Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Energio: A New Hardware-based Energy Benchmark Platform to Develop Energy Ratings for Android Applications

Author:
Leszek Nowaczyk

Supervisor:
Dr. Anandha Gopalan

Second Marker:
Dr. Naranker Dulay

17th June 2020

Abstract

Battery technology is not able to keep up with the demands of the rest of smartphone hardware. As a result, there exist efforts to decrease power consumption for mobile devices. Users are interested in maintaining a good battery life and want to understand how their battery is drained. While energy ratings are easily accessible for household goods, energy ratings for mobile applications are not. The creation of such rating is a new area and prior to the commencement of this project only one other approach was tested - a software based approach using Orka by a MEng student at Imperial College London.

In this project we create an energy benchmark platform that utilises a hardware based energy measurement approach in establishing an energy rating. The platform allows users to benchmark different applications against each other by uploading UI automation scripts that perform the same actions for each application and comparing the energy used. The UI automation scripts are ran on a Google Pixel 2 smartphone, while the energy consumed is measured using an Otii power monitor. The scripts for each application can be grouped together based on the task performed to create a benchmark that compares energy usage of the apps to create an energy rating.

Further to this, a dataset analysis approach of GreenHub's data is explored to see if gathering energy used this way is a feasible alternative to hardware measurements. We tested two ways of finding the average power used by applications - a battery discharge approach and a voltage current approach.

In addition we validate the platform and the ease of creating new benchmarks using it by creating a benchmark for 5 browsers and 5 Reddit application, and comparing the results against the existing approaches. We found that out of the 5 browsers tested Opera performed the best on the 10 tasks, followed by Google Chrome, Brave, Edge and Firefox. Out of the 5 Reddit applications, in the 6 tasks tested rif is fun has performed the best, followed by Relay, BaconReader, RedReader and the official Reddit app.

The results were partially validated with Greenspector's hardware measurement findings, but were uncorrelated with software approaches. The analysis of Greenhub's dataset has given mixed results. While the voltage current approach has shown better results than the battery discharge approach, it still has a lot of variability across devices and does not give consistent results.

Acknowledgements

I want to deeply thank Dr. Anandha Gopalan for supervising this project, for his continuous support, enthusiasm, regular meetings and constructive suggestions throughout my work. I would also like to thank Dr. Naranker Dulay for his suggestions and advice for the project.

I wish to acknowledge the insight and help provided by Victor Boddy, Amine Halimi and Dimitris Moniatis in building my hardware setup, as well as Juan Manuael Artega for access to a power monitor for testing the feasibility of my ideas earlier in the project.

I would also like to show great appreciation to Qoitech for providing their Otii Enterprise software license for the purpose of this project and beyond for the department to use.

Finally, I would like to thank my family and friends who have supported me throughout the pandemic to finish this project.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivations

With the growing awareness of energy consumption and its implications on both the environment and costs, people have been investigating ways in which to increase the energy efficiency of tasks or products. This applies especially to the smartphone industry, where the limited improvements in battery technology have forced engineers, developers and users to consider energy efficiency when designing mobile software or hardware. Research has shown that users are conscious of their battery usage, with applications that are energy wasteful receiving a lower rating on Google Play store [1]. Other findings also show that users want to understand how their battery is drained and how they can control it [2].

While there exist wide spread energy ratings for housing, white goods or cars, there is a clear need for an energy rating system for smartphone applications that allows users to apply the same energy conscious mindset when installing applications.

Another motivation for establishing an energy rating is that research shows that the worldwide energy consumption of smartphones is as high as 0.6% [3], and that mobile applications contribute to at least 6% of global digital CO_2 emissions [4]. Therefore establishing an energy rating that could have even the slightest impact on energy usage could have significant implication on the global scale.

1.2 Objectives

This project aims to provide energy transparency to app users, by creating an energy benchmarking platform that can be used to create an energy rating for Android applications. Such a platform, would allow developers to assess how energy efficient their application is relative to competitors and allow an energy rating to be created. The end goal would allow users to make more conscious decisions about the applications they install and push developers to think about improving their energy rating and having energy in mind when building applications.

While hardware energy measurement techniques have been developed, there have been no attempts prior to the commencement of the project at using these measurements to establish an energy app rating. Since hardware measurements measure power consumed directly, they are the most accurate way of measuring energy. The feasibility of creating a benchmarking solution and a supporting UI automation process should be tested. Applications across the same category can be tested for battery usage when running the same tasks on the same hardware. In particular browsers and Reddit applications are a good category to start testing since they provide users with similar functionality. The results of these relative measurements can serve as a means to calculate the power used and establish an energy rating.

To assess the benchmarking platform, not only can results from past research be considered, but also a data analysis approach taken on an existing battery usage dataset. While results from such analysis estimate the usage, enough data points can give an idea of how energy efficient an app is. Approaches of extracting power from the dataset have to be explored and a query built to extract the average power for a given application.

1.3 Contributions

1. This project introduces Energio - the first hardware-based energy benchmark platform that can be used for energy measurment and establishing energy ratings for Android applications. Energio allows users to upload UI automation scripts to get the energy usage over time and the average power used. By uploading scripts that accomplish the same tasks on different applications, the average power used can be compared to create benchmarks.

2. Energio is available at http://energio.co.uk/ (only when the server is running on a computer). The website allows developers to get hardware energy usage without needing to know the hardware implementation or having access to the physical device.

3. Two categories of applications, browsing and Reddit applications, are benchmarked using Energio to show the feasibility of the platform. Five applications in each category are compared against each other and are given a rating in Section 4.1.4.

4. The Greenhub battery usage dataset is explored to show an alternative software based approach to getting energy usage of applications. In particular two approaches of gathering average power used for applications were explored - a battery discharge approach and a voltage current approach. Queries built in this project can serve as basis for future dataset exploration. Results can be seen in Section 4.2.

5. Energio is open source and available at https://github.com/kyczawon/energio. This gives others the opportunity to build on top of the platform and findings in this report. It also allows developers using the platform to understand why anything might have gone wrong and allows the community to better understand how Energio works.

Chapter 2

Background

2.1 Battery technology

While the processor speed, memory size and disk capacity is experiencing growth at Moore's law level, improving by factors of 800, 120 and 200 respectively between 1990 and 2003, in the same period of time battery density has not even tripled [5]. This is due to chemical limitations of different energy storing compounds, which is illustrated in Figure 2.1. As a result, today's battery capacity is governed by battery volume, which is directly correlated with device screen sizes [6].

Because of these energy limitations, the mobile CPU market has been dominated by the energy efficient ARM CPUs. As software becomes more complex and the hardware demands increase (such as with the introduction of 5G), CPUs not only have to cope with increased performance, but also meet these energy constraints.

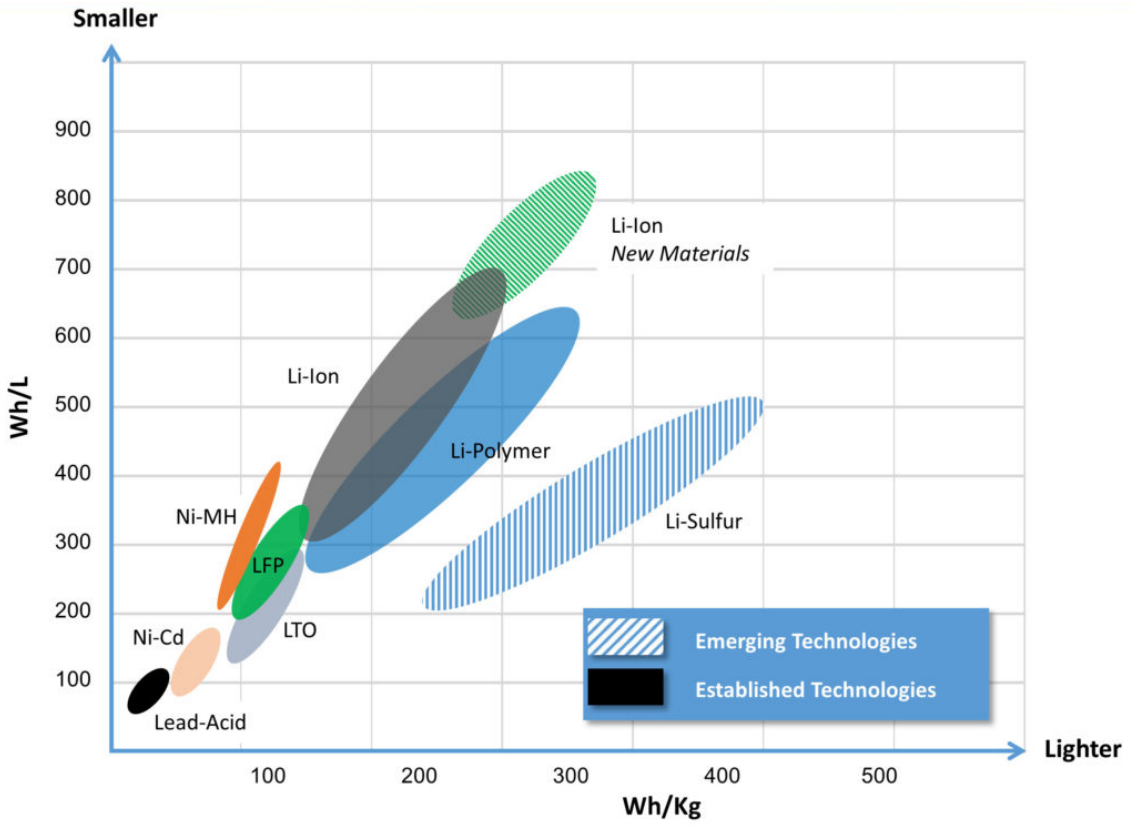


Figure 2.1: Battery Technology Energy Densities, Source: Inventus Power

2.2 Android Operating System

Android is a mobile operating system based on a Linux kernel. The company creating the system was acquired in 2005 by Google and the Android Operating System was released in 2007. The open source nature of Android, the fact that it can be used in modified form, enhanced or redistributed royalty free[7] and the backing of Google has helped it become the market leader. As of May 2020, it has a 72.6% market share as seen Figure 2.2 [8]. The latest version of Android is Android 10, which was launched in September 2019.

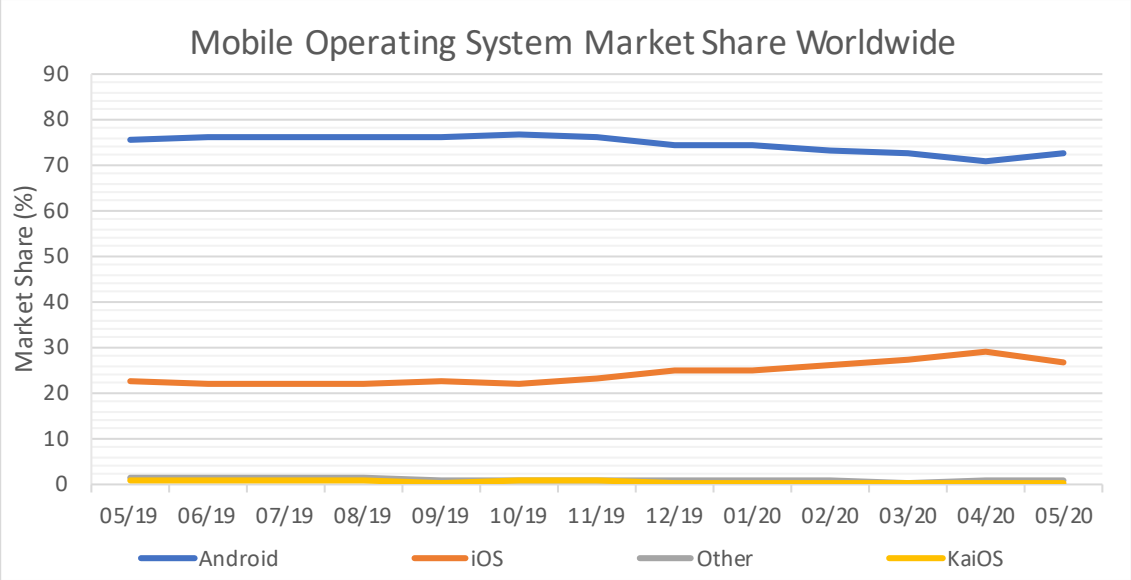


Figure 2.2: Mobile OS Market Share

2.2.1 Android Applications

Android applications (apps) are bundled into an Android Package (APK) that contains the compiled code for different ARM, x86 and MIPS architectures, as well as other files such as assets, resources, classes.dex that provide more information and files needed by the application to be installed and ran.

2.2.2 Android Debug Bridge

Android Debug Bridge (adb) is a command line client-server program that serves as means of communication with the Android device. It has three components:

1. A client that runs on the development machine. An adb command invokes the client.

2. A daemon (adbd) that runs commands on a device. It is a background process on every device.

3. A server that manages the transfer of data between the client and the daemon. It is a background process on the development machine.

When adb is run, an adb server is started (if not running already), which listens on TCP port 5037 for a command from an adb client. It automatically searches and establishes TCP connections to all deamons running on ports 5555 to 5585. Adb can be run over a USB connection or Wi-Fi. In both cases, USB debugging needs to be allowed on the Android device in programming settings. To connect over Wi-Fi, the target device needs to be set to listen on the port 5555 by sending an adb command `adb tcpip 5555` over USB cable.

2.3 Hardware Measurements

The most accurate way of measuring the battery consumption on a device is by directly measuring the power consumed with the help of a power monitor or a scope.

To be able to measure the power consumed by a phone there needs to exist a way to connect to the positive and negative connections of the battery. However, newer phone batteries come with custom connectors, where tapping onto the relevant pins is very difficult. As a result it is easier to take apart the battery and detach it from the custom connector that connects it to the motherboard. The power monitor connection leads can then be soldered onto the place where the battery is normally connected and measurements can be made. However, while this solves the problem of connecting measurement apparatus, it has taken a source of power away.

It is much safer to use an external power source rather than trying to connect the battery back to the leads. In fact, the already soldered connection leads can be connected to both a power monitor and a power supply at the same time. However, as it turns out smartphones will not boot with this setup.

The phone needs more current during boot up than during normal operation and the current needed quickly changes. Due to the long leads a high loop inductance is created. The drop in voltage across an inductor is represented by the equation $V = L\frac{di}{dt}$. As the spikes in current have high frequency, $\frac{di}{dt}$ is large resulting in a voltage drop too high for the phone to handle.

Figure 2.3 shows a simplified circuit when a battery is normally connected. As the connection leads are very short, the inductance is very small and the phone is able to handle the voltage drop across the inductance of the wires during high frequency spikes.

However, when long connections in the form of leads are used, to allow the phone to boot, capacitors have to be added to form a circuit like in Figure 2.4. The right side of the circuit mimics the circuit with short connections before, as long as the capacitors are placed as close as possible to the load (phone). The capacitors will then be able to provide the instantaneous changes in voltage when the voltage is dropped. Also to decrease the effects of the equivalence series resistance (ESR) and equivalence series inductance (ESI) of the capacitor, the capacitors can be placed in parallel.

Qoitech, a company that has tools to help measure mobile energy consumption, recommends 0.5 Farads of total capacitance with voltage rating of at least 50% higher than is expected to be supplied [9].

