Benchmarking the Energy Measurement Accuracy of the Android Runner Framework

L. Kaandorp

Vrije Universiteit (VU) Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam

Abstract. Android Runner is a Python framework, developed by the S2 Group at the Vrije Universiteit Amsterdam, that measures the energy consumption of native and web apps on Android. Android Runner is still in its infancy. Therefore, this paper analyses the level of accuracy that Android Runner provides when measuring the energy consumption of native Android apps. This analysis consists of evaluating the variance in the results of 15 different benchmark applications that each test a different functionality. This paper also categorises the components in Android phones in terms of their energy consumption by evaluating the mean energy consumption increase compared to the baseline reading. Lastly, this paper researches how the execution of experiments involving multiple variations of native Android apps can be facilitated by designing and developing a Python framework that does this. The results showed a low variance for 45 out of the 54 total tests, while the others could be attributed to two factors; the functionality itself inherits variance, or the variance is caused by the test device. The results yielded a clear distinction between components that consume much energy (e.g. CPU and networking) and those that consume little energy (e.g. sensors and speaker). A Python framework called Apk Generator was designed to facilitate the creation of multiple variations of the same Android app. In conclusion, it seems that when only the variance of the results is regarded, Android Runner offers an adequate level of accuracy. The full categorisation of the components can be found in the conclusion of this paper. Apk Generator can be used as a tool to create multiple variations of a native Android app.

Keywords: Android Runner · Android · Energy consumption · Accuracy · Benchmark · Components

1 Introduction

Over the last ten years, the Google Play Store has seen an exponential growth of the number of apps the platform has to offer. [1] Following a similar trend is the average storage capacity in Android smartphones. The average storage capacity was not even 15 GB at the end of 2014 but has grown to almost 70 GB by the end of 2019. [2] On the one hand, these two developments provide smartphone users with more choice and more capacity for installing apps. On the other hand, the battery capacity does not follow this trend and therefore stresses the importance of energy consumption optimisation by app developers.

Smartphones have become more and more complex since the early days. Since the days of the Simon Personal Communicator by IBM, smartphones have evolved to pocket computers that can do almost anything. They have sophisticated sensors like GPS, accelerometer, and gyroscope but also access to a range of connectivity like Bluetooth, WiFi, and 4G. Every component uses energy, and as said before, the advancements in battery capacity have lacked behind. This also stresses the importance of energy consumption optimisation by app developers since the use of these components does not come without a cost in terms of energy consumption.

The Software and Sustainability (S2) Group of the Vrije Universiteit Amsterdam has created Android Runner (`https://github.com/S2-group/android-runner`), a Python framework for running empirical experiments on run-time qualities of native and web apps in Android. [3] This tool could be useful to app developers who want to optimise the energy consumption of their app. However, it could also be useful to researchers working on energy measurement of Android apps. Android Runner is still in its infancy and has not been proven to be accurate.

This paper will analyse whether Android Runner can accurately measure the energy consumption of specific components in Android phones. This will ensure that Android Runner can be used as a reliable tool by researchers in the field of energy measurement. This paper will also categorise components of Android phones in terms of their energy consumption to provide app developers and researchers with a better understanding of the impact that these components have on energy consumption. Hopefully, this will build a solid foundation for further research into the impact that specific components have on energy consumption. Lastly, the paper will provide a framework that facilitates the execution of experiments involving multiple variations of native Android apps.

Section 2 discusses some relevant related literature. Section 3 clarifies the terminology used throughout the rest of the paper. Section 4 discusses the research questions and the setup of the experiment. Section 5 gives an overview of the results. Section 6 states the main findings from the results. Lastly, section 7 contains the conclusions, limitations, and discusses the importance of further research into this topic.

2 Related literature

In recent years, interest in energy consumption measurement and prediction in mobile phones has increased amongst researchers. Many have proposed different techniques, methods, and tools for this purpose. One of these tools is GreenMiner, a hardware testbed, which measures energy consumption directly from the phone using physical wires that are connected to an INA219 chip, proposed by Hindle et al. [4] GreenMiner, just like Android Runner, provides an interface that allows for automation of the testing runs. However, GreenMiner is hardware-based and thus comes with its costs while Android Runner is software-based and requires no additional hardware.

Another tool is an emulation tool, proposed by Mittal et al. [5] This tool allows developers to estimate the energy consumption of their app from their development workstation. This tool was designed for Windows phones, which have been discontinued by Microsoft as of January 2020. This discontinuation severely limits the practical use of the tool.

Hu et al. propose a lightweight and automatic approach to analyse and predict the energy consumption of Android apps. [6] In their approach, they decompile the APK of the app and insert code for measuring the bytecode. This process is quite labour intensive and complicated and would severely hinder app developers and researchers from using this method, even though its results are relatively accurate.

Le et al. propose a set of algorithms for the unified automation for optimising and merging both the power states as well as the power consumption automation of hardware components. [7] Their study also yielded a handy support tool which visualises the power machines and estimates the accumulated power consumption of the application. This, however, is an estimation and the goal of this paper is to determine whether Android Runner can accurately and precisely measure the energy consumption of an app.

A study by Tawalbeh et al. focusses on the different components in Android phones. [8] Their approach uses the PowerTutor and AMobiSense apps to measure energy consumption. Since these apps are also installed on the device and require energy in themselves, these apps influence the experiment. Furthermore, their approach does not automate the experiment. Therefore, this paper aims to use an external tool that automates the experiment and does not influence the phone.

Physalia is another toolset, proposed by Cruz and Abreu. [9] This toolset consists of a Python library that collects energy consumption measurements from Android devices. It also automates UI interactions. The only downside to Physalia is that it requires additional hardware. As of writing, it only works with Monsoon Power Tools.

A paper by Di Nucci et al. proposes a software tool named PETrA. [10] This tool contains an interface which allows users to, for example, determine the number of runs or the number of interactions. These features allow for the automation of the experiments. Also, PETrA has proven to be nearly as accurate as hardware-based solutions like GreenMiner. However, PETrA differs a lot from Android Runner. PETrA measures the energy consumption of the interactions, whereas Android Runner measures the power consumption of the app and the components of the phone.

3 Terminology

Android Runner is a Python framework for running empirical experiments on run-time qualities of native and web apps in Android. This tool is developed by the Software en Sustainability (S2) Group of the Vrije Universiteit Amsterdam. The tool facilitates automated testing of interactions and data aggregation. It is available at: `https://github.com/S2-group/android-runner`.

