files with the corresponding interaction.py file for each app.

The BatteryStats [15] tool is used to calculate the power consumption of the different components of the device. This will be used to compare the consumption of energy between single and multiple activity architectures and identify the impact on the device's energy consumption that corresponds to battery drain. Memory profiling is done to evaluate if there is an increase in memory allocation caused by memory leaks caused by either of the architectures. While the CPU usage is recorded to analyze how much computational power the different apps take in comparison with each other.

By conducting research, it was found that the Android development process may involve a Raspberry Pi, but the requirements of the testing process are entirely different. This is because the Pi isn't powerful enough to run an Android emulator, hence the whole application is built and installed on an actual device Google Pixel 5 with Android version 11, equipped with 8GB RAM and a 4080mAh battery for testing.

C. Measurements

At the start of each trial the phone gets connected to the ASUS GL552V laptop (Ubuntu Linux), equipped with an Intel i5 6300HQ CPU and 8GB of RAM, and battery protection is enabled to avoid it from charging. In order to compare the consumption of the apps from the same baseline, screen brightness is set to a minimum, Wi-Fi and location are turned on and the phone is on the "Home" screen. The connection is tested by running the ADB devices command, which yields the status of the desired device. If the status is correct, we launch the trial.

The trial starts with a command pointing to the pertinent config.json file, which runs the built-release app on the Android device and starts performing the pre-recorded actions on the interaction.py file. The data is collected in two different files, one for CPU and memory, the other for energy consumption. After the navigation through the app is done, the phone is set back to charging mode and a period of 1 minute is set to let the phone cool down. The data obtained is then used for the analysis process.

D. Analysis

All statistical tests and graphical representation will be performed using R Studio, with R version 4.2.2.

Based on the design of the experiment, we perform tests for one factor and two treatments. The data collected for energy, CPU, and memory needs to be analyzed. Primarily, descriptive statistics for the data are obtained by listing the Mean, Median, Minimum, Maximum, 1st, and 3rd Quartile values. The data is also plotted using Box-Plots, to visualize any relevant outliers. To remove the outliers, iris data [16] is used, and to understand the distribution of the variables Shapiro Wilks test is used [17]. Once the analysis has been carried out, we will perform transformation techniques to convert non-normal data to a normal distribution, depending on the skewness values, using the R library called moments [18]. These are further visualized using Q-Q plots. Finally based on the result we will either conduct the two-sided T-Test if the distribution is normal or Mann Whitney U Test (Wilcoxon Rank Sum Test) if it is not.

VI. Results

The results section is split into four parts. The first shows the descriptive statistics of the collected data, the second shows how the data was processed for the outliers, the third shows the tests for normality alone and the fourth displays tests for the hypotheses.

A. Descriptive Statistics

The first step in order to have a better understanding of the collected data is to have an overall look at the descriptive statistics. In Table II, we have a summary of statistics (Min, Max, Mean, Median, etc) of both complexity levels, basic and advanced, for each of the treatments, single and multiple, and dependent variables, energy consumption, CPU usage, and memory usage.

Energy Consumption. By analyzing the value of the median of the energy consumption (e) in Table II, it is possible to determine that for the complexity level basic there is an increase, although it is not significant. While for the complexity level advanced there is a slight increase in the median of multiple-activity compared to a single activity architecture. The boxplots in Fig. 5 give a better overview of the descriptive data in addition to the data present in Table II. Hence, we can support our claim that the energy is fairly similar in the basic app, but there is an increase in energy for multiple activities in the advanced app. We can also observe that there exists a few outliers in both basic and advanced complexity apps, and these need to be removed.

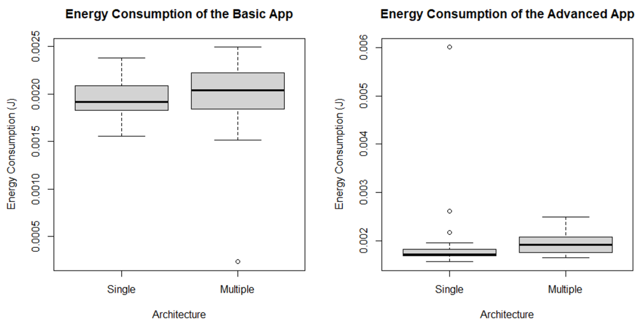


Fig. 5: Box-plot of the energy consumption per treatment for both complexity levels of the App.

Memory Usage. The value of the median of the memory usage (m) in Table II, is larger when the basic app uses single activity versus using multiple activities. However, for the advanced app, there is no significant difference between the median of the two architecture types.