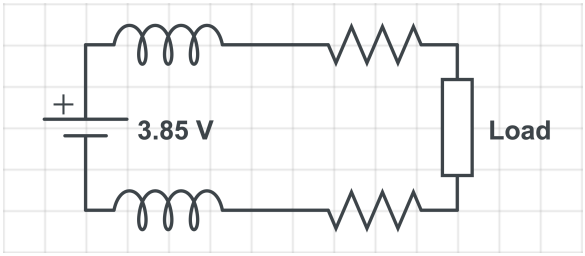


Figure 2.3: Simplified Battery Connection Circuit

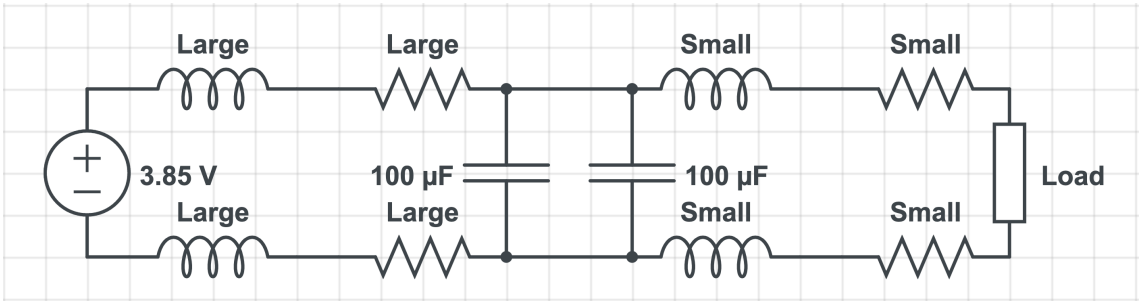


Figure 2.4: Simplified Power Supply Connection Circuit

2.4 Energy Models

Energy models are a mathematical approach to estimating the battery usage of mobile apps based on available software indicators. They do not require any hardware, however most models are specific to a particular device and therefore scaling this energy estimation technique becomes difficult.

2.4.1 SEMO

SEMO [10] developed by Ding et al. is a monitoring tool that uses the state of the battery such as power remaining or temperature of the battery to estimate the power usage of different apps. The SEMO system comprises of:

- an inspector - warns users if the battery is in an undesired state
- a recorder - records the battery usage every minute
- an analyzer - uses the battery data saved to analyze the predicted battery usage and can rank apps accordingly

2.4.2 PowerBooter

PowerBooter [11] is a power model construction technique that relies on battery voltage sensors and knowledge of battery discharge behavior. The authors of PowerBooter also claim that implementing new models for new devices is simple with PowerBooter. PowerTutor uses the models generated by PowerBooter to do online power estimation.

2.4.3 E-Surgeon

There has been a lot of research conducted in the field of energy models. One such solution is E-Surgeon [12] by Noureddine et al., which consists of a system monitoring library called PowerAPI and a software monitoring agent called JALEN. PowerAPI supplies E-Surgeon with information about hardware devices and JALEN uses bytecode instrumentation to estimate (in real-time) the power consumption of a Java application.

2.5 Software Measurements

2.5.1 GreenHub Project

GreenHub is an open source project that relies on crowd-sourcing to gather battery usage data from users who install the GreenHub BatteryHub Android app. To encourage regular users to download it, the app offers real-time battery statistics and monitoring of the current status of the phone. As users use the app, information about their battery usage is uploaded to the GreebHub servers.
The project provides developers with 2 tools to access the following data:

1. **GreenHub Farmer** - provides REST API to query the dataset as well as the option to download the whole dataset as csv.
2. **GreenHub Lumberjack** - a command line tool that allows the user to more easily query the REST API.

At the time of writing the downloadable dataset was last updated in August 2019 and contains over 38 million samples. The structure and all fields available in the GreenHub dataset are shown in the entity relationship diagram in Figure 2.5.

In summary, GreenHub contains the following tables:

1. **Devices** - information about the devices enrolled - model, manufacturer, version, whether it is rooted etc.
2. **Samples** - acts as a join for the rest of the tables, but also contains generic information about the sample such as battery_level, memory occupied, timestamp etc.

3. **App Processes** - contains detailed information about applications running at a given sample - application name, where it is running (foreground, service etc.), version etc.
4. **Settings** - information about system settings such as Bluetooth, power saver, location, NFC, flashlight etc.
5. **CPU Statuses** - detailed information about CPU - CPU utilization, up time, usage etc.
6. **Storage details** - detailed information about CPU - size of storage, how much is free etc.
7. **Battery details** - information about the state of the battery, such as current information, health, voltage etc.
8. **Network details** - network information, such as signal strength, speed, status of both Wi-Fi, mobile network etc.

This dataset is growing every day (although the downloadable dataset was last updated in August 2019) and by analyzing the different statuses and processes it has the potential to serve as a basis for an energy app rating.

2.5.2 Orka

Orka is a software energy estimation technique that uses the findings of Linares-Vasquez et al [13] who found that the majority of energy is spent calling the Application Programming Interface (API). Orka uses 807 API call energy cost estimates to estimate battery usage of an app. Orka was developed by B. Westfield in 2015 [14] to provide feedback on app energy drain at method level. Then in 2018 it was extended by A. Cornet to provide feedback at source-line level [15]. Orka works by asking a developer to provide a monkey runner script that represents typical use of an application. It then uses bytecode analysis and performs log injection by decompiling and recompiling the APK file. It runs the APK file on an emulator or physical device, builds an execution trace based on logs, and monitors other information such as BatteryStatsInfo.bin or network usage, to access hardware costs of running the app. At the end of execution Orka outputs 2 CSV files that show the estimated hardware and routine costs of running the app through the provided scenario.

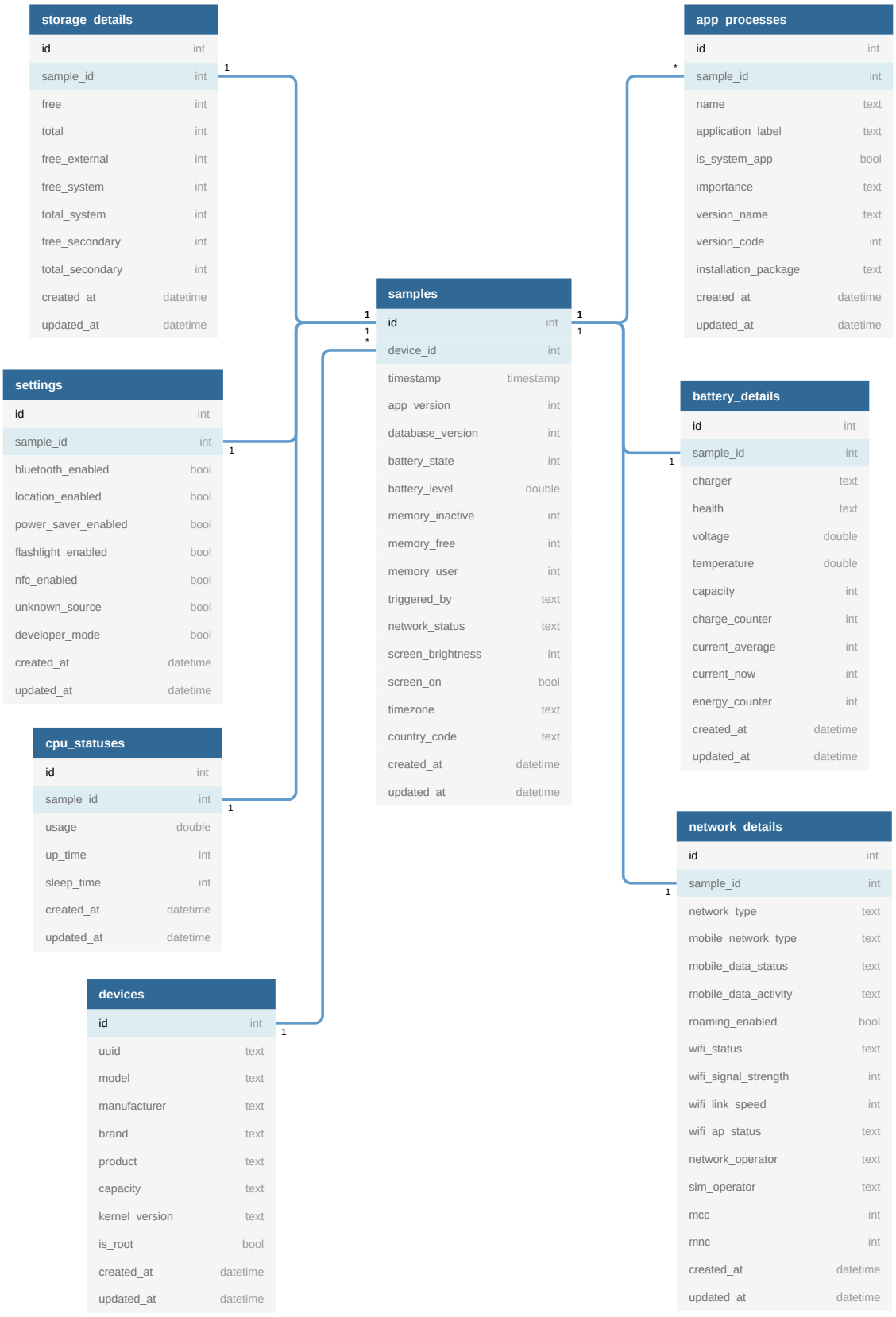


Figure 2.5: Greenhub Database Entity Relationship Diagram

2.6 Android UI automation tools

For the purpose of hardware benchmarking, apps of the same category have to be tested for energy consumption based on the same tasks. To speed up the process of gathering energy measurements and make the task execution consistent, scripts have to be written for each app which navigate through the UI. There exists a wide range of existing UI automation tools which can help solve this problem. Since benchmarking aims at testing apps without having the source code, the UI automation tool has to work reliably in a black-box scenario.

2.6.1 UI/Application Exerciser Monkey

The Monkey [16] is a command line tool that is capable of generating pseudo-random user events such as clicks, touches, or gestures, and some system-level events.

The tool is configurable with settings such as:

- number of events executed
- type of events and frequency
- debugging options like ignoring crashes or timeouts
- operational constraints, such as limiting the tool to a package

2.6.2 DroidMate

DroidMate [17] is a fully automated UI tool that explores the app. In contrast with Monkey it does not just perform random events, but instead reads the runtime GUI and makes a decision on what event to click based on the exploration strategy set out. It continues execution until the stopping criteria has been met. Droid mate can be run as both a command line tool or through its Java API. As input it takes an Android APK and as output it provides a serialized Java object that represents the exploration.

2.6.3 Monkeyrunner

Monkeyrunner [18] is a command line tool that runs Python scripts that use the Monkeyrunner API to control an Android emulator or physical device. Scripts written using Monkeyrunner can control multiple devices at the same time and provide keystrokes to each device. Then by taking a screenshot and comparing it to a saved known correct screenshot it can evaluate whether the test has run correctly. The actions can be recorded in an Android emulator or on physical devices by running the provided monkey_recorder [19]. This speeds up the process of creating the Python script.

2.6.4 UI Automator

Uiautomator [20] is a UI testing framework that allows for black box-style automation by targeting UI components currently displayed on the screen. Uiautomatorviewer, shown in Figure 2.6, is an accompanying program that allows a user to inspect the UI components that are in the foreground on the device. The framework provides a UiDevice class that can be used to access and change device properties, such as device orientation or display size. It also allows simulating a hardware key press, pressing the Back, Home, or Menu buttons, opening the notification drawer or taking a screenshot. UI Automator also exposed an API which developers can use to interact with and manipulate the UI.

2.6.5 AndroidViewClient and CulebraTester

AndroidViewClient is a Python tool that uses UiAutomator as the back end to run automated tasks. Scripts written using the framework target UI elements with UIAutomator BySelectors, meaning views can be targeted with IDs or by text they contain. It also allows interaction with adb and comes with Culebra Tester [21], which is a UI automation tool that allows recording a sequence of moves. Culebra Tester works through a Culebra Android App installed on the target

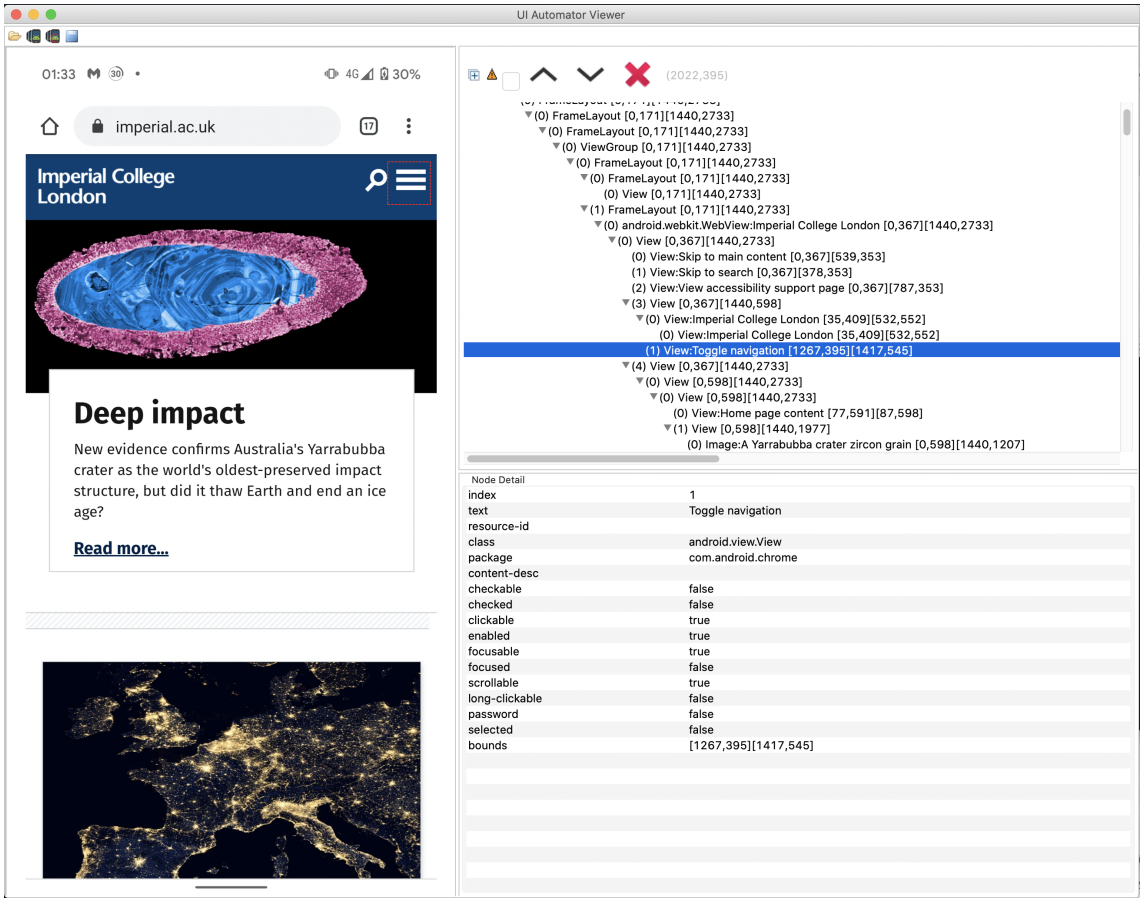


Figure 2.6: Uiautomatorviewer

device and then using a web UI (as seen in Figure 2.7) on the computer to interact with the mobile phone UI. The sequence of the UI clicks can be saved and the test can be run.