A **benchmark app** is a native Android application, built using the Kotlin language, which stresses a particular functionality in the smartphone. This allows the measurement of the energy consumption of this functionality. These applications are the subjects of our energy consumption measurements.

A **test device** is an Android smartphone that is used to conduct the experiment.

A **functionality** is a method or feature of the test device. Examples of functionalities are writing to internal storage or the camera. All these functionalities are native to the Android framework. Each benchmark app tests exactly one functionality.

Continuous benchmark apps stress a functionality continuously. An example of a continuous benchmark app is the one designed to stress the accelerometer; the Android framework allows for continuous monitoring of this sensor and its output. Benchmark apps are preferably made continuous since this is likely to yield the most accurate results.

Interval benchmark apps stress a functionality at the specified interval. An example of an interval benchmark app is the one designed to stress the camera, which stresses the camera by taking a picture every fifteen, ten, or five seconds.

An **intensity set** is a set of intensity levels that is used for testing interval benchmark apps. The most used intensity set uses an interval of 3 seconds for low intensity, 2 seconds for medium intensity, and a 1 second interval for high intensity.

4 Experiment

This section discusses the construct of the experiment. The first subsection elucidates the research questions that the experiment answers. The second subsection describes the setup of the experiment.

4.1 Research questions

RQ1. What is the level of accuracy of Android Runner when measuring energy consumption in native Android apps?

This paper tests whether Android Runner can accurately measure the energy consumption of native Android apps. To do this, we have developed benchmark apps that stress isolated components with minimal code. The results are analysed based on their variance. This will indicate whether Android Runner can accurately measure the energy consumption of native Android apps or not.

RQ2. What is the impact that components in Android phones have on the energy consumption of native Android apps?

This paper seeks to identify which components in Android phones have the most impact on the energy consumption of native Android apps. The aim is to create three categories out of the tested components, namely: high, medium, and low consumption. This will hopefully provide app developers with a better understanding of how they can optimise their apps for energy consumption.

RQ3. How can the execution of experiments involving multiple variations of native Android apps be facilitated?

This paper provides a Python framework that facilitates the execution of experiments involving multiple variations of native Android apps. This framework will allow users to automatically create different versions of their native apps with simple Python code.

4.2 Experiment setup

This section explains the setup of the experiment by discussing each component of it in the subsections below.

4.2.1 Application environment

The benchmark apps are stored in a repository on GitHub, a free software storing and hosting service. To improve version control and experiment concurrency, every folder in the apps subfolder contains a single benchmark app. Therefore, multiple experiments can be run at the same time.

4.2.2 Test devices

The test devices used for this experiment are listed in table 1 below. The optimised row means that the phone is close to stock Android.

4.2.3 Benchmark apps

The subjects of the experiment are the benchmark apps. Some of these apps have a continuous process; therefore, no intensity can be specified. However, other apps are interval-based and thus have an intensity which specifies the time between processes. There are two different intensity sets. For the first intensity

	Mi 9T Pro	**LG Nexus 5X**
Year of release	2019	2015
Price at launch	€399,-	€449,-
CPU	Octo-core 2.84GHz Qualcomm Snapdragon 855	Hexa-core 1.8 GHz Qualcomm Snapdragon 808
RAM	6GB	2GB
Display	6.39" 2340x1080 OLED @403dpi	5.2" 1920x1080 IPS LCD @424dpi
Battery	4000mAh Li-Polymer	2700mAh Lithium-Ion
OS	Android 10 - MIUI 11.0.5	Android 6.0
Optimised*	No	Yes

Table 1. Test devices

set, the low intensity has an interval of 3 seconds, the medium intensity 2 seconds, and the high intensity has an interval of 1 second. This intensity set is denoted as *IS1* in table 2. For the second intensity set, the low intensity has an interval of 15 seconds, the medium intensity 10 seconds, and the high intensity has an interval of 5 seconds. This intensity set is denoted as *IS2* in table 2. Table 2 provides an overview of all the benchmark apps and their intensities.

4.2.4 Experiment method

Android Runner is used to conduct the experiment on every test device. This means that every benchmark app is run on every test device. For every benchmark app that has different intensities, the test is also repeated for each intensity. In total, there are 27 benchmark and intensity combinations. Every benchmark app and intensity combination is run for 3 minutes and is then automatically stopped after this period by Android Runner. Each run is replicated 30 times with 2 minutes between each run, in order to account for the fluidity of energy consumption as a measure. Figure 1 represents the experiment method visually.

The measurements are gathered using the Batterystats profiler. This profiler was introduced to the Android Runner framework to combat the limitations of Trepn. Since Batterystats runs on the PC, it creates no overhead on the test device and thus allows for more accurate measurements as compared to Trepn. [11]

4.2.5 Assumptions

Android apps ask for permission when using specific components or features of a phone. For example, when using the camera. One assumption made in this experiment is that these permissions have been granted to the benchmark apps before the experiment is started.

Functionality	Method	Continuous / interval	IS1	IS2
Baseline	No functionality	Continuous		
Accelerometer	Listens for sensor updates	Continuous		
Ambient light	Listens for sensor updates	Continuous		
Camera	Takes picture at interval	Interval		x
CPU	Computes large factorial at interval	Interval	x	
Display	Plays video at full brightness	Continuous		
GPS	Listens for location updates based on priority at interval	Interval	x	
Gravity	Listens for sensor updates	Continuous		
Gyroscope	Listens for sensor updates	Continuous		
Local storage	Writes 1 MB string to local storage at interval	Interval	x	
Magnetic field	Listens for sensor updates	Continuous		
Microphone	Records audio from microphone	Continuous		
Networking	Sends HTTP Get request at interval	Interval	x	
Room database	Writes 1 MB string to Room database at interval	Interval	x	
Speaker	Plays audio file at half volume	Continuous		

Table 2. Benchmark apps

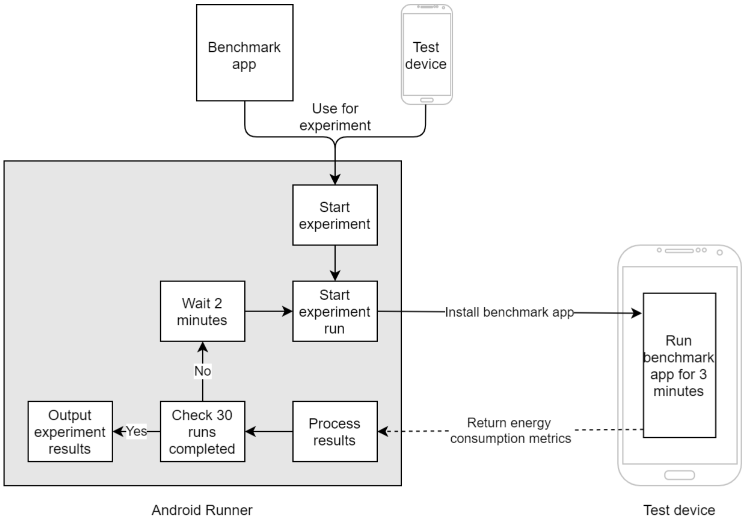

Fig. 1. Visual representation of experiment method

Another assumption made in this experiment is that as little connectivity as possible is enabled on the phone. WiFi, Bluetooth, and location services are all turned off unless required by the functionality that is tested.