The boxplot in Fig. 6 supports the inferences made based on the descriptive statistics, as the spectrum of data for the basic app using multiple activities is much smaller than for the app using single activity. For the advanced app, the values are fairly similar. Similar to the other variables, we can also observe a few outliers in the basic and advanced apps.

CPU Usage. The value of the median of the CPU usage (c) in Table II, are fairly similar between the architecture types, for both complexity levels, with the single being slightly higher in the basic app. It is possible to see a big gap between the maximum value and the mean for all four scenarios, which could entail potential outliers.

The boxplot in Fig. 7, supports the initial inferences about the data. The single architecture type is slightly higher than multiple architectures in the basic app, also showcasing a large number of outliers. While the advanced app shows fairly similar values for both architecture types.

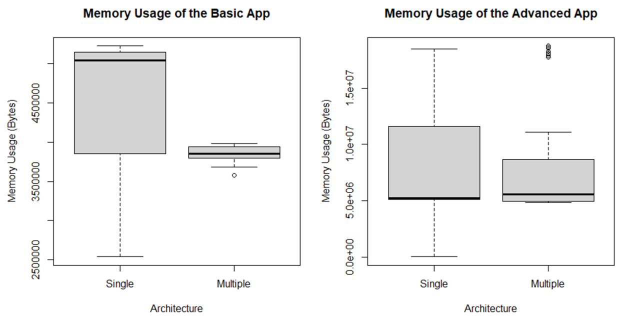


Fig. 6: Box-plot of the memory usage per treatment for both complexity levels of the App.

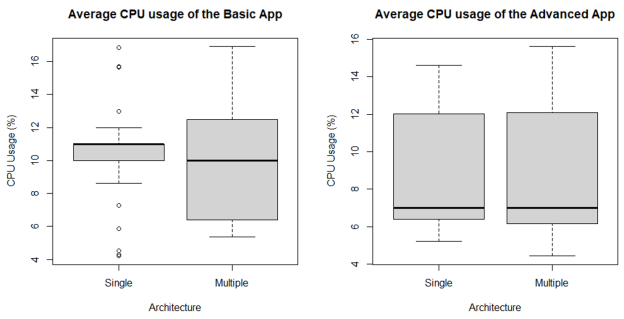


Fig. 7: Average CPU usage for both complexity levels.

B. Removing Outliers

As observed in the section above, it can be noted that multiple datasets have outliers that need to be treated. This can be caused due to various external factors while collecting the data. The treatment of the data can be done in multiple ways. One of them includes selecting the quantitative variables from iris data. By calculating the first and third quantiles (Q values), the interquartile range (IQR values), and by finding the lower and upper limits for outliers. After this, we remove the outliers by taking a subset of the original data that lay within this range that is calculated using Q and IQR values. We implement in-built R functions such as $\text{quantile}()$, $\text{IQR}()$, and $\text{subset}()$ and create data with minimal outliers for each dependent variable (Energy, Memory, CPU Usage) for both basic and advanced type of applications for the two treatments single and multiple activity architectures [16].

C. Check for Normality and Transformation

Testing for normality is an essential step before testing the hypotheses. We checked if the energy consumption, memory usage, and CPU usage values are normally distributed via (i) a visual analysis of Q-Q Plots and (ii) the application of the Shapiro-Wilks statistical test with $\alpha = 0.05$ [17]. We also applied several data transformation techniques to explore the possibility of having a normal distribution, which can potentially lead to higher statistical power. In conclusion, the energy consumption values we measured are normally

TABLE II: Descriptive Statistics for all treatments and dependent variables: energy (e), memory (m), and CPU (c).

Complexity	**Basic**						**Advanced**					
Architecture	**Single**			**Multiple**			**Single**			**Multiple**		
Variable	**e (mJ)**	**m (MB)**	**c (%)**	**e (mJ)**	**m (MB)**	**c (%)**	**e (mJ)**	**m (MB)**	**c (%)**	**e (mJ)**	**m (MB)**	**c (%)**
Minimum	1.553	2.540	4.200	0.234	3.577	5.370	1.565	0.038	5.227	1.655	4.827	4.435
1st Quantile	1.830	3.862	10.000	1.844	3.797	6.410	1.688	5.145	6.408	1.776	4.962	6.201
Median	1.914	5.041	11.000	2.039	3.849	9.990	1.721	5.238	7.015	1.916	5.576	7.014
Mean	1.943	4.644	10.470	1.970	3.852	10.060	2.005	8.319	8.723	1.943	8.332	8.792
3rd Quantile	2.085	5.140	11.000	2.212	3.939	12.280	1.817	8.455	12.014	2.073	7.453	12.074
Maximum	2.378	5.223	16.840	2.491	3.984	16.900	6.007	18.465	14.594	2.488	18.722	15.603

distributed and will follow a parametric test. CPU Usage and Memory could not be transformed to a normal distribution and will follow a non-parametric test. The Q-Q plots, results of the Shapiro-Wilks test, and the details about normality checks and transformation are available in the replication package.