Culebra Tester supports exporting the sequence of events into a UI Automator Java or Android-ViewClient Python file, effectively speeding up the process of building UI Automator framework files.

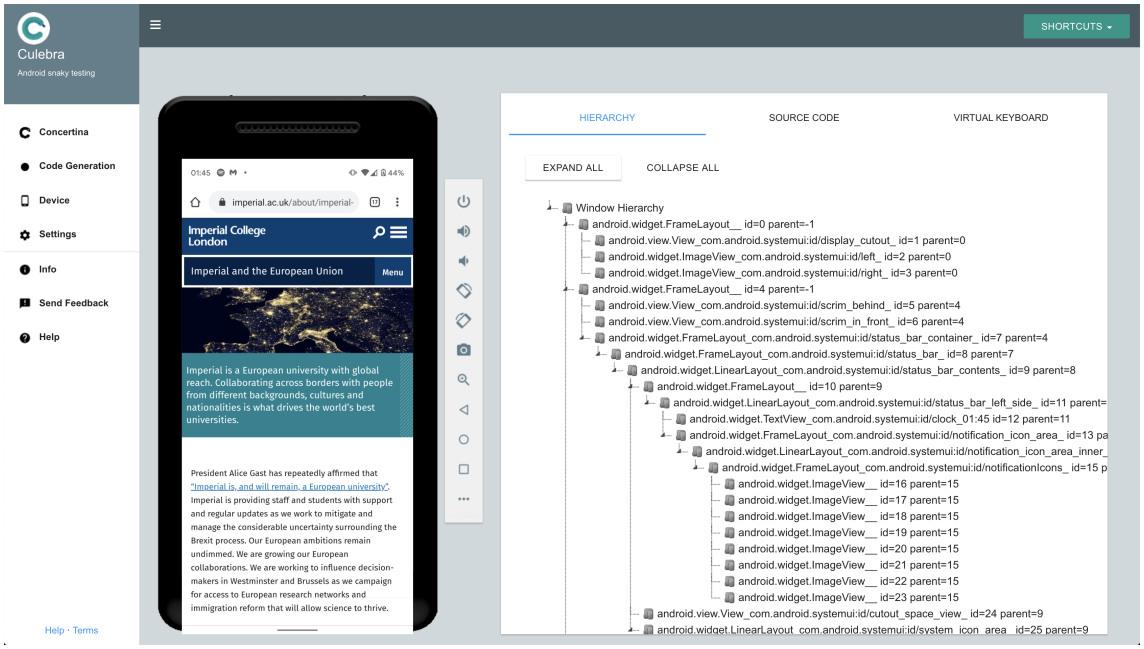


Figure 2.7: Web View of Culebra Tester

2.7 Rating Systems

2.7.1 EU Energy Efficiency

Since 2010, The EU requires products like refrigerators, televisions, air conditioners or washing machines to have an energy label that ranks products based on their energy usage on the scale A-G. Currently, once most products reach the rank A in a particular category, additional A+, A++, and A+++ ranks can be added. However, from 2021 these additional ranks will be removed as they caused confusion for consumers [22]. In 2021, new labels will be introduced too, providing additional non-energy related information through pictograms. They will also contain a QR code which acts as a link to the database that contains more detail information about the product [23]. An example of the new labels, can be seen in Figure 2.8

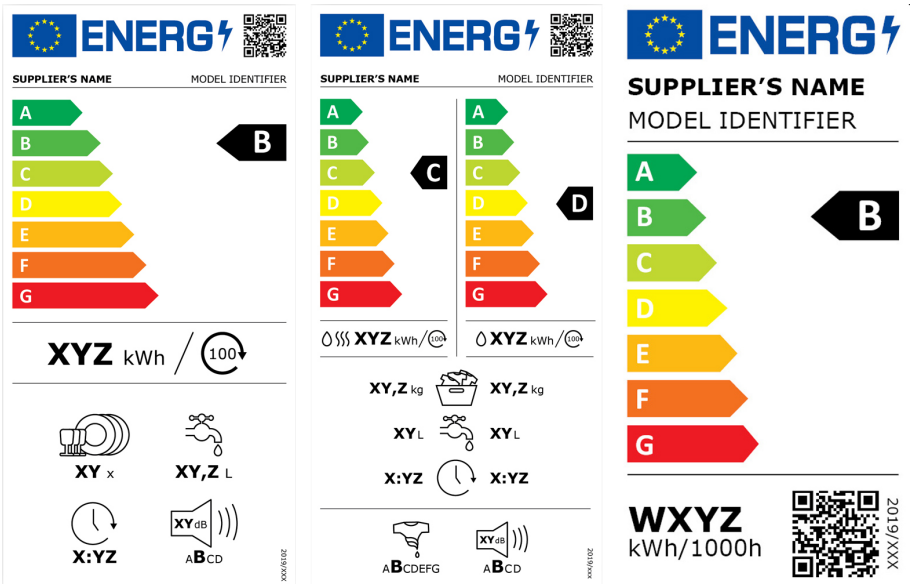


Figure 2.8: 2021 EU Energy Rating Label, Source: buildup EU

2.7.2 Google Play Store Rating

The Google Play Store rating is calculated based on a star rating (between 1 and 5) that users provide the app. Up until May 2019, it was calculated as the average of all scores throughout the life of the app. However since then, the newer scores have been assigned a higher score to reflect how the newer versions of apps have improved [24]. The rating is presented visually through the use of colors - red which is naturally seen as stimulating and disagreeable is used to show negative ratings, while green which was found to be quieting and agreeable is used for positive ratings [25] [26]. As seen in Figure 2.10, in addition the total score is shown big in bold and the sample space is shown below the rating.

When analyzing a large dataset such as the one GreenHub provides, the same logic could be followed to give a better reflection of the how efficient the newer versions of the app are.

2.7.3 Benchmarking

Benchmarking is the process of of comparing performance of different programs based on specific indicators. The biggest disadvantage of benchmarking is that by design it tests very specific tasks usually in isolation. This means that if the benchmark method is revealed, different programs can adapt to get better only in that particular area, effectively cheating the system to gain a better score. To avoid this, benchmarking has to expand to as many indicators as possible to catch programs that perform well only in particular scenarios.

One way to provide an energy app rating would be to keep factors such as hardware and tasks constant, and compare how much energy apps use for those scenarios.

Antutu Benchmark

Antutu benchmark is the most popular benchmarking app in the world with over 10 million downloads. It compares the performance of different smartphones, by seeing how quickly it performs tasks, at what level of detail and at what frame rate. It compares the performance of UX, GPU, RAM, CPU, I/O. Each task is allocated a score, which can be compared with the performance of other devices based on a ranking and percentage of 'defeated users'. An example Antutu score for Google Pixel 3 XL is shown in Figure 2.9.

DXOMARK Camera Benchmark

DXOMARK is a manual benchmark that checks the performance of different smartphone cameras across the same scenarios for photos and videos. The engineers check for indicators such as exposure and contrast, color, autofocus, texture, noise, artifacts, flash, and stabilization. Although this manual testing is time consuming compared to automatic testing like Antuntu, due to its precise measurements and transparent criteria, it is still considered as the leading way of comparing phones for photo and video. The final score is calculated by taking a linear combination of the individual scores and is represented by a number (current scores are around 100). An example of the score for Huawei Mate 30 Pro 5G can be seen in Figure 2.11.

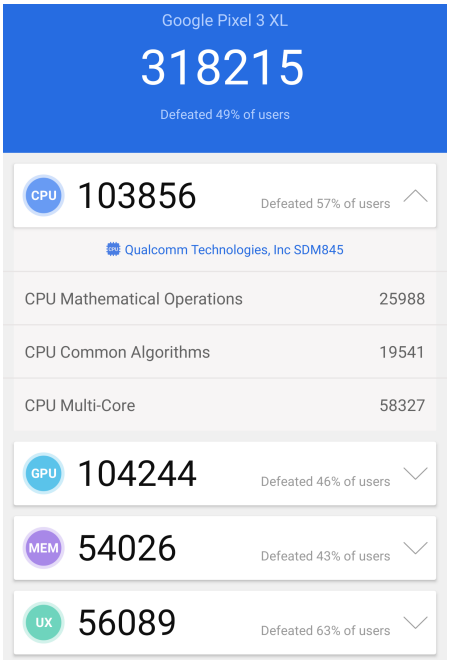


Figure 2.9: Antutu Benchmark Score

Figure 2.10: Google Play Score

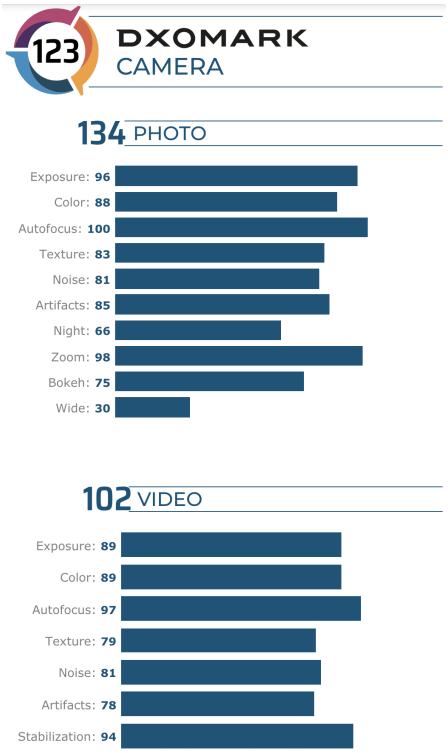


Figure 2.11: DXOMARK Score

2.8 Smartphone Energy Rating

2.8.1 Aeon

Aeon [27] is a software based system, that is deployed as a web application whereby specifying an app category and uploading an APK, an app energy rating is provided. Aeon calls Orka (which has been extended to work with newer APIs) to estimate the energy used by an application. Aeon creates an energy app rating by benchmarking against other crowd sourced energy results in the same category. The rating is given in the same format as the EU Energy Efficiency rating.

Aeon allows the energy of the apps to be evaluated either with a supplied Monkeyrunner script or by using DroidMate2 to automatically test the app. It then also creates a statement coverage report using ACVTool to ensure that DroidMate2 automated exploration is representative and comparable across apps.

The biggest drawback of Aeon's energy rating system is that it currently is unable to provide energy feedback for every app due to crashes. This is particularly true with obfuscated black-box apps on the Google Play store. In the worst case, the authors of Aeon reported only 58% of the applications to be successfully assessed.

The other disadvantage of using Orka as the underlying grading system is that it is based on energy estimates for API calls from 2014, which with dynamically changing hardware and operating system updates may no longer be accurate.

2.8.2 GREENSPECTOR App Mark

In November 2019, GREENSPECTOR, has launched the first efficiency indicator for mobile applications [28]. In its Playstore Efficiency Report [4], GREENSPECTOR judges applications based on 5 criteria:

1. **Inclusion** - The application should work with older versions of Android, on older devices and at unstable network conditions. It must also not exclude users with disabilities by complying

with accessibility standards.

2. **Sobriety** - The application should limit the amount of energy, resources (CPU, memory) and network it uses.
3. **Performance** - Applications must be fast to launch and loading times should be acceptable in all network conditions.
4. **Discretion** - The applications does not ask for permissions it does not need and has little to no trackers.
5. **Ecology** - The CO2 impact due to usage of resources by the application and wear on the components should be minimized.

A hundred data points across these categories are collected to come up with the overall 'business ecoscore' out of 100 for the application.

A complementary 'technical ecoscore' also out of 100 is created by measuring the hardware performance on the following 8 tasks:

1. First launch
2. Second launch that is representative of regular launches.
3. Idle foreground
4. Idle Background
5. Rotating the screen
6. State after closing the application
7. Application loading in 3G
8. Application loading in 2G

Greenspector has tested over 1000 applications from 30 categories across the Google Play Store and published some of the results in their Playstore Efficiency Report. While the final rating is revealed, the measurements and score for each category is not, making comparing applications on the energy front not possible.

Further to the study, Greenspector has also released more detailed comparisons for browsers [29], video conferencing [30], and social media networks [31]. All these studies also reveal the battery usage as the discharge in mAh over the testing period.

Chapter 3

Project & Implementation

The purpose of the first part of this chapter is to go through the implementation details of Energio - of how it works, what the technology used is and what design decisions have been taken. The details about the hardware side of the project and the software side are provided. The chapter also details how the platform is built to enable for gathering of benchmarks. The second part of this chapter explores how the power used by applications can be extracted from the GreenHub dataset, and the technical details of how to do so. The reasons for and implementation of a web scraping script to fetch missing data are also provided.

3.1 Energio Overview

Energio is a benchmarking platform where users can upload Android UI automation scripts to measure and compare the energy used by different applications. It groups together results in a presentable way and creates graphs and tables of the results. The goal of the platform is to enable developers to upload UI automation scripts to benchmark applications across each other without needing to know the underlying hardware implementation details. Any failures from the script or platform need to be displayed to the user. To achieve these tasks a way to measure the energy consumption needs to be created, as well as a website platform that can serve as the user interface to the system.

3.2 Hardware Energy Measurement

To perform hardware energy measurements on a phone, a custom power lead connection to replace the battery of the phone has to be made as discussed in Section 2.3. These leads can then be connected to the power monitor to measure the power used by the phone when running specific applications. The process of performing these measurements can be automated by creating a system, with steps as follows:

1. UI automation scripts are fetched from the user.

2. Before running a script on the device, the system should synchronize with the power monitor to start the measurement.

3. The system runs the UI automation script on the device (scripts use Android Debug Bridge command to communicate with the phone).

4. Once the script is done executing, the power monitor should be asked to stopped recording and the results from the measurements fetched to be presented to the user.

Steps 2-4 need to be repeated for each user uploaded script. This system flow is illustrated by Figure 3.1.

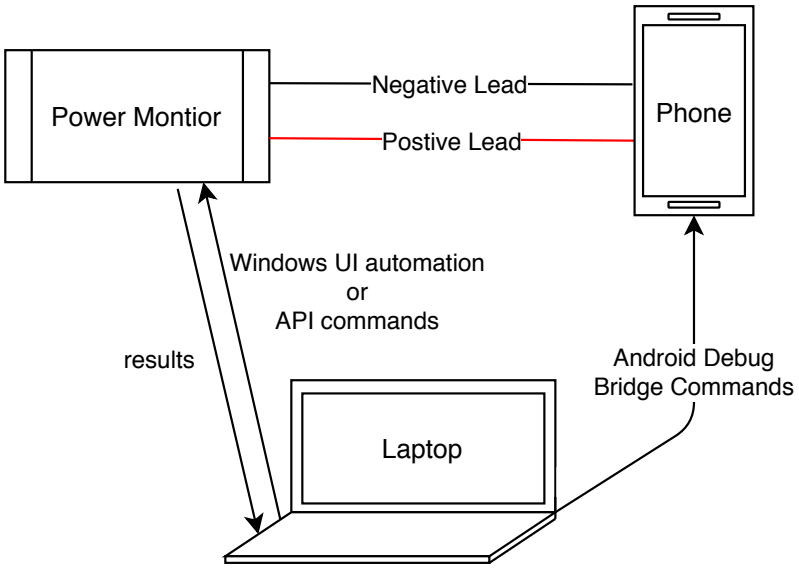


Figure 3.1: Automatic Hardware Measurement System

3.2.1 Device Choice

Google Pixel 2 [1] was selected as the smartphone of choice as it still has Google software support and as a result the latest version of stock Android - Android 10. It is a flagship released in October 2017, so it still has good specs in 2020 (Snapdragon 835 and 4GB RAM), while at the same has a cost of around £100, which is feasible for this project.