The last assumption made, is that all tests are conducted with the brightness of the test device's screen set to minimum.

4.2.6 Replicability

All benchmark apps, code, and results are available in the following GitHub repository: `https://github.com/Luuk99/BachelorProject`. Any further research is free to build upon this repository.

5 Results

This section discusses the results of the different research questions. Each subsection contains the results and findings of a research question.

5.1 Android Runner accuracy results

The benchmark results for the Xiaomi Mi 9T and LG Nexus 5X are visualised by means of violin plots in figure 2 and figure 3 respectively. Summaries of the results can be found in table 5 and table 6 in appendix A. Figure 4 compares the results of both test devices directly to each other.

The summarised results in table 5 and table 6 show that 45 out of 54 tests have a Coefficient of Variation (CV) below 5%. From the other 9 tests, 6 tests are from the networking functionality. For the Xiaomi Mi 9T, the CV of the networking functionality ranges from 7.21% to 11.40%. For the LG Nexus 5X, the CV of the networking functionality ranges from 6.23% to 8.06%.

One of the 3 tests that are left is the microphone test on the Xiaomi Mi 9T that shows a CV of 10.63%. However, the same test on the LG Nexus 5X yields a considerably lower CV of 2.43%. The other 2 tests are the local storage test on high frequency and the Room database test on medium frequency with a CV of 5.10% and 6.89% respectively. The results of these tests on the Xiaomi Mi 9T are notably lower at 0.59% and 0.92%. The same tests on the LG Nexus 5X with different frequencies also yielded lower CVs.

A few tests show interesting results. The first of which, being the local storage functionality test on the Xiaomi Mi 9T as depicted in figure 2. The medium frequency of this test showed a generally higher consumption than that of the high frequency. However, the results for both the Xiaomi Mi 9T and LG Nexus 5X show that there is little difference between the three frequencies. Therefore, the fact that the medium frequency on the Xiaomi Mi 9T exceeds the high frequency can be attributed to variance in the energy consumption of the device.