D. Hypotheses Testing

The **Energy** data follows a normal distribution, and hence we consider a parametric test called a two-sided T-Test. For **Memory** and **CPU Usage**, we conduct a non-parametric test called Mann Whitney Wilcoxon Rank Test, as it also considers two-tailed values and is for data that is not normally distributed. All of these tests were performed in R using the various in-built functions `t.test()` and `wilcox.test()`. The values of the p-value for each experiment, obtained after running the tests on single and multiple activity datsets, for basic and advanced types can be seen in Table III.

TABLE III: P-values for all dependent variables.

	Hypothesis Test	**Basic**	**Advance**
Energy	T-Test	0.1604	**0.00041**
Memory	Mann Whitney Wilcoxon Test	**2.6e-05**	0.6809
CPU	Mann Whitney Wilcoxon Test	0.436	0.5928

The significance level for both types of tests was fixed at 0.05. If the p-value is lower than 0.05, we reject the null hypothesis. As is in the case of Energy for the advanced complexity app and Memory for the basic complexity app. In both these cases, we can conclude that the single and multiple activity apps have a significant difference. The rest of the results show us a value greater than 0.05 and we fail to reject the null hypothesis, thus stating that both single and multiple-activity do not have a significant difference.

Conclusion Single- and multiple-activities architectures tend to perform similarly in terms o energy consumption, CPU usage, and memory usage, with two (statistically significant) exceptions: (i) multiple-activities architecture consumes more energy than single-activity architecture in the advanced apps and (ii) multiple-activities architecture uses less memory than single-activity architecture in the basic apps.

VII. Discussion

To answer the main research question RQ1, the sub-questions will be answered separately.

[RQ1.1]: As noted in Table III, the p-value for Energy is lower than the significance level of 0.05 and we reject the null hypothesis in the advanced complexity app. This means there is a difference between single and multiple-activity architecture types. We do not see a significant difference in the basic complexity app, as the value is greater than the significance threshold. This could be due to the fact that our basic app is too simple to be able to exhibit noticeable differences in terms of energy. Whereas, for the advanced app, we can observe a difference because of the various added features to the app that consume energy, such as the video player, location API, etc.. We conjecture that this result is due to the fact that in multiple-activity apps the OS needs to recreate components when transitioning among the various activities, while in single activity apps the OS just needs to such components to the already-existing activity layout.

[RQ1.2]: For Memory, the basic app shows a significant difference between single and multiple-activity architecture in Table III whereas there is no difference in the advanced complexity app. This difference in the basic app could be because of an increase in memory allocation and the saving and restoring of data between fragments. For multiple-activity apps, the size of the data to be sent back and forth is very small, but over time as the app continues to be utilized, the data size increases. In single activity, this is much simpler as the data has to be read and written within the same activity.

For CPU there exists no significant difference in any of the complexities, basic or advanced. We believe the CPU data could be somehow distorted due to external factors as we can see that there are several outliers, and the data does not follow a specific pattern. There is a certain density of data points clustered together, which the tests consider as plot points rather than dealing with them as outliers. A more comprehensive outlier detection method can be used to treat the data, for mitigating this potential source of bias.

The results of this study provide developers and researchers an indication that using a single-activity architecture might lead to different levels of energy and memory usage in Android apps. Researchers may consider this result to lay down the foundation for their future research, as we found only a few studies related to the runtime impact of single-activity and multiple-activity app architectures. As for the developers, they may consider this result as initial preliminary evidence that refactoring their Android app into a single-activity app can lead to non-negligible changes on its energy consumption and performance.

VIII. Threats To Validity

Internal Validity. Regarding the mono-method bias, we used only one tool to measure each dependent variable (i.e., CPU, memory, and energy) which may result in a bias from the profilers. To mitigate this threat, we performed several tests to contrast the validity of the data and we assessed the reliability of the measures. Moreover, we executed all the runs with the same conditions for the two apps: minimum brightness, Wi-Fi connection and locations enabled, and disabled push notifications.

External Validity. The most relevant threat to the external validity of our experiment is the use of apps that were developed by us, due to the difficulty of finding two identical apps developed with these two different architectures (single and multiple activities). This may lead to the possibility of the results not being as representative as they would be, with the pre-existing apps with a higher level of complexity than what we consider our advanced app to be. However, to mitigate this effect we designed the advanced app so that it uses different features of the Android platform representing recurrent functionalities of Android apps, such as requests to the Cloud, accessing user's location, showing video contents and images, etc. Lastly, the interaction of setting and treatment and the interaction of history and treatment should not be a threat to validity for our experiment, since the trials were completed in the same computer over the span of several days and there was no difference between either of the days.