3.2.2 Modifying the Google Pixel 2

To access the battery connector that needs to be modified, the screen of the Pixel 2 was first detached with the help of a heat gun, then protection panels unscrewed to detach the battery. Finally, to detach the battery connector, the battery itself was taken apart and the battery leads were unscrewed from the connector.

As mentioned in Section 2.3 to allow the phone to boot 0.5 Farads of capacitance is recommended to be added to the connector. To have an extra margin of safety, 10 100µF capacitors [2] were ordered and soldered in parallel to achieve 1F of total capacitance. The capacitors where soldered on the board directly (to be as close as possible to the circuit), and in parallel, as seen in Figure 3.2. Finally, to avoid exposing an open circuit when the connector is plugged in, it was further insulated as in Figure 3.3. The phone powered solely by a power supply is shown in Figure 3.4.

The phone cannot be connected to the computer with a USB C cable, because the phone would be getting power from the computer, and the energy supplied and measured by Otii would be significantly less. The cable itself cannot be modified to only have the data channel (cutting the power), because the data channel relies on the power channel to know when a device is connected. Therefore, to get good results from the power monitor, the phone has to be connected to the computer through Wi-Fi adb, which slightly decreases the latency in sending commands.

3.2.3 Power Monitor Choice

The power monitor used for this project needs to be able to supply the 3.85V voltage that the battery usually supplies, as well as current in the range of milliamps to a few amps that phone draws. It also needs to be capable of logging the measurements (to match measurements to UI automation scripts that are ran on the device), and have a high enough sampling rate to be able to accurately measure the power consumption over time (at least 100Hz).

[1] https://www.gsmarena.com/google_pixel_2-8733.php
[2] Taiyo Yuden 1210 (3225M)

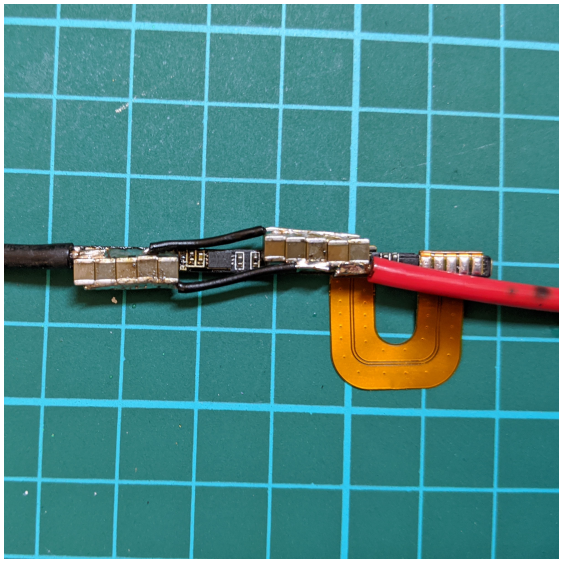


Figure 3.2: Soldered capacitors to battery connector

Figure 3.3: Insulated connector

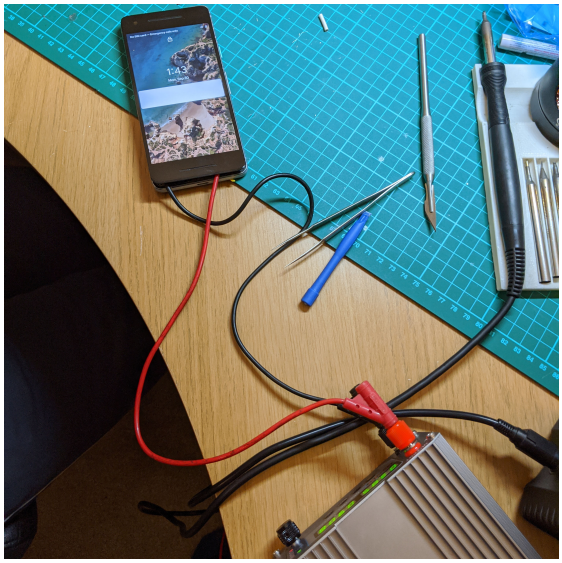


Figure 3.4: Phone powered by power supply only

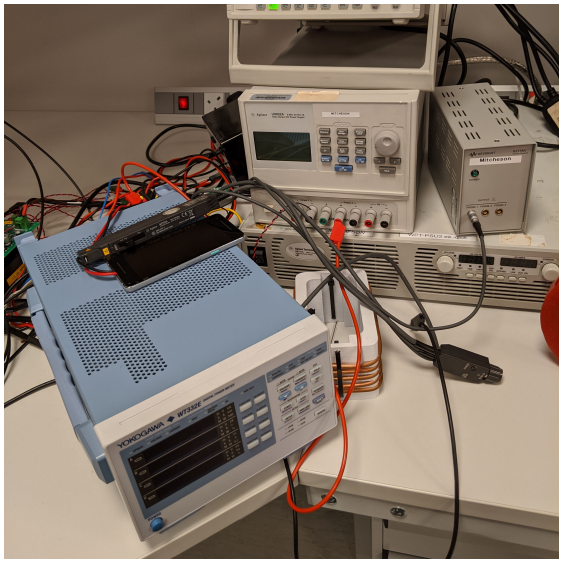


Figure 3.5: Test Power monitor setup

Verification of hardware measurement

A power monitor with specifications above is not available in student electrical laboratories at Imperial College. However, for the purpose of the project, the Power Research Group at Imperial, gave limited supervised access to their Yokogawa WT332E power monitor. The power monitor comes with WTViewerFreePlus software that is able to record data samples and export them into a CSV file. Unfortunately there is no API that programmers can call to collect data. The phone connected to this power monitor is shown in in Figure 3.5.

Despite no available API and the measurement software being only available for Windows, an automated energy measurement system can still be created. This was done by using a Windows GUI automation scripting language, Autoit [3], and the window info tool [4] that comes with it to find the classNameNN properties of UI elements to reference. Using AutoIt allows to automate clicks in WTViewerFreePlus to synchronize starting and stopping the recording as needed. The software automatically creates a csv file after each run and this can be located at a specific path.

While using the power monitor from the Power Group verified that the automated hardware energy measurement is feasible, the fact that the power monitor is used for research and can only be used on request at specific time periods, makes the equipment a bottleneck for the project. Not only would any benchmark software running on this power monitor not be open for use at all time (due to availability), but it would also make the development and testing process really difficult.

Final Power Monitor Choice

As the proof of concept above worked it was decided to purchase a device for the use of this project and beyond. While there are many choices for a power monitor that would fit the specifications required, there are 2 available power monitors with an API that can be easily integrated into automated tasks. The first, Monsoon[5], is the more established in the field of energy consumption with better hardware that is capable of producing a bigger voltage range and support higher currents without the need of an external power supply. The second, newer, power monitor from Qoitech, Otii [6], has a better software stack (supports any language that can communicate over TCP sockets), but these features require an additional enterprise license. The full summary of the specifications of both power monitors can be seen in Table 3.1 below.

After contacting Otii, the company agreed to provide the Enterprise license for the department as long as this project can be featured on their website and future projects mention their name. As Otii has more software features, while being cheaper, it was chosen as the power monitor to buy.

[3] https://www.autoitscript.com/site/autoit/
[4] https://www.autoitscript.com/autoit3/docs/intro/au3spy.htm
[5] https://www.msoon.com/specifications
[6] https://www.qoitech.com/techspec

	Monsoon	Otii
Price with shipping	$974	£455
Sampling rate	5 ksps	5 ksps below 19mA, 1 ksps otherwise
Voltage and Current Ranges	0.8-13.5V, 6Amps continuous current	0-5V, 0-2A continuous current With External Power supply: 0-5V, 0-2.5A continuous current, 5A peak current
Sampling rate	5 ksps	5 ksps below 19mA, 1 ksps otherwise
Current Accuracy	Fine current scale 1 μA: - +/- 1% or +/- 50 μA (whichever is greater) Coarse current scale 100 μA: +/- 1% or +/- 1 mA (whichever is greater)	$\pm$(0.1% + 50nA) for currents below 19mA $\pm$(0.1% + 150μA) for higher currents.
Voltage Accuracy	not available	$\pm$(0.1% + 1.5mV)
API	Python	Python, Java, Matlab
GUI environment	Windows	Windows, Mac and Linux
Measuring inputs	USB, Main channel	Main Channel, Sense, ADC, GPO1, GPO2, TX
Extra features	USB channel can act as data, External power supply AU	Battery profiling and emulation

Table 3.1: Comparison of Specifications of Monsoon and Otii Power Monitors

3.3 Energio User Interface

The Energio platform needs a way for people to interact with the system where they can easily upload new scripts to be tested and view results from their previous benchmarking runs. As the platform requires direct connection to the smartphone and the Otii Power monitor, the back end needs to be running on the same computer. This gives 2 possible approaches to build such a UI - a standalone program or a website. While a standalone program could be done using a technology like Electron that allows the application to be portable across devices, it was decided to go with the website approach. This is because if a web server is set up on the computer, the user interface would be accessible from anywhere and not just available to the person with direct connection to the hardware.

As a result, the Energio website was built and is available at http://energio.co.uk/ (only when the host computer is on) and is hosted on a computer with direct connection to Google Pixel 2 and the Otii power monitor.

3.3.1 Energio Architecture

The website back end is built using Django and front end using Django Templates, HTML, CSS and JavaScript. The website is hosted on the computer with NGINX acting as the web server and uWSGI as the layer communicating between the Django and NGINX. Django-rq and Redis are also used to allow tasks to be queued and executed asynchronously. Each of the technologies is explained in detail in Section 3.3.3.

The Architectural Diagram of Energio is shown in Figure 3.6. The modules in purple (homebrew-duckdns, NGINX, uWSGI, django-rq, Redis Server and Otii TCP Server) is software that is used but not written for the purpose of the project. The modules in gray (Urls, Templates, View Logic, Helpers, Run App Task, Model and Otii) are ones that contain code written to make the website work, these are:

1. **Urls** - Contain the regex URL patterns that the website supports and tells Django which

view logic element to execute.

2. **View Logic** - For each view (page), described in further detail in Section 3.3.4, the correct models are loaded or created and the template module called to return the correct HTML page.

3. **Templates** - For each page a Django template describes how the data passed from the logic should be displayed and what actions are allowed for the user to interface with.

4. **Helpers** - A custom python module that handles unpacking zip files as well as queuing benchmarks for execution.

5. **Run App Task** - A function that runs asynchronously on django-rq and controls the order of execution, when controlling Otii and the Google Pixel 2. For each application task combination, the function clears cache, closes applications, runs the initialization UI script, starts Otii recording, runs the main UI script, gets results from Otii, creates graphs, CSV files, and calculates the average power used by an application.

6. **Otii** - A module that groups Otii TCP Client commands into simple functions that perform Otii functionality needed for measurement. This involves creating Otii objects and calling the relevant functions for starting Otii, stopping Otii, starting a recording, and stopping a recording.

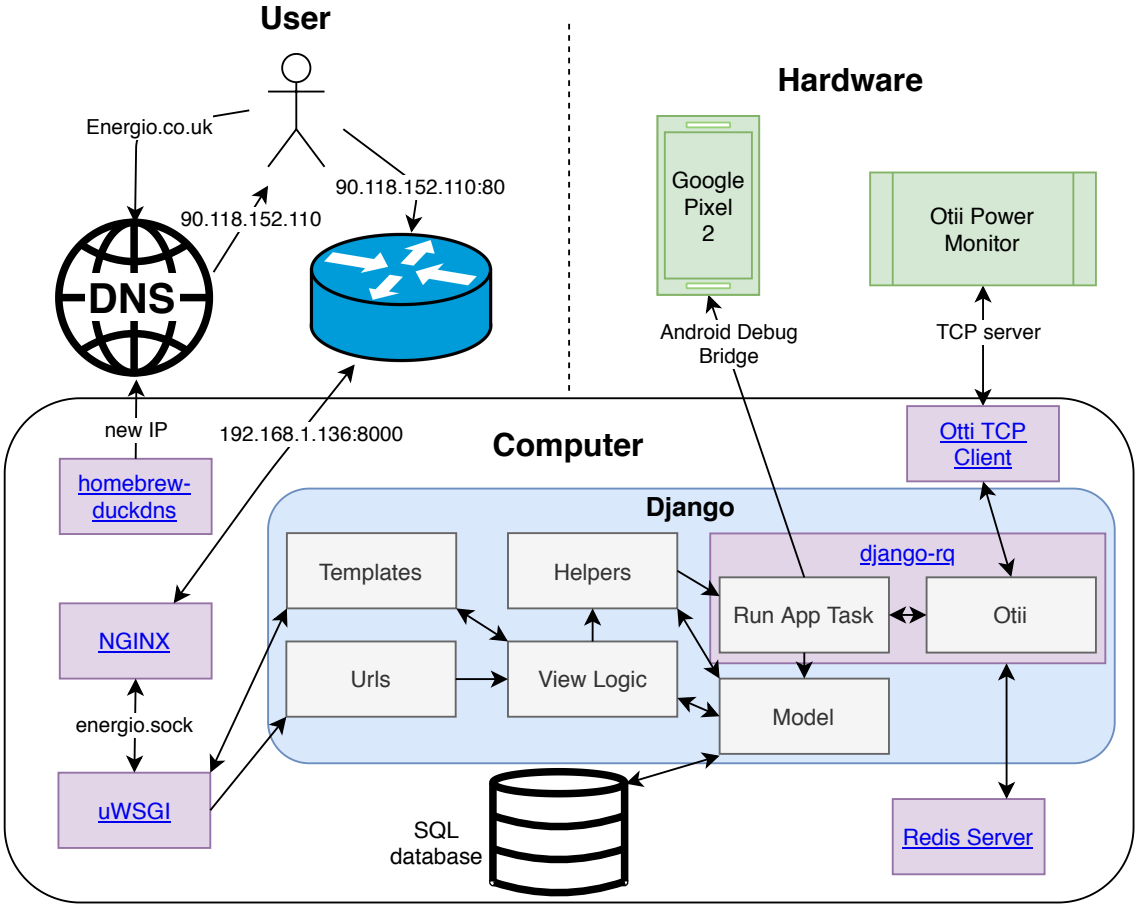


Figure 3.6: Energio Architectural Diagram

3.3.2 User Flows

The following user flows explain how the elements of the architecture work together, by going through 2 typical user flows of the application. Arrows in the Architectural Diagram in Figure

3.6 show the data flow that results from user interaction. The technologies used are explained in Section 3.3.3 and the storage models used in section 3.3.5.