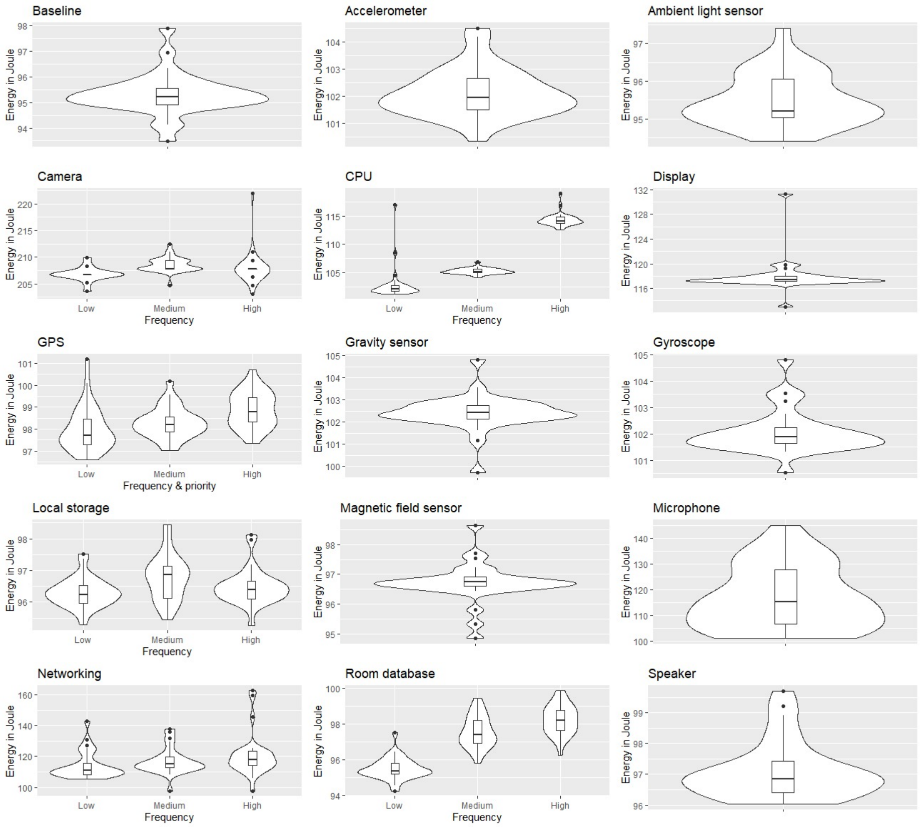

Fig. 2. Benchmark results Xiaomi Mi 9T

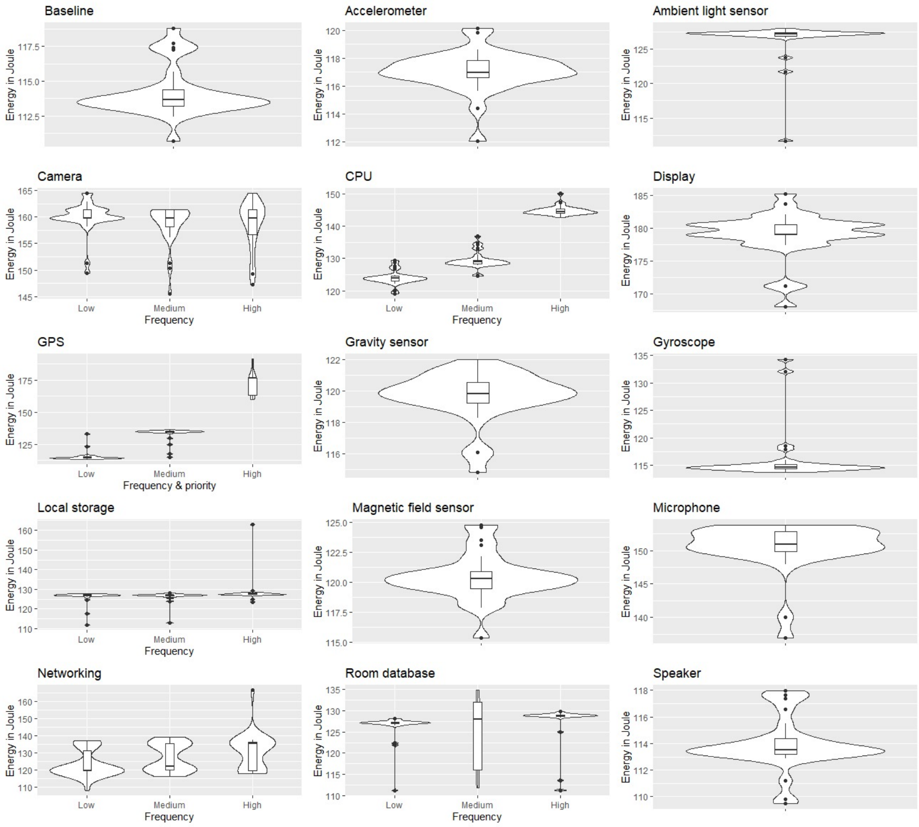

Fig. 3. Benchmark results LG Nexus 5X

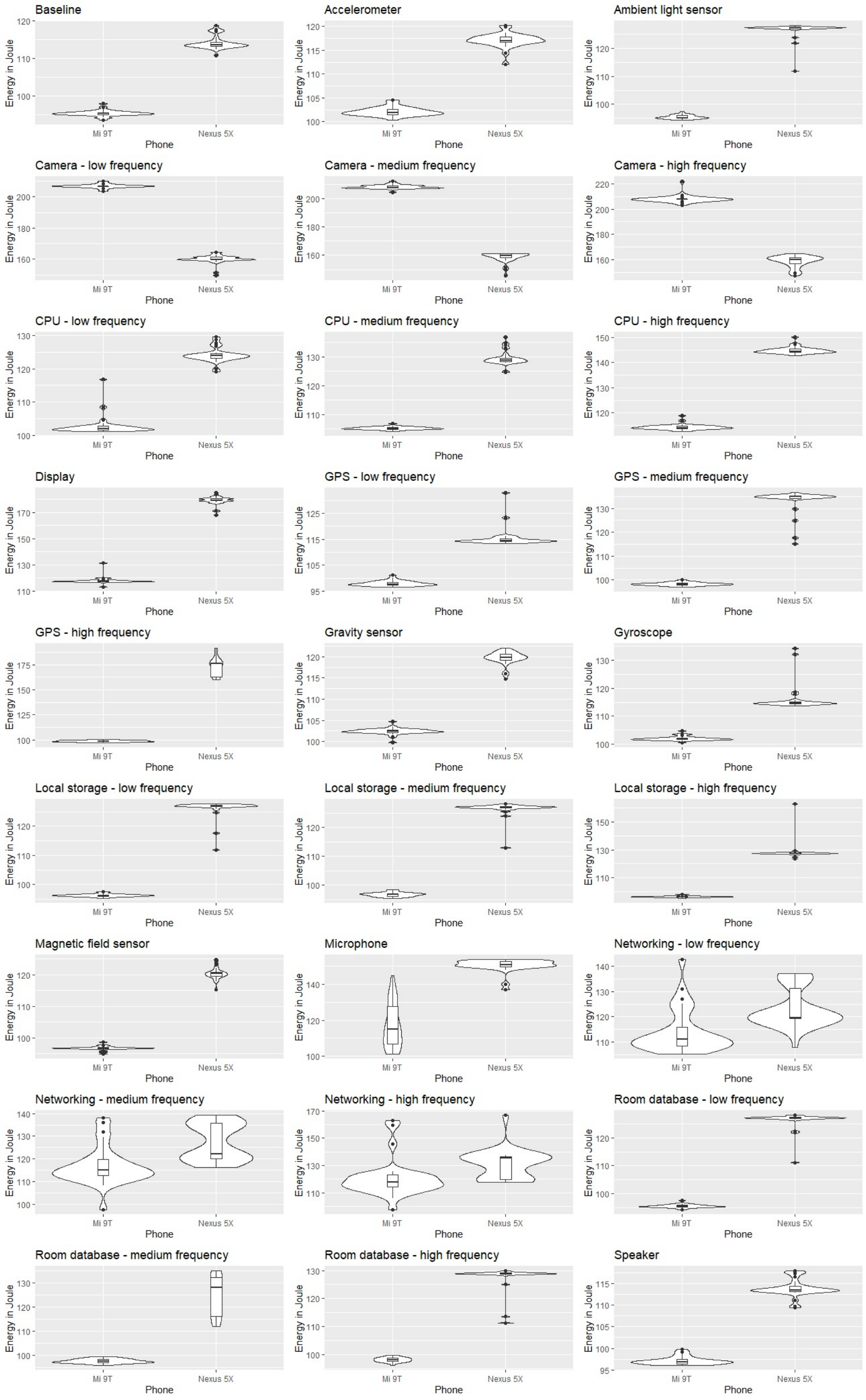

Fig. 4. Benchmark results both test devices

Another interesting test is the networking functionality test. The different frequencies show surprisingly little differences in energy consumption. This can be attributed to energy optimisations, where the network card remains activated after the app makes a network request so that future network requests do not need to reactivate the network card. The GPS functionality test on the Xiaomi Mi 9T also shows little differences between the frequencies. This is also due to energy optimisations. The LG Nexus 5X however, does not show the same level of optimisation and shows more significant differences between the functionalities. The differences between the test devices can be attributed to the fact that the LG Nexus 5X is an older phone, running an older version of Android.

The last interesting test is the Room database functionality test. Figure 3 shows that this functionality tends to consume similar amounts of energy for the three intensities. However, figure 2 shows there are apparent differences between the intensities on the Xiaomi Mi 9T. This difference between the two phones might be attributed to a difference in age and Android version. The Xiaomi Mi 9T is a newer phone and runs a newer version of Android than the LG Nexus 5X. Therefore, the Xiaomi Mi 9T yields better energy optimisations as compared to the LG Nexus 5X.