Construct Validity. For this experiment, we defined the constructs early in the design process by using the GQM (Goal, Question, Metric) method. This experiment was carried out taking solely the architecture of the app as the independent variable. This means that our experiment might undergo a mono-operation bias. However, to mitigate this effect and the possible bias from our measurements, we performed multiple repeated runs on the two subjects for the two different treatments, each.

Conclusion Validity. It was crucial that the statistical tests were appropriate to the type of distribution. For this, all the data were tested for normality through the Shapiro-Wilk Test and via Q-Q Plots. For the normally distributed data, the hypotheses were analysed using t-tests. For the data that did not present normal distribution, even after implementing several techniques to transform the data to normal considering the skewness levels of the data, as mentioned in the previous section, the hypotheses were analysed using a non-parametric test called Mann Whitney Wilcoxon Test.

IX. Conclusions

In conclusion, this paper researched the energy consumption of a single-activity and multiple-activity Android application. The testing was conducted on 4 android applications. Two of them were created for a single activity and two for a multiple-activity architecture with the principle of basic and advanced complexity. Memory usage, energy consumption, and CPU usage measurements were gathered from the test runs and analyzed. It was found that for a basic app, Memory usage shows a significant difference between single and multiple-activity app architecture which could be caused due to memory leaks and data sharing. Moreover, it was noted that for the advanced app, there is a significant difference between single and multiple-activity for the Energy consumption measurements, most likely due to the recreation of components that consume more energy. Possible future work could include replicating the experiment while making the advanced applications more complex as it would help to find a clearer difference between the two architecture types. Another approach would be testing the applications across several Android devices, as our experiment was only done on Google Pixel 5 with Android version 11. It would be more insightful to see what would be the outcome when using different Android versions and different devices.

References

[1] A. Developers. (2022) Fragments. [Online]. Available: https://developer.android.com/guide/fragments

[2] Y. Li, J. Ouyang, B. Mao, K. Ma, and S. Guo, "Data flow analysis on android platform with fragment lifecycle modeling and callbacks," *EAI Endorsed Transactions on Security and Safety*, vol. 4, no. 11, 12 2017.

[3] M. Jun, L. Sheng, Y. Shengtao, T. Xianping, and L. Jian, "Leakdaf: An automated tool for detecting leaked activities and fragments of android applications," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2017, pp. 23–32.

[4] J. Wang, Y. Liu, C. Xu, X. Ma, and J. Lu, "E-greendroid: Effective energy inefficiency analysis for android applications," 09 2016.

[5] X. Li, G. Li, and X. Cui, "Retriple: Reduction of redundant rendering on android devices for performance and energy optimizations," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[6] O. Signal. (2015) Android fragments. [Online]. Available: https://www.opensignal.com/reports/2015/08/android-fragmentation

[7] "Android Developers - EditText." [Online]. Available: https://developer.android.com/reference/android/widget/EditText

[8] "Android Developers - Location." [Online]. Available: https://developer.android.com/training/location

[9] "Android Developers - ListView." [Online]. Available: https://developer.android.com/reference/android/widget/ListView

[10] "Android Developers - Video Player." [Online]. Available: https://developer.android.com/guide/topics/media-apps/video-app/building-a-video-player-activity

[11] I. Malavolta, E. M. Grua, C.-Y. Lam, R. de Vries, F. Tan, E. Zielinski, M. Peters, and L. Kaandorp, "A Framework for the Automatic Execution of Measurement-based Experiments on Android Devices," in *35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '20)*. ACM, 2020. [Online]. Available: https://github.com/S2-group/android-runner/blob/master/documentation/A_Mobile_2020.pdf

[12] "Monkeyrunner utility." [Online]. Available: https://developer.android.com/studio/test/monkeyrunner

[13] "Android Debug Bridge." [Online]. Available: https://developer.android.com/studio/command-line/adb

[14] "The android profiler." [Online]. Available: https://developer.android.com/studio/profile/android-profiler

[15] "Batterystat utility." [Online]. Available: https://developer.android.com/topic/performance/power/setup-battery-historian

[16] "How to Remove Outliers from Data in R." [Online]. Available: https://universeofdatascience.com/how-to-remove-outliers-from-data-in-r/

[17] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965. [Online]. Available: http://www.jstor.org/stable/2333709

[18] Comprehensive R Archive Network (CRAN), "CRAN - Package moments." [Online]. Available: https://cran.r-project.org/web/packages/moments/index.html