User requests the website

1. The user request the IP address of energio.co.uk from the DNS server which returns the IP address of the router of the network where the website is hosted.
2. The user then gets redirected to this IP address and the router forwards the request using port forwarding to the NGINX 8000 port of the computers static local IP address.
3. NGINX is configured to redirect the user to the upstream uWSGI UNIX socket. It also has the static file path and the upload configured to alias the correct location locally.
4. uSWGI runs Django and forwards the request to Django.
5. Django matches the incoming request against the Regex urls in **Urls** and executes the associated **View Logic**.
6. The **View Logic** takes data from the model, displays it using **Templates** and returns the response back to uSWGI, which is propagated back to the user through NGINX.

User uploads the benchmark

1. When a user uploads a zip file containing the benchmark files, a POST request is sent that is handled by the **View Logic**.
2. The **View Logic** calls the **Helpers** to unpack all the files, create a new Benchmark model object and add a **Run App Task** job to the django-rq queue.
3. The **Run App Task** module goes through each folder of the unzipped file to create a Task object. Each folder contains UI automation scripts to be run for measurement named after the application they are testing. A Result object is created for each script, and for each unique application name in that benchmark a unique App object is created.
4. Before each script is run, depending on the user configuration, the cache of the application is cleared, all applications closed and an initialisation script run.
5. Using the **Otii** module, a message is sent to Otii to start recording, the UI automation script is executed on the phone through adb and the Otii recording is stopped once the script finishes. The measurements are fetched from Otii in the form of Pandas frames. A Measurement object for each samples in the measurement is created and the average power used appended to the Results object. A screenshot is taken of the final state of the application, a Matplotlib graph visualising the data is saved to a PNG file and the measurements are also stored in a CSV file.
6. The cache of the application is cleared.

3.3.3 Technologies used

Otii

With its enterprise feature Otii [32] server allows control of Otii through a TCP socket with any language that supports TCP communication. Otii provides wrappers available for Python, Java and Matlab. Python was chosen in order to natively work with the Django framework.

Django

Django [33] is a Python web framework that follows the Model View Controller design pattern. Django has models that contain the business logic of the application by using object-relational mapping to hide away the database implementation. Views act as controllers handling HTTP request and responses, while templates act as views that generate the HTML response. Django has built in security measures to prevent security issues such as cross site scripting, SQL injection,

clickjacking or cross site request forgery [34]. It also comes with its own web server for development and allows for building reusable modules (called applications) across different projects.

Django was chosen as the web framework for this project, because the rest of the stack (Otii and UI automation scripts) is written in Python.

NGINX

NGINX [35] is a web server, but it can also act as a load balancer, reverse proxy or HTTP cache. NGINX was chosen as the choice of web server as it is one of two recommended web servers by Django (the other being Apache) and tutorials are widely available.

Duck DNS

Since the host computer does not have a static IP address accessible from external networks and the router's external IP address is dynamic, a dynamic DNS service needs to be used to provide the correct IP address for external users.

Duck DNS[7] is a free dynamic DNS hosted on AWS. Since the IP address of the router changes the host computer needs to notify Duck DNS if the router's IP address changes. On Mac OS this can be done with a homebrew-duckdns which runs as a Homebrew service in the background. This creates a DNS record at http://energio.duckdns.org/.

Domain Registration Service Settings

The domain http://energio.co.uk/ was purchased on https://www.123-reg.co.uk/. To link this DNS record to the domain name, the CNAME record in the domain registration service is made to point to http://energio.duckdns.org/.

Router Settings

It should be noted that in order for requests to be redirected to the computer from the router, a static local IP address has to be assigned to the computer on the local network, and port forwarding set up to forward requests to port 80 of router's IP address to NGINX's port 8000 of the computer's IP address.

uWSGI

uWSGI [36] is an implementation of the Web Server Gateway Interface (WSGI) Python standard that is capable of running the Django application taking the requests from the web server and returning responses from Django.

django-rq

Django-rq [37] is a Django application that uses the Redis Queue (RS) [38] python library to handle asynchronous execution of python functions. The RS library requires a Redis Server [39] to be running on the computer to allow for the asynchronous execution and communication to take place. A Redis Server is a data structure store that acts as a database, cache and message broker. To process the queue a Redis Server and rq-worker needs to be started, this can be done with a `redis-server` and `python3 manage.py rqworker` respectively when in the Energio application folder.

User Interface (UI)

Django templates are HTML pages with support for tags that can be used for introducing logic while rendering. For example, loops, filters or if statements can be introduced based on the data passed from the Django view. This allows to display elements from the models.

Any UI web framework that uses HTML, CSS and JavaScript can be integrated with Django. For the purpose of this project the following two were used:

[7] https://www.duckdns.org/

- Bootstrap [8] - a front end framework that allows to style HTML elements and make them interactive by adding classes to tags.

- Dropzone.js [9] - a library that provides a drag and drop feature to upload files.

Otii communication

Otii is connected to the host computer through a USB cable as shown in Figure 3.7. Communication with the Otii power monitor is done through Otii's TCP server and the corresponding Otii TCP Client Python module. To start the TCP server one needs to be logged into the Otii Desktop Application [10] with an Enterprise licence. On start of the Otii application the TCP server is started. Alternatively the Otii server can be started through the command line `otiicli --server --username <username> --password <password>` command. The Otii TCP Client Python module exposes a convenience function that can be called to interact with Otii from Python. The custom built Otii module that bundles Otii commands that are needed uses this TCP client module.

Adb smartphone communication & power monitor connection

To avoid charging the phone through USB and affecting power measurements the smartphone needs to be connected to the computer through adb over Wi-Fi. Firstly, usb debugging needs to be enabled on the Pixel 2 to enable adb and then the phone needs to be connected once to run the tcp server with the command `adb tcpip 5555`. Once this is complete, the phone can be unplugged. Then adb needs to be connected using the following command `adb connect <ip_address>:5555`. It is recommended to assign a static IP address to the phone from the router, so that the command stays the same. Another reason for a static IP Address is that the Helpers module needs to know the IP address, as it needs to reestablish connection in case the phone disconnects when running a benchmark.

For power and energy measurements, the power leads of the phone need to by plugged into Otii's main channel as shown in Figure 3.7. The Otii module sets the voltage to 3.87V as opposed to 3.85V supplied on average by the battery, in order to give some margin of safety.

3.3.4 Energio Pages

The Energio website currently has 6 pages, which allow users to upload benchmarks, view the results from all past benchmarks and control Otii. Energio's pages are in the list below with 2 of the 6 pages in Figures below and the rest in Section A.1.

- **Index** (Figure 3.8) - The index page serves as the home page of the website. It allows users to upload a benchmark with a drag and drop dropzone, select the category of their benchmark, the execution program for the contained scripts and allows to choose options for the benchmark - whether to clear cache, close all applications before running and whether to run the initialisation scripts. The dropzone is created using the Dropzone.js library. On pressing the submit button a POST request is sent to the same page which creates the necessary data structures and queues all the scripts to run on the smartphone and the energy to be measured by Otii.

[8] https://getbootstrap.com/
[9] https://www.dropzonejs.com/
[10] https://www.qoitech.com/download

Figure 3.7: Final Energio Connection Setup

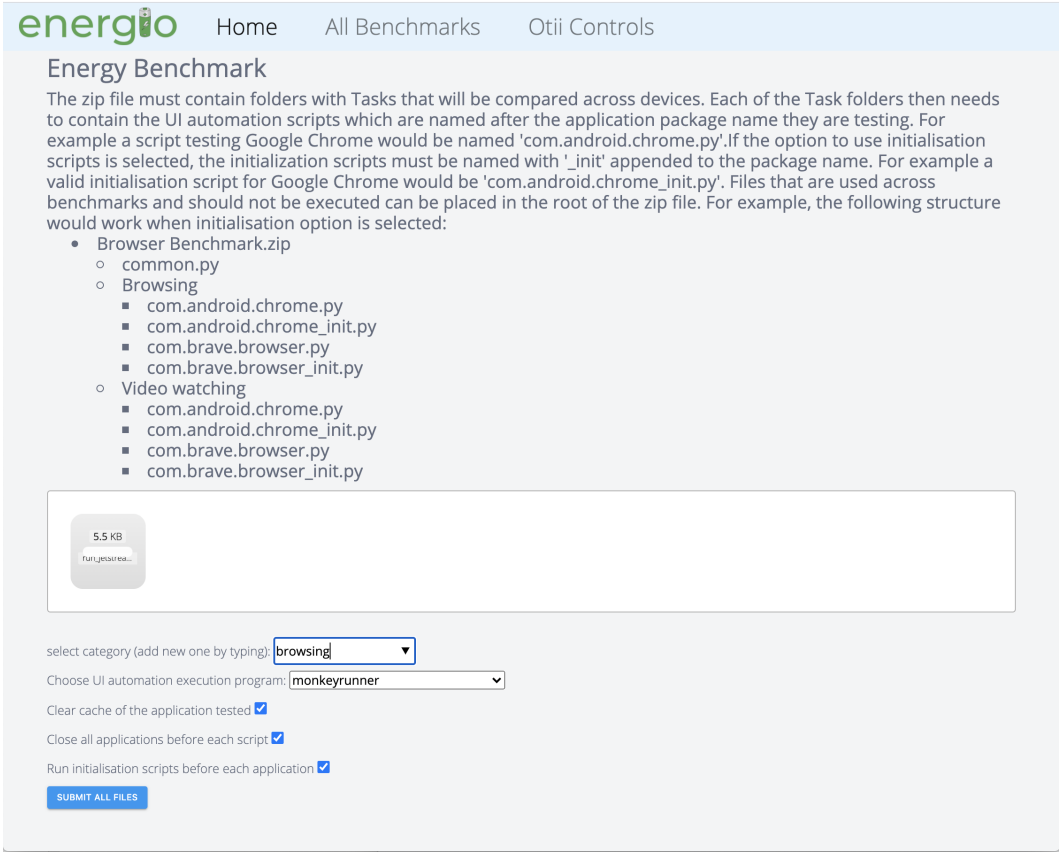


Figure 3.8: Energio Index Page

- **All Benchmarks** (Figure A.1) - Shows all of the past benchmarks in a Bootstrap table.

It gives an overview of the progress of each benchmark using a Bootstrap progress bar - green represents tasks that are finished successfully, red represents tasks that failed and an animated blue shows the task is currently running (not queued, finished nor failed). See Section 3.3.5 for all possible states of a benchmark.

- **This Benchmark** (Figure A.2) - Shows the average power consumed for each task and application in a Bootstrap table with tasks as rows and applications as columns. This makes the power consumption comparable for across the benchmark. It also displays the same progress bar for the current benchmark as in All Benchmarks page. It also displays the logs for all of the task below the table.
- **Result** (Figure 3.9) - Displays the measurements collected by Otii in a table for that particular script, as well as displays graphs of the result and allows for a CSV download. It also shows the screenshot of final state of smartphone after running the state, and has a toggle button to show hide logs for this run.
- **Otii Controls** (Figure A.3) - This page serves as a simple way of controlling Otii - it allows to turn the main channel to 3.87V required by the phone and turn it off. This page can be hidden behind a log in screen to protect unauthorized people from interacting with the page.
- **django-rq** -A page created by the django-rq application that allows to see the RQ jobs currently enqueued and modify them. This page is behind a log in screen to protect unauthorised people from interacting with it.

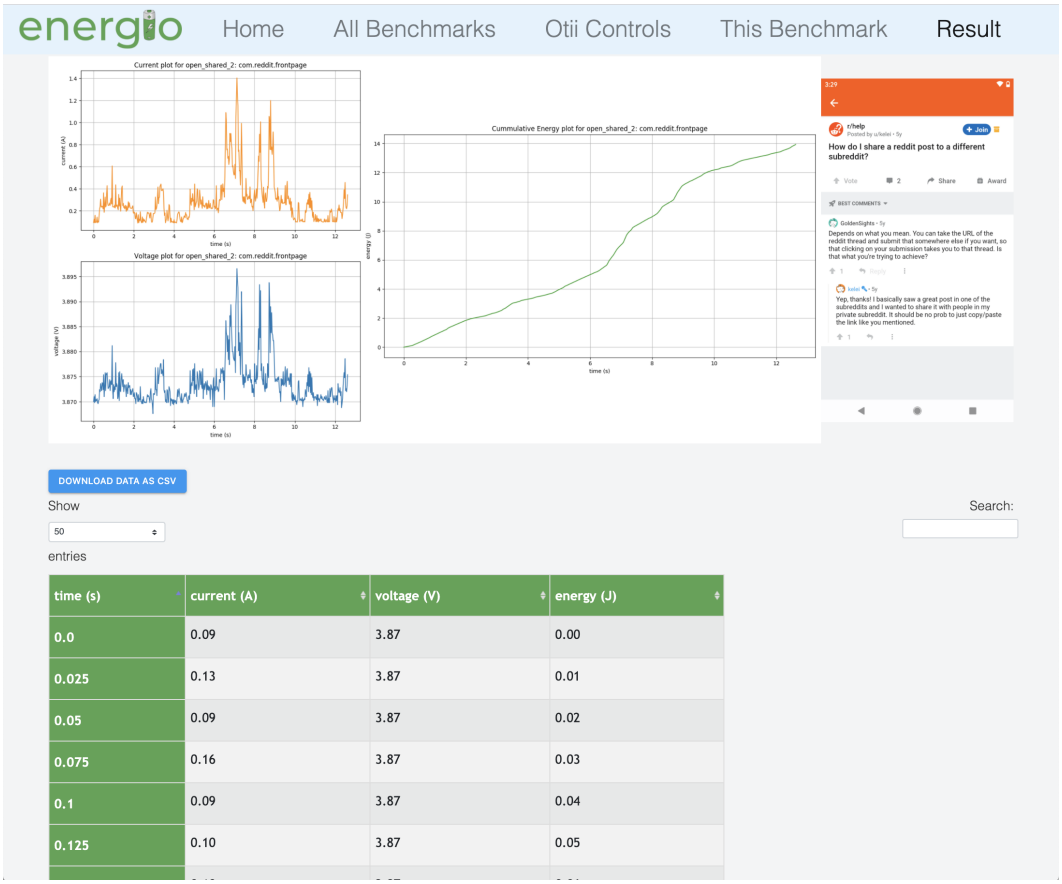


Figure 3.9: Energio Result Page

3.3.5 Storage Models

The uploaded zip file is stored on the /uploads folder of the web server. For each uploaded zip file there are also reference database entries created, which have information about the benchmark, its execution and the final results. The full schema of the tables in the database is shown in Figure

3.10 and explained below. The reason why Task and App table exist even though both could have their fields in the Result table is for future extensibility where further information about tasks or applications could be added.