5.2 Categorisation of functionalities results

To categorise the functionalities, the results of the functionalities are compared to the baseline of both phones. For functionalities with different intensities, the average increase in both the mean and median from the baseline is taken. This is then averaged over both test devices to create one average increase in percentage.

Table 3 shows significant differences between the two test devices. For example, the camera functionality test shows an increase of the mean of 118.01% for the Xiaomi Mi 9T while this is 39.28% for the LG Nexus 5X. Another example is the ambient light functionality test, which shows almost no increase in the mean for the Xiaomi Mi 9T at only 0.18% while the LG Nexus 5X shows an increase of 10.67%. Furthermore, figure 4 shows that the two test devices yield very a different energy consumption for almost every functionality. In most of the tests, the LG Nexus 5X consumes far more energy than the Xiaomi Mi 9T.

The average mean and median increase in table 3 do show a few clear high consumers, as well as a few clear low consumers. These high consumers are the camera, microphone, and display. The high consumers are highlighted in table 3 with bold text for their average mean and median increase. The low consumers are the speaker, accelerometer, gyroscope, magnetic field sensor, ambient light sensor, gravity sensor, local storage, and Room database.

When comparing the local storage and Room database functionality results to the results found by Mohan et al., their results tell a different story. [12] The results found by Mohan et al. show that for IO-intensive workloads, storage can

	Mean increase			Median increase		
Functionality	**Mi 9T**	**Nexus 5X**	**Average**	**Mi 9T**	**Nexus 5X**	**Average**
Accelerometer	7.14%	2.52%	4.83%	7.07%	2.90%	4.98%
Ambient light	0.18%	10.67%	5.42%	-0.02%	11.90%	5.94%
Camera	118.01%	39.28%	**78.64%**	117.87%	40.57%	**79.22%**
CPU	12.77%	16.31%	14.54%	12.45%	16.52%	14.49%
Display	23.79%	56.80%	**40.29%**	23.25%	57.50%	**40.38%**
GPS	3.21%	23.12%	13.16%	3.17%	24.83%	14.00%
Gravity	7.45%	4.80%	6.12%	7.57%	5.43%	6.50%
Gyroscope	7.07%	1.65%	4.36%	6.99%	0.89%	3.94%
Local storage	1.26%	11.24%	6.25%	1.34%	11.86%	6.60%
Magnetic field	1.47%	5.46%	3.46%	1.61%	5.84%	3.72%
Microphone	22.46%	31.84%	**27.15%**	20.93%	32.78%	**26.86%**
Networking	22.82%	11.36%	17.09%	20.33%	10.65%	15.49%
Room database	1.87%	10.26%	6.07%	1.85%	12.60%	7.22%
Speaker	1.88%	-0.24%	0.82%	1.70%	-0.14%	0.78%

Table 3. Average increase of energy consumption compared to baseline

consume more energy than the network, and as much energy as the display. This required larger, 100 MB, files, whereas the local storage and Room database functionality tests in this paper only use 1 MB files.

5.3 Framework implementation results

To facilitate the execution of experiments involving multiple variations of native android apps, we propose a Python framework named *'Apk Generator'*. The typical users of this framework are researchers in the field of energy consumption, especially researchers who utilise or research Android Runner. Researchers can use Apk Generator to create multiple versions of native Android apps without having to alter the code each time manually. For example, researchers can create multiple versions with different intensity levels of the same app. Apk Generator is developed with user experience and efficiency in mind. The resulting framework consists of two components, namely *apk_generator.py* and *config.py.*

The first component, *apk_generator.py*, does all the heavy lifting for the framework. It takes an Android project folder, number of runs, and a boolean, to indicate whether to keep the temporary files or not, as arguments. Once started, it engages a loop that repeats the following steps for the number of runs.

1. A copy of the Android project folder is created and moved to the 'Temp' folder.
2. An iteration over the files in the Java folder of the copied project is started. For every file, the following actions are performed.
   (a) The file name, file path, and current run are passed to the *update_file* function of *config.py.*

   (b) The file is opened and each line is passed to the *update_loc* function of *config.py* alongside the file name and current run.
   (c) All lines returned from *config.py* are written to the file.

3. An APK of the copied and altered Android project is made and moved to the ‘Outputs’ folder.

After all runs have been completed, the ‘Temp’ folder is either deleted or kept, depending on the boolean value provided as an argument.

The second component, *config.py*, consists of two easily modifiable functions that can alter the Java/Kotlin files. The first function is the aforementioned *update_file* which takes the file name, file path, and current run as arguments. This function can easily be populated to alter the file. It then returns either true or false, depending on whether the file has changed or not and should, therefore, be altered on a line of code basis. The second function is *update_loc* which takes the file name, line of code, and current run as arguments. This is an easily modifiable function that can alter individual lines of code. It then returns the line of code, which can be altered or kept the same.

The code snippet below shows a simple implementation of the *update_file* function in *config.py*. This code snippet would replace the line "Hello World" with "Hello there" while leaving everything else unchanged.

```python
def update_file(file_name, file, run):
    for line in fileinput.input(file, inplace=True):
        if line == "Hello World":
            sys.stdout.write("Hello there")
        else:
            sys.stdout.write(line)
    return True
```

The following code snippet shows a simple implementation of the *update_loc* function in *config.py*. It has the exact same functionality as the code snippet of the *update_file* function.

```python
def update_loc(file_name, line, run):
    if line == "Hello World":
        return "Hello there"
    return line
```

Figure 5 provides a visualisation of the workings of the framework.

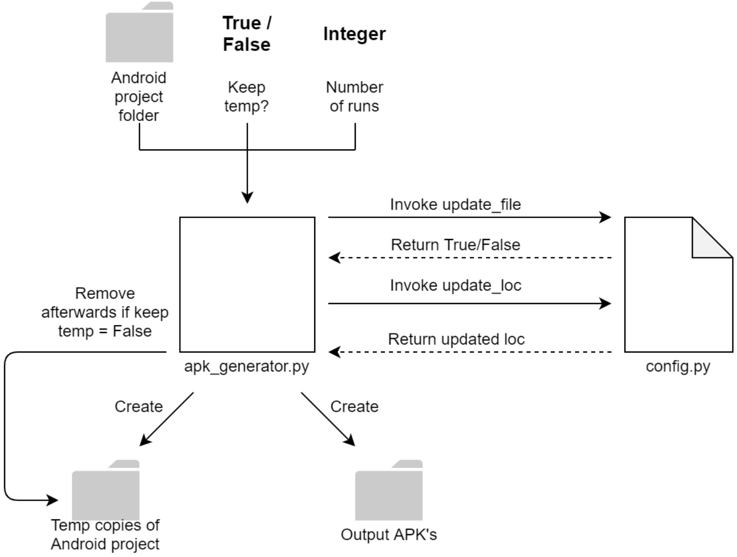

Fig. 5. Visual representation of the Apk Generator framework

6 Discussion of main findings

This section discusses the main findings of the experiment. This section contains two subsections that cover the research questions related to the experiment.