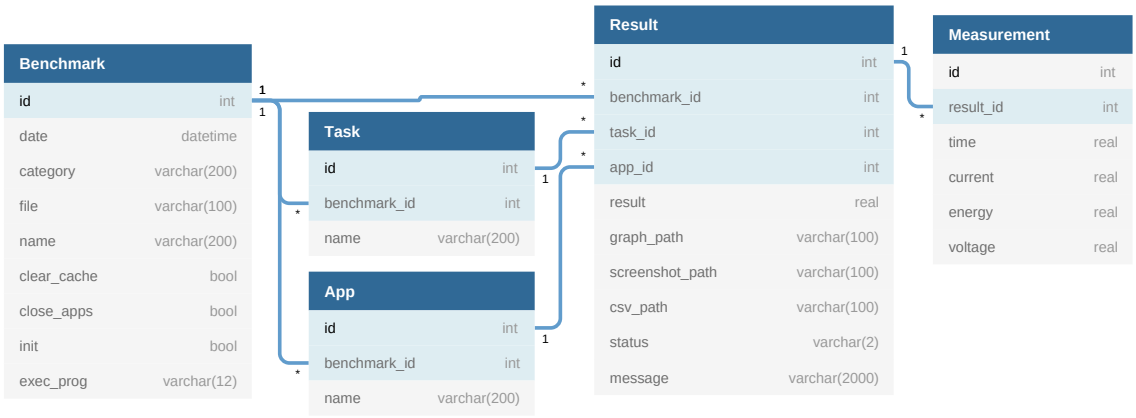


Figure 3.10: Energio Database Entity Relationship Diagram

Benchmark

When a user submits the files from the Index page, a Benchmark object is created with reference to the `file` path, `name` of the benchmark, `date` when created and options selected - `category` which is tested, whether cache should be cleared (`clear_cache`), whether apps should be closed (`close_apps`), whether initialisation scripts should be run `init` and what execution program should be run `exec_prog`. While `exec_prog` is stored as a varchar in the database, in Django's storage model it is an Enum with only the allowed programs - python2, python3, Java and monkeyrunner. This avoids potential attackers from running other programs on the host computer. The Enum also maps the execution program to more details about the program and the framework that should be run using it:

- **monkeyrunner** ⇒ monkeyrunner
- **java** ⇒ Java 1.8.0_241-b07 (UI Automator)
- **python** ⇒ python 2.7.16 (androidViewClient)
- **python3** ⇒ python 3.7.4

Task

For each folder in the uploaded zip file a Task object is created with the `name` of the folder and a `benchmark_id` foreign key to the benchmark.

App

For each unique file `name` across all folders an App object is created with the file and a `benchmark_id` foreign key to the benchmark.

Result

For each file there is a Result object created that stores the final output of the benchmark - `result` (average power used), `graph_path`, `screenshot_path`, `csv_path`, the `status` of running the application, and `message`, which stores the output and errors from script execution. It also has a foreign key `benchmark_id` to the benchmark, `task_id` to task and `app_id` to app to be able to identify which application and task the result is for. Furhter to this `status` is an Enum that maps 2 character strings to the current state of the application.

- **QU** ⇒ QUEUED

- **ST** ⇒ STARTED
- **CC** ⇒ CLEARING CACHE
- **CA** ⇒ CLOSING APPS
- **RI** ⇒ RUNNING INIT SCRIPT
- **TS** ⇒ TAKING SCREENSHOT
- **SD** ⇒ SAVING DATA
- **FA** ⇒ FAILED
- **FI** ⇒ FINISHED

Measurement

For each sample of the energy data recorded by Otii a Measurement object is created that contains the `time` of the sample, `current`, `voltage` and `energy` measured.

3.3.6 Upload File Structure

For the uploaded zip file to be correctly executed on Energio it must have a predefined structure. The zip file must contain folders with Tasks that will be compared across devices. Each of the Task folders then needs to contain the UI automation scripts which are named after the application package name they are testing. For example a script testing Google Chrome would be named 'com.android.chrome.py'. If the option to use initialisation scripts is selected, the initialization scripts must be named with '_init' appended to the package name. For example a valid initialisation script for Google Chrome would be 'com.android.chrome_init.py'. Files that are used across benchmarks and should not be executed can be placed in the root of the zip file. For example, the following structure would work when initialisation option is selected:

```
Browser Benchmark.zip
├── common.py
├── Browsing
│   ├── com.android.chrome.py
│   ├── com.android.chrome_init.py
│   ├── com.brave.browser.py
│   ├── com.brave.browser_init.py
│   ├── org.mozilla.firefox.py
│   └── org.mozilla.firefox_init.py
└── Speedometer Benchmark
    ├── com.android.chrome.py
    ├── com.android.chrome_init.py
    ├── com.brave.browser.py
    ├── com.brave.browser_init.py
    ├── org.mozilla.firefox.py
    └── org.mozilla.firefox_init.py
```

This structure is enforced in order to simplify the upload user interface to a single upload, and to support an unlimited amount tests to run without clogging the user interface (the alternative would be to let the user create task objects one by one and uploading files for each one). Also, to be able to clear cache of the particular application that is tested the full package name is needed in the file name. The alternative would be to clear the cache of all applications running, but that adds about 2 seconds of time for each application installed on the smartphone, making benchmarks take longer with no impact on the tested application itself (since other applications are closed regardless).

3.3.7 Measurement Units

Otii Measures the total energy consumed over a time period in Joules (J). To compare the average energy consumed over a set period of time it is more convenient to use the unit of power Watts (W). The power can be calculated from energy using Equation 1.

$$Power\ (W) = \frac{Energy\ (J)}{Time\ (s)} \tag{1}$$

The charge of a battery is given in mAh, so the total discharge of a battery when an app is running could be given in mAh. However, as shown through Equation 2, charge to energy conversion depends on voltage, so with a changing voltage the energy drawn per mAh would be different.

$$Energy\ (J) = 3.6 \times Voltage\ (V) \times Charge\ (mAh) \tag{2}$$

Although milliampere hour (mAh) are widely used as a unit of charge for batteries and Greenspector use it in their benchmarks, it is much better to use a unit of Power (Watts). Power in Watts is comparable between benchmarks ran at different voltages, at different timescales and even on different devices.

3.3.8 Fair Benchmarking

To create a reliable benchmark the effects of factors other than the one measured have to be minimised. This allows for a fair comparison between the tested tasks. The following are kept constant across the tests:

- Adaptive brightness is off and screen brightness of Google Pixel 2 is kept at 60%. The screen is one of the most energy draining elements, so brightness is kept at 60% so that impact of the screen is reduced, while also allowing visibility even in bright days (useful for testing the platform).
- Volume of Google Pixel 2 is at 4/7 for system, notifications, alarm, voice call and ring, and 12/25 for music. The sound settings are set to as close as possible to half volume to mimic typical use.
- After booting at least 3 minutes cool off time is given before benchmarks are run, as it was found that 2 minutes is enough for energy use to drop to a steady state.
- Closing applications - before each run, Energio user can select to close all of the applications on the system (on by default). This is done with a UI automation script.
- Clearing cache - before and after each run the cache of the application tested is cleared if the Energio user selects the option (on by default). This is done with an adb command.
- Wi-fi - Google Pixel 2 is connected to Wi-fi with 50 Mbps download and upload speeds. Wi-fi is chosen over 3G as the network is stable and so network performance won't affect the tests.
- GPS and Bluetooth are disabled, so that they don't impact the energy drain.

3.3.9 Energio Run Options

- Closing applications - before each run, Energio user can select to close all of the applications on the system (on by default). This is done with a UI automation script.
- Clearing cache - before and after each run the cache of the application tested is cleared if the option is selected by the user (on by default).
- Initialization Scripts - before each run initialization scripts can be run to bring the application to a known starting state. Energio distinguishes initialisation scripts by the '_init' suffix in the file name. For example a 'com.android.chrome.py' script could have an initialisation script called 'com.android.chome_init.py'

- UI automation execution program - When uploading the benchmark zip file, the UI automation script execution program can be selected from a dropdown. The selected program is what is used in the backend to run the UI autoamation scripts uploaded. Energio currently support running monkeyrunner, Java (UI automator), Python 2 (AndroidViewClient) and Python3 (AndroidViewClient). Extending to allow other programs only requires extending the exec_prog Enum in the Benchmark table and installing the required program on the computer.

3.3.10 Identifying Failures

People using the Energio platform do not have access to the devices (Smartphone and Otii) or the logs of the server and RQ queue execution. To make identifying what went wrong easier, an exception handler is attached to the RQ queue which takes a screenshot and saves the current output and exception in the message field of the Result object. Further to this, any output from the UI automation scripts (including exceptions) is captured and printed in the message for each app task run. Finally, even if no exceptions are thrown and the task completes, any print statements are captured and a screenshot is taken to show the final state of the UI automation script execution, which allows the users to see if their UI automation script reached the desired state on the smartphone.

3.3.11 Integration Testing

Firstly, to test whether the DNS is set up correctly, along with NGINX and uWSGI to communicate with Django it is enough to test whether http://energio.co.uk/ returns the correct response and the page is loaded correctly in the browser.

To then test the execution of benchmarks two zip files are uploaded separately:

1. **Simple execution scripts** - These 3 UI automation scripts each only open an application and wait for 3 seconds. If the running of these scripts succeed, then the following all work: the adb connection to the phone, closing applications, clearing cache, the TCP connection to Otii, screenshot, graph creation, CSV file creation and database object creation. If these scripts work, the Redis Server and django-rq also work in processing jobs asynchronously. To see if the data collection and database saves and queries work correctly the results of these simple benchmarks can be viewed. If any of them don't work, ways of identifying failures in Section 3.3.10 provide ways of finding which component failed.

2. **Failure Scripts** - ZIP files that intentionally cause errors test whether the system is able to catch error and handle them correctly. This includes: wrong folder structure, wrong file extensions and errors in the UI automation scripts. When an error occurs all the error messages should be displayed on the website logs and a failure screenshot taken.

3.4 Battery dataset analysis - Greenhub

The second part of this project involved taking an existing Greenhub dataset and analysing it to see the average power used by applications estimated using software measurements. This approach can be compared to the Energio benchmark platform to see if results obtained are valid and the ease of comparing applications for their battery usage.

The tables that Greenhub provides are described in Section 2.5.1. The REST API that Greenhub provides does not give access to the `app_processes` table that is needed for the purposes of this project. They do however, give access to all data through the Greenhub Farmer. The downloaded data is in the format of CSV files and unpacked has the size of over 140GB. This makes them too big to load to RAM and use applications that open CSV files directly.

3.4.1 Setting up a database

The CSV files can however be analysed by setting up a database and making queries to it. MariaDB was chosen over MySQL, since it is built to load data faster and is more efficient with bigger datasets. Initially the database was set up on an external 256GB SSD drive and the data uploaded using the fastest possible upload SQL query, shown in Listing 3.1.

```
LOCAL INFILE 'filename.csv' IGNORE INTO TABLE table_name FIELDS TERMINATED BY ';'
    LINES TERMINATED BY '\n'
```

Listing 3.1: Greenhub SQL upload query

It took over 12 hours to upload the data and the database took up nearly 210GB. While the database worked, even filter queries with limits took minutes to run. Once the final query was finalised, indexes on all of the filtered and joined columns had to be built or else establishing a rating from the dataset would take too long to run. Building indexes proved to be impossible with only 45GB remaining on the external drive.

This is when the database had to be rebuilt on Azure. Uploading the data to Azure from the computer was slow, so instead a virtual machine was set up to which the dataset was downloaded. Upload speeds from the virtual machine to the cloud MariaDB instance was only a bit slower than local, with the full upload taking 14 hours. Building the necessary indexes took another 6 hours, and in total takes 286.7GB.

Querying data for application energy consumption

From the fields that exist in Greenhub, the energy consumed by applications can be assessed in two ways. One, is to consider the discharge rate of the battery over the period the application was running and the other is to consider the average power drawn over that period. From this point, the first method will be referred to as 'battery discharge method' and the second method as 'voltage current method'.

The `battery_details` table contains a `battery_level` field which records the battery percentage at a particular point in time. The percentage drop over time can be calculated and if the capacity of the battery is known, it can be multiplied by the percentage drop to get the discharge of the battery in mAh. Further to this, when knowing the voltage, the discharge of the battery can be used to calculate the average power drawn over time as shown in Equation (3) below.

$$\begin{aligned} Power\ (W) &= Voltage\ (V) \times Current\ (I) \\ &= Voltage\ (V) \times \frac{Charge\ (C)}{time\ (s)} \\ &= Voltage\ (V) \times \frac{3.6 \times Charge\ (mAh)}{time\ (s)} \end{aligned} \tag{3}$$

The other approach is to use the `current_now` and `voltage` fields in the `battery_details` table to calculate the power drawn over time, by using Equation 1 that relates power to voltage and current. The average power over a time period can be calculated as the average current multiplied by the average voltage.

The first approach has a major disadvantage in that a battery level only has 100 possible values, so the precision of such discharge calculation is very low. The `capacity` field in the `battery_details` table is unfortunately empty, so solely using the dataset the average discharge can only be given as a percentage drop.

The second approach is the most accurate and precise way of calculating the average power of a device. However, the problem with the second approach is that in the dataset not all samples of devices have the `current_now` field filled. This means that while this approach is more accurate, it cannot cover the whole dataset and only certain devices can have the power calculated this way.

Querying Power intuition

As both methods have their drawbacks, both will be considered. To calculate power from queries from Greenhub, 4 tables are of interest - `devices`, `samples`, `app_processes` and `battery_details`. All the fields that need to be queried for are shown in an entity diagram in Figure 3.11.

Diagram in Figure 3.12 serves as an illustration of how on a certain device, application processes are running over time in the foreground. Greenhub collects its samples at almost regular intervals. Table 3.2 shows some fake data for samples when Process A is running to illustrate how the power queries can be built for Greenhub. For each device, samples, battery level, voltage and current can be queried when a specific application (`name='<package name>'`) is in the foregorund (`importance='Foreground App'`) and the phone is discharging (`battery_level='discharging'`).

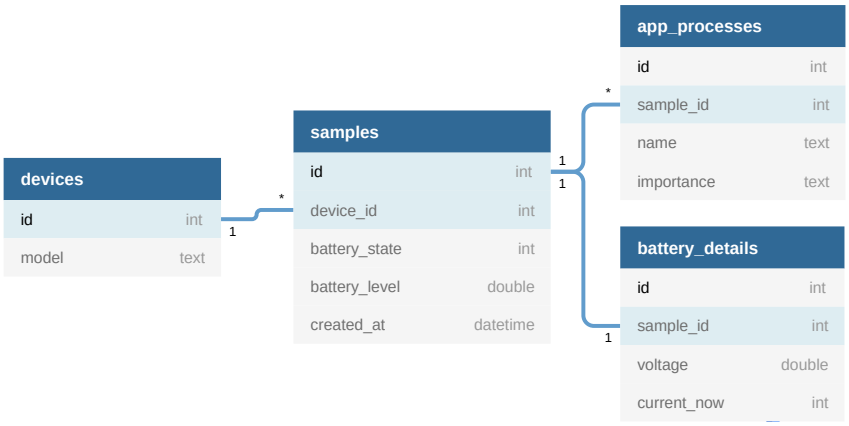


Figure 3.11: Greenhub Query Database Entity Relationship Diagram

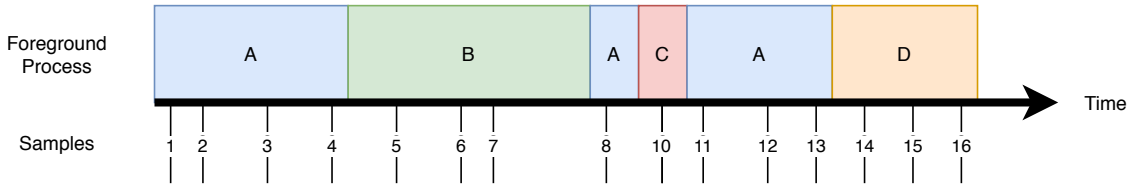


Figure 3.12: Greenhub Query Diagram

Then continuous samples need to be selected - in this example Samples 1-4, 8 and 11-13 would be the periods for which the average power should be calculated. Table 3.4 shows the battery discharge method (with the assumption that a battery is 4000mAh), while Table 3.3 shows the voltage current method of calculating average power.