6.1 Android Runner accuracy findings

This section discusses the main findings of the first research question. Each subsection contains one of the main findings concerning this research question.

6.1.1 Overall low variance
When 5% is taken as the golden rule for the Coefficient of Variation (CV), most results seem to indicate a low variance. When we combine the results from both phones, 45 out of 54 tests are below the 5% threshold. These tests thus indicate a low variance. Low variance means that individual runs of Android Runner always result in a measured level of energy consumption that is close to the results of the other runs. Therefore, a low variance is a sign of reliability and accuracy. Thus, the overall low variance indicates that Android Runner is indeed reliable and accurate when measuring energy consumption in native Android apps.

6.1.2 High networking variance
For both phones, the CV of the networking tests exceed the 5% threshold and are, therefore, classified as high variance. This can be explained by the fact that other apps are installed, and processes like automated updates are started when

networking is enabled. This theory is backed up by the difference in results for the two test devices. The LG Nexus 5X has a lower CV than the Xiaomi Mi 9T. This can be attributed to the fact that the LG Nexus 5X has fewer apps installed and is thus closer to stock Android. Not only the other apps and automated updates influence the variance of the networking results, but also by the fact that the Internet adds a level of uncertainty due to delays and latencies which cannot be controlled. For example, some HTTP requests may never get a response, which will most likely decrease energy consumption, or the response is delayed and forces the connection to be open for longer, which increases energy consumption.

6.1.3 Few exceptions

The results also show a few other exceptions from the overall low variance. However, for all these exceptions counts that this is only the case for one of the test devices while the other test device adheres to the overall rule of low variance. This seems to indicate that the high variance of the exception is caused by the test device and not by Android Runner.

6.2 Categorisation of components findings

This section discusses the main findings of the second research question. Each subsection contains one of the main findings concerning this research question.

6.2.1 Big differences between test devices

The results show that the increase in mean and median of the benchmark apps, compared to the baseline, differ a lot between the two test devices. For example, the mean increase of the local storage functionality is a lot higher on the LG Nexus 5X (11.24%) as compared to on the Xiaomi Mi 9T (1.26%). Another example is the camera which indicates an even more significant difference between the two devices. The camera in the Xiaomi Mi 9T results to more than a doubling of the energy consumption of the baseline. In contrast, the camera in the LG Nexus 5X sees an increase of just below 40%, compared to the baseline. This means that there is no general categorisation of the components in terms of their energy consumption that applies to both phones. Therefore, it is strongly recommended for future experiments, related to energy efficiency, to use multiple test devices.

6.2.2 High consuming components

The results show three clear high consumers for both test devices. The average mean and median increase of the camera (78.64% and 79.22%), display (40.29% and 40.38%), and microphone (27.15% and 26.86%) are remarkably higher than the other functionalities. Developers of apps should keep the high energy consumption of these functionalities in mind when working with them. It is advised to limit the use of these functionalities in order to restrict the energy consumption of apps.

6.2.3 Low consuming components

The results also show a few clear low consumers for both test devices. The average mean and median increase of the speaker (0.82% and 0.78%), accelerometer (4.83% and 4.98%), gyroscope (4.36% and 3.94%), magnetic field sensor (3.46% and 3.72%), ambient light sensor (5.42% and 5.94%), gravity sensor (6.12% and 6.50%), local storage (6.25% and 6.60%), and Room database (6.07% and 7.22%) are noticeably lower than the other functionalities. This means that developers of apps can use these functionalities more freely and do not need to limit the use of them for the sake of energy efficiency.

7 Conclusions, limitations, and future works

This section first states the conclusions found in this paper, followed by the limitations of the experiment. The last section recommends some courses for future works.

7.1 Conclusions

RQ1. What is the level of accuracy of Android Runner when measuring energy consumption in native Android apps?

When only the variance in the results is regarded, it seems that Android Runner can accurately measure energy consumption in native Android apps. The results showed that 45 out of 54 tests have a Coefficient of Variation below 5%. This thus indicates a low variance. The other 9 tests can be attributed to two factors. The first factor being that the functionality itself inherits variance; this is the case for the networking functionality. The second factor is that variance is caused by the test device; this is apparent in the other tests where one device shows high variance, while the other device is well within the 5% margin.

RQ2. What is the impact that components in Android phones have on the energy consumption of native Android apps?

The results have shown significant differences between the two test devices in terms of their increase in energy consumption as compared to the baseline. Some of the functionalities behave differently for the two test devices. This difference between the two test devices can be attributed to software optimisations and changing hardware over time. For example, cameras in smartphones have become of higher quality over recent years. This makes it hard to provide a general categorisation of the functionalities. Furthermore, the energy consumption of the functionalities is highly dependent on the way they are tested. For example, as appeared from the comparison between the results of the local storage and Room database functionalities with the results from Mohan et al., some functionalities consume considerably more energy when further stressed. [12] Therefore, the categorisation in table 4, as deducted from the average mean increase over the baseline for both devices, should be taken with a grain of salt.

Low (< 10%)	**Medium (< 30%)**	**High (> 30%)**
Accelerometer	CPU	Camera
Ambient light sensor	GPS	Display
Gravity sensor	Networking	Microphone
Gyroscope		
Local storage		
Room database		
Speaker		

Table 4. Categorisation of components based on average mean increase

RQ3. How can the execution of experiments involving multiple variations of native Android apps be facilitated?

To facilitate the execution of experiments involving multiple variations of native android apps, we propose a Python framework named *'Apk Generator'*. Apk Generator consists of two components, namely *apk_generator.py* and *config.py*. The first component does all the heavy lifting, while the second allows the user to make alterations to Kotlin and Java files within Android project folders using simple Python code. The source code of this tool is publicly available as an open-source project and can be found on the Github page (`https://github.com/Luuk99/BachelorProject`) of this project. Future works are free to build upon this tool and can use the source code without any limitations.