Sample	1	2	3	4	8	11	12	13
Time (s)	0	2	7	10	27	34	38	41
Battery Level (%)	0.99	0.99	0.98	0.99	0.97	0.96	0.95	0.95
Voltage (V)	3.86	3.87	3.82	3.81	3.9	3.81	3.86	0.95
Current (mA)	760	821	800	1000	810	1200	700	0.95

Table 3.2: Greenhub: Illustrational Fake Samples Data

	Samples 1-4	**Sample 8**	**Samples 11-13**
Average Voltage (V)	3.850	3.990	2.873
Average Current (mA)	793	1000	633
Average Power (W)	3.056	3.9	1.821

Table 3.3: Greenhub: Illustrational Power Calculation for Voltage Current Method

	Samples 1-4	**Samples 11-13**
Discharge (%)	0.01	0.02
Average Discharge (%/s)	0.0014	0.0014
Average Discharge (mAh)	5.714	5.714
Average Power (W)	2.939	1.470

Table 3.4: Greenhub: Illustrational Power Calculation for Battery Discharge Method

SQL Query to Get Power

The Greenhub dataset can be used to create a similar benchmarking table as Energio creates - a table showing the average percentage discharge per second of each device (device corresponding to a task in Energio) for each benchmarked app. The full SQL query that achieves that is shown in Listing A.1. It is composed of smaller queries that perform the following:

1. The `apps` table contains all of the samples where the all of the applications benchmarked are in the foreground and the phone is discharging.This done by mergin the `app_processes` table with the `samples` and `devices` tables and filtering on the `name` of the application running, `importance='Foreground app'`) and `battery_state='Discharging'`.

```
with apps as (
    SELECT samples.id as id, importance, battery_details.voltage as voltage,
    current_now, devices.capacity as capacity, name, samples.device_id as
    device_id, battery_level, app_processes.created_at as d, model
    FROM app_processes JOIN samples ON app_processes.sample_id = samples.id
    JOIN devices ON samples.device_id = devices.id JOIN battery_details ON
    battery_details.sample_id = samples.id
    WHERE name in ('com.microsoft.emmx', 'com.brave.browser', 'com.android.
    chrome', 'com.opera.browser', 'org.mozilla.firefox') and battery_state='
    Discharging'
),
```

Listing 3.2: Greenhub samples SQL Query

2. `t_start` and `t_end` tables are created that contain the beginning and end timestamps of periods with consecutive samples respectively. `t_start` is created by going through the apps table and for each device calculating the time difference between the current sample and the previous sample. Samples where the time difference is high signal the start of a sequence of samples. `t_end` is created by going through the `apps` table and for each device calculating the time difference between the current sample and the next sample. Samples where the time difference is low signal the end of a sequence of samples.

```
t_start as (
    select *
from (
    SELECT T1.id,
        T1.device_id as device_id,
        T1.battery_level as battery_level,
        T1.d as d,
        IFNULL(TIME_TO_SEC(TIMEDIFF(T1.d, Max(T2.d))),5000) AS TimeDiff
    FROM apps T1
        LEFT JOIN apps T2
        ON T1.device_id = T2.device_id
            AND T2.d < T1.d
    GROUP BY T1.device_id, T1.d
    ) as x
where x.TimeDiff > 40
),
t_end as (
    select *
from (SELECT T1.id,
        T1.device_id as device_id,
        T1.battery_level as battery_level,
        T1.d as d,
        IFNULL(TIME_TO_SEC(TIMEDIFF(T1.d, MIN(T2.d))),-5000) AS TimeDiff
    FROM apps T1
        LEFT JOIN apps T2
        ON T1.device_id = T2.device_id
            AND T2.d > T1.d
    GROUP BY T1.device_id, T1.d) as x
where x.TimeDiff < -40
),
```

Listing 3.3: Greenhub start/end SQL Query

3. The `t_boundaries` table contains the start and end dates of consecutive samples for each device. There can be more than one of these boundaries per device. This table is created by joining the `t_start` and `t_end` tables on device_id and for each `t_start` date fidning the earliest date from `t_end`.

```
t_boundaries as (
    SELECT
    T1.device_id as device_id,
    T1.d as t1,
    MIN(T2.d) AS t2
FROM t_start T1
    LEFT JOIN t_end T2
    ON T1.device_id = T2.device_id
        AND T2.d >= T1.d
GROUP BY T1.device_id, T1.d
),
```

Listing 3.4: Greenhub boundaries SQL Query

4. The final select query takes the average battery percentage drop in between the boundary dates for each device and application. This is done by joining the apps table with the boundaries table on `device_id`, and for samples between the boundaries taking the minimum and maximum `battery_level`. Then the wrapping select takes the average discharge by taking the difference between the two levels of the time difference between the two boundary samples.

```
select
    model,
    name,
    importance,
    sum(count) as num_samples,
    sum(total_discharge) / sum(TimeDiff) as avg_discharge,
    sum(total_discharge) / sum(TimeDiff) * capacity * 3.6 / sum(TimeDiff) *
    AVG(voltage) as avg_power,
    AVG(voltage) * AVG(current_now) / -1000 as avg_power2

    FROM (
        select t1.device_id as device_id,
            count(*) as count,
            model,
            capacity,
            name,
            voltage,
            current_now,
            importance,
            (MAX(battery_level) - min(battery_level)) as total_discharge,
            MAX(battery_level) as max,
            min(battery_level) as min,
            GROUP_CONCAT(battery_level),
            TIME_TO_SEC(TIMEDIFF(t_boundaries.t2,t_boundaries.t1)) AS TimeDiff
        FROM apps t1
            INNER JOIN t_boundaries on t1.d BETWEEN t_boundaries.t1 and
    t_boundaries.t2
        where t1.device_id = t_boundaries.device_id
        and current_now < 0
        group by t_boundaries.t1
        having TimeDiff<>0
    ) as y
where y.total_discharge > 0
group by y.model, y.name, y.importance;
```

Listing 3.5: Greenhub average discharge SQL Query

5. Finally, optionally the resultant table can be pivoted to create a table like Energio creates (where the discharge for each device is in rows and applications in columns). Maria db does not have a dedicated pivot query and instead a dynamic query has to built using SQL prepared statements. This requires saving the previous full query in steps 1-4 in a temporary

table (it is called `results` in the code below) and then performing the dynamic pivot query. The full query in Listing A.1 shows how this would be done.

```
SET @sql = NULL;
SELECT
  GROUP_CONCAT(DISTINCT
     CONCAT(
       'MAX(IF(name = ''',
   name,
   ''', avg_discharge, NULL)) AS ',
   CONCAT("'",name,"'")
 )
   ) INTO @sql
FROM result;

SET @sql = CONCAT('SELECT model, ', @sql, ' FROM result GROUP BY model');

PREPARE stmt FROM @sql;
EXECUTE stmt;
```

Listing 3.6: Greenhub Pivot SQL queries

Each of the sub queries has been manually tested in isolation to ensure that the full query works as expected.

3.4.2 Battery capacity of devices

Although Greenhub has the capacity field present in the devices table, it is empty for all devices. This means that with the query above only the average discharge as percentage of the battery can be calculated. Therefore energy measurements cannot be compared or averaged out across all devices, making the only way (using the discharge method) to compare browsers be by comparing the battery drop for the same device. With so many device models available, even with such a large dataset not all devices have battery measurements corresponding for them. This leads to the data in Table 3.5:

	Chrome	Brave	Firefox	Opera	Edge
GT-I9195	0.007862595		0.013431372		0.026538461
SM-G903F	0.006911764		0.002352941		
SM-G920F	0.008483606		0.008461538		
SM-G925F	0.014000000	0.007105262			
SM-G928F	0.007924527				
SM-G930F	0.004166666				
SM-G935F	0.010697674				
SM-G950F	0.011860464		0.019999999	0.000322581	
SM-G955F	0.009999999		0.004999999		
SM-J100H	0.009999999			0.003529411	

Table 3.5: Greenhub: Battery Discharge Rate for Browsers in the Foreground

The solution to this problem is to fetch all the battery capacities from the web and insert them into the database. There are 10662 unique device models in the database, so a manual search and insert is infeasible.

There exists an Open Device Definition Repository (openDDR) [11] that provides a list of devices on the market and basic information about them like what OS they are running, what is the display resolution or whether they supports Ajax. Unfortunately it does not contain battery capacity information.

The only sources that have battery capacity information are phone specification websites, and data from these websites can be scraped. The most popular website with phone specifications is https://www.gsmarena.com/. However, GSMArena does not offer an open API and only exposes device battery statistics in a table of their battery life test. These contain a rating on how long

[11] https://github.com/OpenDDRmobi/openddr-data

the battery should last and not on the capacity of the battery. The only way to fetch the capacity is to scrape the data directly from the website.

There are 3 community built APIs:

- **gsm**[12] - capable of hosting a local web server with a REST API that can get all brands, the devices of each brand and the phone specifications.
- **gsmarena-scraper-json**[13] - a Node.js server that offers REST API to get all brands, devices of each brand, phone specifications, reviews and has a search tool.
- **gsmarena-API**[14] - a PHP library that can get brands, search for devices and get their details.

All of the APIs were tried and while they offer the ability to fetch details of a device, the user has to know the exact link to the page (that can be found through getting devices per brand).

The Greenhub dataset only contains the device model and manufacturer. Although two of the APIs offer search, the search functionality does not work with device models. This is in parallel with the direct search on the GSMArena website which is also not able to find all devices solely based on their model names. GSMArena does have a field for models, but that is part of the specifications of the phone. It would therefore be possible to fetch all of the devices using the APIs, storing them and then searching through them to find a matching details page with the model name.

3.4.3 Battery capacity web crawler

As it turns out when Google is provided with a model name and 'GSMarena' in the search phrase is able to match the device name to the model and show the correct specifications page. This means that a web crawling script can be built that searches the phone model on google and then scrapes the battery information directly from GSMArena. This makes it a better approach then using an API where all of the possible devices have to be fetched first.

Such a web crawler performs the following tasks:

1. Perform a SQL query `SELECT DISTINCT model FROM devices` to get all distinct model names from the Greenhub database.

And then for each device model:

2. Get the HTML of the google search with 'GSMArena specification' prepended before the device model name.
3. Press on the first link that is from GSMArena and fetch the HTML of the page.
4. Extract the battery capacity value from the specifications table and store it in a dictionary with the model as key and battery capacity as value.
5. Construct a Pandas dataframe form the dictionary and save the result as CSV on disk.
6. Print information about failures, successes and store information in logs.

This works well for the first 50 consecutive requests and making more requests makes GSMArena block the host making these requests for 10 hours. One way to avoid this, is to have the web scrapping script connect to the TOR network and change the IP address every time it is blocked by GSMArena's web server.

While using TOR solves GSMArena access, it makes Google search inaccessible. Google is inaccessible from most TOR devices, requiring a CAPTCHA to be filled, because a lot of people around the world make requests to Google from these TOR IP addresses. In the end a mixture of the 2 solutions is used - where Google search is accessed through (step 3) the local computer's IP and GSMArena (step 4) through the TOR network.

The final web crawler architecture is visualised in Figure 3.13.

[12] https://github.com/xpresservers/gsm
[13] https://github.com/karnadii/gsmarena-scraper-json
[14] https://github.com/ramtin2025/gsmarena-API

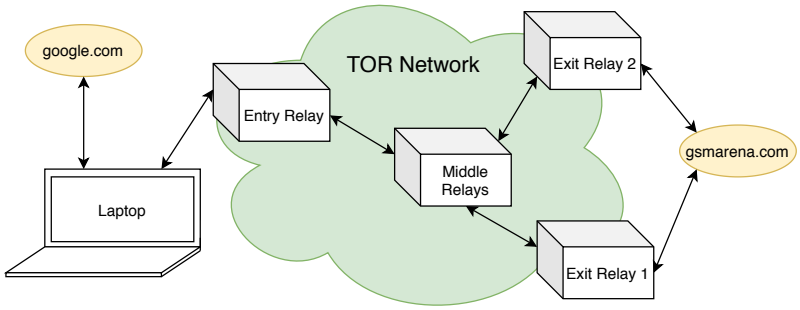


Figure 3.13: Diagram of TOR Web Crawler

Failure Exit prevention and logs

The script detects if it gets blocked by GSM Arena by checking whether the status code of the response matches '429 Too Many Requests'. It proceeds to get a new exit relay IP address when this happens. If a request is other 429 the script goes to search for the capacity of the next model, and if it is other than 200, it also prints what the status code was (for example some pages return 404 errors). The script also proceeds to the next model if the battery capacity does not exist on GSM Arena's page or if any other exception occurs when extracting a link (invalid link from Google) or no GSM Arena links were found. Whenever any issue resulting in a skipped model occurs, the reason for abortion is logged and the failed model is added to the list of failures. Failed models at the end of the script can be searched for manually on manufacturer websites directly.

Web Crawler Execution

To try to find the capacity values for the 10662 unique devices, the custom web scraping script ran for over 7 hours and the exit relay of the TOR network had to be changed 205 times. The script successfully scraped the capacity data for 7163 devices. The rest 3499 models could not be scrapped due to a GSM page not existing for these devices or no battery data present for them.

To update the database the CSV file created by the scraping script was iterated over and an update statement added to the database.

3.4.4 Summary

In Summary, creating a benchmarking energy platform required modifying hardware and setting up connections to the power monitor and a smartphone. To enable easy access to the platform it required a website to be built, which enables users to queue benchmarking UI automation jobs for which the energy consumption is measured. It allows people to interact remotely with the system without needing to interact with the hardware directly. The platform back end is built using Django and an NGINX web server serves the website to users.

The second part of the project involves looking at an existing battery dataset and creating queries to extract the average power used for applications. Two approaches were considered and implemented in this section - a power calculated through battery discharge approach and a power calculated through average voltage and current approach. The battery discharge approach also required setting up a web crawler that uses the TOR network to fetch data that was missing in the dataset.

Chapter 4

Evaluation

This section evaluates the Energio platform by writing UI automation scripts that create benchmarks and compare applications of the same category. This is done for Browsers and Reddit categories, using Monkeyrunner and AndroidViewClient as UI automation tools. The Greenhub dataset method is also evaluated, by using the SQL query built in section 3.4.1 and comparing the two described approaches for power extraction (battery discharge method and voltage, current method. In addition, the results of Energio are compared with Greenspector's and Aeon's results.

4.1 Energio Results

To create a Benchmark using the Energio platform it is enough to create UI automation scripts for each task that perform the accomplish the same goal across all applications tested. In this section, two UI automation methods are used - Monkeyrunner and AndroidViewClient.