7.2 Limitations

This section discusses the limitations of this paper. Every subsection discusses one limitation.

7.2.1 No comparison to hardware readings

This paper does not compare the Android Runner results of the benchmark tests directly to hardware results of the same tests. This would be the ideal way to measure the level of accuracy that Android Runner provides. However, this fell outside of the scope of this paper and is left for future works.

7.2.2 Small set of benchmark functionalities

This paper uses only a small set of functionalities for the benchmark apps. There are, of course, many other functionalities within current Android phones and possibly even more in the future. For example, Bluetooth and more CPU stressing functionalities would be interesting additions for future works.

7.2.3 Few test devices

The experiment in this paper only uses two test devices. Since the results show significant differences between the two test devices, it is hard to draw definitive conclusions from these results.

7.2.4 Dependence on test method

As discussed in the conclusion, the energy consumption of the functionalities is highly dependent on the way they are tested. For example, the local storage and Room database functionalities are tested using 1 MB files. Increasing the file size could further stress the functionalities and result in higher energy consumption, as the comparison with the results from Mohan et al. has shown. [12]

7.2.5 Apk Generator limitations

The current implementation of Apk Generator is limited to Kotlin or Java files inside the Java folder of the project. This means there is currently no way to alter, for example, XML files or alter files in different folders within the Android project folder.

7.3 Future works

Future works could focus on the aforementioned limitations of this paper. The main limitation of this paper is the lack of a direct comparison between readings from Android Runner and readings from hardware. Future works could do the physical testing using hardware, and together with the findings of this paper, this would create a better understanding of the level of accuracy that Android Runner provides.

In future works, more functionalities within Android phones could be tested to provide a more extensive categorisation of the components in terms of their energy consumption. However, these functionalities could also be used to further test the level of accuracy that Android Runner provides.

Another limitation that future works could solve is the small set of test devices. In future works, more test devices could be used to test Android Runner accuracy. Additionally, these test devices could also be used to create a more general categorisation of the components in Android phones in terms of their energy consumption.

An interesting path for future works would be to find a way to compare functionalities more fairly. For example, this paper showed that using 1 MB files local storage and Room database functionalities consumed little energy, while the results by Mohan et al. showed that using 100 MB files local storage functionality can consume as much as the display. [12] Therefore, future work could focus on finding a fair and objective method to compare the different functionalities.

Future works could also solve the limitations of Apk Generator. A more in-depth research into a framework that facilitates the execution of experiments involving multiple variations of native Android apps could build upon Apk Generator. These future works could expand the features of the framework. For example, to allow the alteration of XML files or files within different sub-folders of the Android project folder.

References

1. Clement, J.: *Number of available applications in the Google Play Store from December 2009 to December 2019.* Statista (2020). Online available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
2. Lim, S.: *Average Storage Capacity in Smartphones to Cross 80GB by End-2019.* Counterpoint Research (2019). Online available: https://www.counterpointresearch.com/average-storage-capacity-smartphones-cross-80gb-end-2019/
3. Lam, C.: *A Python framework for automating experiments on Android.* S2 Group, Vrije Universiteit Amsterdam (2019). Online available: https://drive.google.com/file/d/0B7Fel9yGl5-xc2lEWmNVYkU5d2c/view
4. Hindle, A., Wilson, A., Rasmussen, K., Barlow, E.J., Campbell, J., and Romansky, S.: *GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework.* Proc. of the Working Conference on Mining Software Repositories, pp. 12-21 (2014). doi: 10.1145/2597073.2597097. Online available: http://softwareprocess.es/2015/greenminer.pdf
5. Mittal, R., Kansal, A., Chandra, R.: *Empowering developers to estimate app energy consumption.* Mobicom 2012: Proceedings of the 18th annual international conference on Mobile computing and networking, pp. 317–328 (2012). doi: 10.1145/2348543.2348583. Online available: https://dl.acm.org/doi/10.1145/2348543.2348583
6. Hu, Y., Yan, J., Yan, D., Lu, Q., and Yan, J.: *Lightweight energy consumption analysis and prediction for Android applications.* Science of Computer Programming vol. 162, pp. 132-147 (2018). doi: 10.1016/j.scico.2017.05.002. Online available: https://www-sciencedirect-com.vu-nl.idm.oclc.org/science/article/pii/S0167642317300953
7. Le, H.A, Bui, A.T., and Truong, N.: *An Approach to Modeling and Estimating Power Consumption of Mobile Applications.* Mobile Networks and Applications vol. 24, pp. 124–133 (2019). doi: 10.1016/j.procs.2016.08.028. Online available: https://www-sciencedirect-com.vu-nl.idm.oclc.org/science/article/pii/S1877050916317756
8. Tawalbeha, M., Eardleya, A., and Tawalbeh, L.: *Studying the Energy Consumption in Mobile Devices.* Procedia Computer Science vol. 94, pp. 183-189 (2016). doi: 10.1016/j.procs.2016.08.028. Online available: https://www-sciencedirect-com.vu-nl.idm.oclc.org/science/article/pii/S1877050916317756
9. Cruz, L. and Abreu, R.: *On the Energy Footprint of Mobile Testing Frameworks.* IEEE Transactions on Software Engineering (2019). doi: 10.21227/kb5s-1r43. Online available: https://arxiv.org/pdf/1910.08768.pdf
10. Di Nucci, D., Palomba, F., Prota, A., Panichella, A., Zaidman, A., and De Lucia, A.: *PETrA: a Software-Based Tool for Estimating the Energy Profile of Android Applications.* IEEE/ACM 39th International Conference on Software Engineering Companion (2017). doi: 10.1109/ICSE-C.2017.18. Online available: https://repository.tudelft.nl/islandora/object/uuid:b5ba7baf-539f-4c2c-8cf3-57a3d62366cd?collection=research
11. Hoque, M.A., Siekkinen, M., Khan, K.N., Xiao, Y., and Tarkoma, S.: *Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices.* ACM Computing Surveys vol. 48(3), article 39 (2016). doi: 10.1145/284072. Online available: https://dl-acm-org.vu-nl.idm.oclc.org/doi/10.1145/2840723

12. Mohan, J., Purohith, D., Halpern, M., Chidambaram, V., Reddi, V.J.: *Storage on Your Smartphone Uses More Energy Than You Think*. 9th USENIX Workshop on Hot Topics in Storage and File Systems (2017). Online available: `https://www.cs.utexas.edu/~jaya/pdf/hotstorage17-energy.pdf`

A Summary benchmark results

Functionality	**Freq.**	**Mean**	**Med.**	**Min**	**Max**	**Q1**	**Q3**	**IQR**	**SD**	**CV**
Baseline	-	95.32	95.22	93.49	97.90	94.91	95.54	0.63	0.82	0.86%
Accelerometer	-	102.13	101.95	100.36	104.50	101.51	102.67	1.15	0.89	0.87%
Ambient light	-	95.49	95.20	94.41	97.40	95.04	96.07	1.02	0.70	0.73%
Camera	L	206.82	206.76	203.58	209.94	206.76	206.76	0	1.35	0.65%
Camera	M	208.59	207.80	204.66	212.53	207.80	209.38	1.57	1.42	0.68%
Camera	H	208.01	207.80	203.08	221.97	207.80	207.80	0	3.03	1.46%
CPU	L	102.89	102.09	101.23	116.97	101.70	102.76	1.06	2.98	2.90%
CPU	M	105.26	105.11	104.16	106.85	104.95	105.58	0.63	0.52	0.49%
CPU	H	114.32	114.04	112.61	118.97	113.72	114.79	1.07	1.20	1.05%
Display	-	118.00	117.37	113.03	131.30	117.21	117.99	0.77	2.77	2.34%
GPS (L prio)	L	97.99	97.71	96.60	101.20	97.27	98.46	1.19	1.02	1.03%
GPS (M prio)	M	98.28	98.21	97.02	100.20	97.85	98.57	0.72	0.72	0.73%
GPS (H prio)	H	98.88	98.79	97.36	100.70	98.31	99.43	1.11	0.82	0.83%
Gravity	-	102.42	102.43	99.72	104.81	102.15	102.75	0.60	0.81	0.79%
Gyroscope	-	102.06	101.88	100.54	104.80	101.64	102.24	0.59	0.83	0.81%
Local storage	L	96.30	96.25	95.29	97.52	95.97	96.52	0.56	0.47	0.48%
Local storage	M	96.79	96.86	95.45	98.44	96.11	97.13	1.02	0.73	0.75%
Local storage	H	96.48	96.38	95.27	98.13	96.10	96.66	0.56	0.57	0.59%
Magnetic field	-	96.72	96.75	94.86	98.64	96.59	96.91	0.31	0.66	0.68%
Microphone	-	116.73	115.15	101.15	145.05	106.60	127.83	21.23	12.41	10.63%
Networking	L	114.10	110.99	105.19	142.70	108.29	115.76	7.47	8.69	7.61%
Networking	M	116.82	114.87	97.56	138.00	112.59	119.83	7.24	8.42	7.21%
Networking	H	120.31	117.87	97.49	162.74	114.12	123.24	9.12	13.71	11.40%
Room database	L	95.50	95.34	94.23	97.54	95.18	95.81	0.63	0.62	0.65%
Room database	M	97.57	97.39	95.81	99.45	96.92	98.19	1.27	0.90	0.92%
Room database	H	98.24	98.21	96.22	99.88	97.66	98.77	1.11	0.85	0.86%
Speaker	-	97.11	96.84	96.04	99.70	96.40	97.43	1.03	0.94	0.96%

Table 5. Summary benchmark results Xiaomi Mi 9T

Functionality	**Freq.**	**Mean**	**Med.**	**Min**	**Max**	**Q1**	**Q3**	**IQR**	**SD**	**CV**
Baseline	-	114.20	113.66	110.70	118.79	113.19	114.40	1.21	1.80	1.58%
Accelerometer	-	117.08	116.96	112.08	120.14	116.61	117.81	1.29	1.49	1.27%
Ambient light	-	126.39	127.19	111.68	127.97	126.88	127.35	0.47	3.03	2.39%
Camera	L	159.98	159.77	149.43	164.47	159.77	161.34	1.57	3.03	1.89%
Camera	M	158.63	159.77	145.51	161.34	158.20	161.34	3.13	3.73	2.35%
Camera	H	158.57	159.77	147.24	164.47	156.64	161.34	4.70	4.60	2.90%
CPU	L	124.07	123.90	119.04	129.54	123.11	124.49	1.37	2.20	1.78%
CPU	M	129.53	128.91	124.53	136.90	128.32	129.54	1.21	2.63	2.03%
CPU	H	144.89	144.50	142.70	150.21	144.14	145.32	1.17	1.43	0.99%
Display	-	179.07	179.01	168.12	185.24	179.01	180.57	1.56	3.53	1.97%
GPS (L prio)	L	115.45	114.43	113.28	133.00	114.20	115.20	1.00	3.76	3.26%
GPS (M prio)	M	133.20	134.86	114.97	136.59	134.24	135.29	1.06	5.07	3.81%
GPS (H prio)	H	173.15	176.33	159.90	191.28	162.89	176.33	13.35	9.00	5.20%
Gravity	-	119.69	119.83	114.81	122.02	119.24	120.57	1.33	1.62	1.36%
Gyroscope	-	116.09	114.67	113.74	134.27	114.36	115.13	0.77	4.77	4.10%
Local storage	L	126.12	126.88	111.84	127.66	126.72	127.35	0.63	3.24	2.57%
Local storage	M	126.43	127.03	112.93	128.13	126.72	127.19	0.47	2.66	2.11%
Local storage	H	128.57	127.50	123.59	162.90	127.23	127.97	0.74	6.56	5.10%
Magnetic field	-	120.43	120.29	115.35	124.78	119.44	120.87	1.43	1.94	1.61%
Microphone	-	150.56	150.92	136.96	153.80	149.90	152.83	2.92	3.66	2.43%
Networking	L	123.50	119.69	107.86	137.05	119.38	131.21	11.83	7.81	6.32%
Networking	M	127.32	122.18	116.22	139.25	120.18	135.61	15.43	7.93	6.23%
Networking	H	130.71	135.44	117.84	166.81	119.67	136.16	16.49	10.54	8.06%
Room database	L	126.33	127.12	111.14	128.13	126.89	127.35	0.47	3.16	2.50%
Room database	M	124.33	127.97	111.84	134.86	116.07	132.00	15.94	8.56	6.89%
Room database	H	127.11	128.83	111.21	129.85	128.60	129.07	0.47	5.20	4.09%
Speaker	-	113.93	113.50	109.50	117.97	113.20	114.39	1.19	2.01	1.76%

Table 6. Summary benchmark results LG Nexus 5X