4.1.1 Building UI automation scripts

Monkeyrunner

Monkeyrunner scripts interact directly with ADB and the biggest drawback of such approach is that touches require pixel coordinates. This means that to write scripts that would work on the Google Pixel 2, a device or emulator with the same screen dimension is needed to test and get the coordinates for touch events.

Getting Touch Coordinates

For Monkeyrunner any presses have to be at a specific pixel location on the screen. To find a location the command 4.1 was built that shows the integer pixel location of the location where the phone is tapped.

```
1 adb shell getevent -l | grep ABS_MT_POSITION --line-buffered | awk '{a = substr($0
      ,54,8); sub(/^0+/, "", a); b = sprintf("0x%s",a); printf("%d\n",strtonum(b))}'
```

Listing 4.1: ADB Get Tap Location Command

The command works by getting a list of events, then using grep to search for lines where 'ABS_MT_POSITION' is present (gets the line with touch events in hex) and finally using `awk` to get the relevant hex values, strip them of zeros and convert hex to decimal that Monkeyrunner uses. This continuously prints the x and y coordinates in the terminal only the device is tapped.

Getting current activity

All UI automation scripts require to know which application and activity within that application to launch. To find what the application package name and activity that needs to be launched is called, the application can be opened on any Android device and the following command executed on terminal [1]:

[1] https://stackoverflow.com/questions/13193592/adb-android-getting-the-name-of-the-current-activity

```
adb shell "dumpsys activity activities | grep mResumedActivity"
```

Listing 4.2: ADB Get Activity Name

AndroidViewClient

As explained in the background section 2.6.5, AndroidViewClient uses UIAutomator BySelectors which target views directly. Scripts written using this framework can target views based on ids or text that they can contain. This makes scripts made using this framework reusable across devices, meaning that a developer can write these UI automation scripts without access to a Google Pixel 2 device.

To get the IDs or text of the screen AndroidViewClient provides a `dump` script that shows the view layout with the corresponding ids and text. This allows for knowing how to target specific views.

For some UI automation tasks it is convenient to wait for a view to appear and assert that it shows up before continuing to the next task. To do this a few helper functions were built that help with this task.

The function waits for the ID for up to a time defined by `timeout`. To update the state that AndroidViewClient dump is performed in a while loop every `refresh_time` seconds and checked for the view that is being searched. Sometimes AndoirdViewClient dump hangs during screen transitions, so to avoid this problem an OS signal alarm is attached to timeout after 4 seconds and the exception caught to retry is again. The pseudo code for a function that wait for an id is shown in Listing 4.3 below:

```
def wait_for_id(vc, id, timeout = 10, refresh_time = 0.5):
    while execution time < timeout:
        try:
            register system signal alarm for 4 seconds
            vc.dump
            unregister system signal alarm
            if vc.findViewById(id) is None:
                sleep for refresh_time
            else:
                return
        except (KeyboardInterrupt, SystemExit):
            raise
        except:
            print exception details
                sleep for refresh_time
    raise RuntimeError('Failed waiting '+ timeout +'seconds for id: ' + id)
```

Listing 4.3: Python Wait For View Pseudocode

The same function that waits for text was also created, and additional convenience functions where the waited for event is touched as soon as it is found.

4.1.2 Browsers

Browsers are one of the most comparable applications as they all serve the same purpose - browsing the internet. Four browsers which came up first under Google Play search and the default Chrome browser installed on Google Pixel 2 are tested. The browsers tested are: Google Chrome, Edge, Firefox, Brave and Opera, and the version and package names tested can be found in Section A.1. To compare these applications 10 tasks are performed on each browser, all of which explained in further detail in section 4.1.2 and 4.8.

Initialisation Scripts

To get a consistent and fair result, the cache reset option of Energio is used along with the initialisation scripts option. This allows placing each application in a known and clear state before running each benchmark. On each launch after the cache each application has set up pages that need to be navigated for to reach the browsing page. All of the sync features and personalised adds of the browsers are turned off. Only Brave's shields are on as they are a core feature of the browser. The UI initialisation scripts click through each browser and set it up as follows:

- **Brave** - Google is selected as the default search engine, brave shields are enabled (as they are a core feature of brave), and brave rewards are disabled.
- **Google Chrome** - usage statistic are turned off, and the browser faster feature too. Sync is also turned off.
- **Edge** - sync is off, sharing data about website you visited is off and sharing usage data for personalisation is also off. Setting up Edge as the default browsing application is also skipped.
- **Firefox** - Sign in is skipped, sync is turned off, and send tab to desktop feature is also off.
- **Opera** - personalised news and ads are off.

Browsing

To test the browsers energy usage on navigation, a browsing test that goes through the most popular websites on mobile devices was conducted. The top 100 most popular website visited on mobile devices was taken from similarweb.com [2] on 10.05.2020 and can be seen in Appendix A.4.

For each task the cache clear option was selected and the initialisation script for each browser navigated through the initial setup pages to get to the first open tab. The browsing test was performed using Monkeyrunner by going through the list of top websites to visit and performing the following steps:

1. Click on the search bar of the browser
2. Type the address of the website
3. Press search button on keyboard

For each application 3 tasks were performed using this method - visiting top 5 websites, visiting top 10 and visiting top 100 websites. The results from the benchmark can be seen in Table 4.1 and in Figure 4.1, where the dotted lines represent the average power measured for each application.

Browser	**Top 5 Power (W)**	**Top 10 Power (W)**	**Top 100 Power (W)**
Brave	1.450	1.546	1.692
Edge	1.606	1.692	1.957
Firefox	1.563	1.752	2.270
Google Chrome	1.463	1.601	1.942
Opera	1.415	1.459	1.665

Table 4.1: Browsing Top Websites Results

The Top 100 websites visited test is the most representative of the 3. From this test it can be seen that Opera performs the best, followed by Brave, Google Chrome, Edge and finally Firefox.

Benchmarks

Another way to test browsers to compare the energy usage when running browser specific benchmarks. 7 browser benchmarks were selected for this purpose - 3 JavaScript benchmarks, 2 Graphics benchmarks and 2 performance benchmarks.

To test the performance of the browsers only for the browser benchmarks, the initialisation scripts go through through the first setup of each browser as in section 4.1.2 and navigate to the benchmark web page. For MotionMark the screen is also rotated as the benchmark requires landscape orientation for the run. The UI automation script for which the energy is measured only presses the start button of each benchmark and waits until a specific id or text appears that signals the end of the browser benchmark. To verify that the benchmark has executed and to get the score of the browsing benchmark, the final screenshot can be manually checked for.

Having the score of the benchmark allows for not only an average power analysis, but also a two dimensional analysis - power per point scored in a browsing benchmark. This allows to compare

[2] https://www.similarweb.com/corp/blog/mobile-web-top-websites/

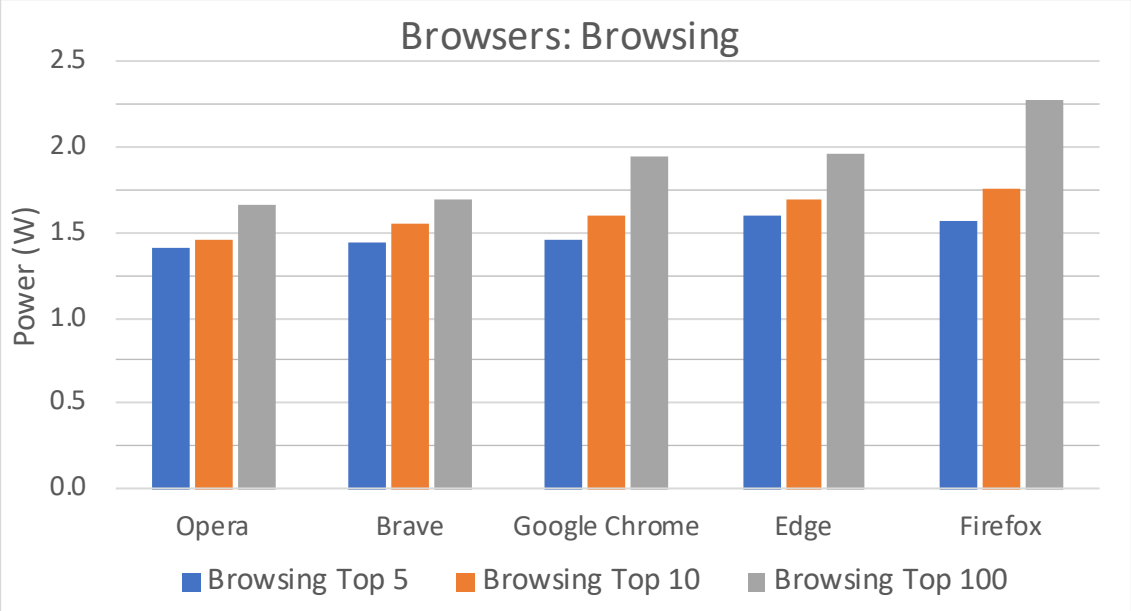


Figure 4.1: Energio Browsers Results: Browsing

browsers more fairly, as some may consume less energy and have lower power drain, but suffer from poorer performance. Poorer performance in turns means that an application needs to run for longer to accomplish the same task and use more energy than it may seem initially seem.

The power used per browsing benchmark point scored is what will be referred to as 'Efficiency Score' and will allow for application comparison across the same benchmark. To make this measure comparable across all browsing benchmarks it has to be further normalised. This is done in relation to Google Chrome as it is the default browser on the Google Pixel 2. To normalise the Efficiency Score, it is divided by the efficiency score of Google Chrome in the same benchmark. This works well for all benchmarks except for Kraken, because for most benchmarks the better the browser preforms the better the score. This means that a browser that performs better will have a lower Efficiency score. However for Kraken the lower the score the better the browser performed. To go around this, the normalised efficiency is calculated as for other benchmarks, but an additional mirroring step is performed where all scores are mirrored around 1 to account for how a lower score should be better.

JavaScript Benchmarks

3 JavaScript benchmarks were chosen to test the energy efficiency of JavaScript related tasks for each browser. The benchmarks tested for each browser are:

- **Octane 2** [40] - A JavaScript benchmark created by Google's V8 that runs 17 tests to measure JavaScript Performance. Octane was retired in April 2017 as the V8 team found that most JavaScript compilers had optimisations in place to achieve high scores in Octane, and any further optimisations increasing the Octane score actually negatively impacted performance in real-world scenarios.

- **Kraken (1.1)** [41] - A JavaScript benchmark created by Mozilla that runs 12 tests that measure the performance for ai, audio, imaging, json and cryptography.

- **JetStream2** [42] - A JavaScrtipt and Web Assembly benchmark created by Apple's Webkit that runs a series of tests programming techniques and different workloads.

The results for Octane, Kraken and JetStream2 averaged over 3 trials are in Tables 4.2, 4.3 and 4.4 respectively. More detailed results for each trial are in Appendix A.5.2. Figure 4.2 shows the average power measured for each applications and Figure 4.3 the Efficiency Score (power per benchmark point) normalised to Google Chrome. In both Figures the horizontal dotted lines represent the average of the benchmarks performed for that application.

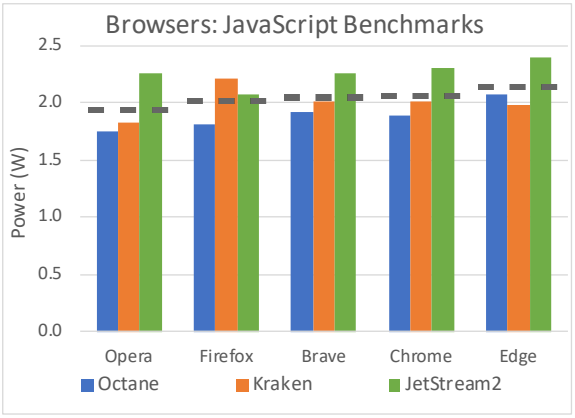


Figure 4.2: Energio Browsers Results: JavaScript Benchmarks

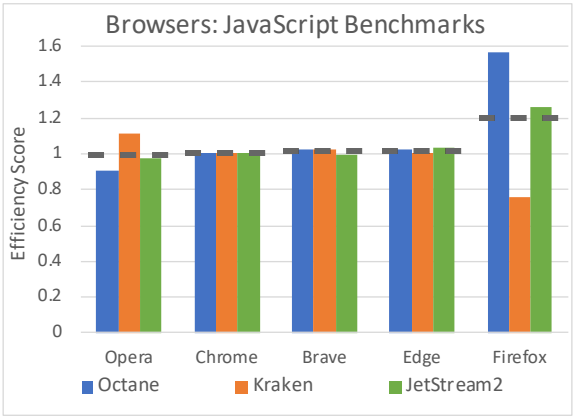


Figure 4.3: Energio Browsers Results: Efficiency Score JavaScript Benchmarks

Browser	Power (W)	Score	Efficiency	Normalised
Brave	1.924	10550	0.182	1.019
Chrome	1.881	10516	0.179	1.000
Edge	2.075	11334	0.183	1.023
Firefox	1.811	6459	0.280	1.567
Opera	1.753	10858	0.161	0.902

Table 4.2: Octane Benchmark Results

Browser	Power (W)	Score	Efficiency	Normalised	Normalised Mirrored
Brave	2.006	3809.2	0.527	0.976	1.024
Chrome	2.008	3718.7	0.540	1.000	1.000
Edge	1.975	3658.0	0.540	1.000	1.000
Firefox	2.205	3300.6	0.668	1.237	0.763
Opera	1.831	3824.0	0.479	0.887	1.113

Table 4.3: Kraken Benchmark Results

Browser	Power (W)	Score	Efficiency	Normalised
Brave	2.248	33.871	66.380	0.995
Chrome	2.304	34.526	66.732	1.000
Edge	2.393	34.771	68.822	1.031
Firefox	2.068	24.516	84.354	1.264
Opera	2.255	34.528	65.319	0.979

Table 4.4: JetStream2 Benchmark Results

As can be seen in the results taking only the power measured into consideration Firefox performs very well for JavaScript, but once benchmark scores are taken into consideration it can be seen that Firefox actually performs much worse in terms of energy for each benchmark point it gets. As expected, Firefox performs exceptionally well in Mozilla's Kraken benchmark. However it performs the worst in the other 2 benchmarks. This highlights the importance of using a wide range of tasks to test an application to catch applications that are optimized only to a specific benchmark.Chrome, Brave and Edge are all based on the same Chromium engine so a similar result in terms of energy and performance is to be expected.

For JavaScript Opera is the most energy efficient, followed by Chrome, Brave, Edge and finally Firefox.

Graphics Benchmarks

2 graphics benchmarks were chosen to test the energy efficiency of graphics related tasks for each browser. The benchmarks tested for each browser are:

- **Motionmark 1.1** [43] - A graphics benchmark created by Apple's Webkit that tests the graphics system of a browser on CSS, SVG and canvas drawing.

- **Wirple** [44] - This graphics benchmakrk tests HTML5 3D applications, by looking at the peformance of Canvas3D and WebGL.

The results for MotionMark and Wirple averaged over 3 trials are in Tables 4.5 and 4.6 respectively. More detailed results for each trial are in Appendix A.5.3. Figure 4.4 shows the average power measured for each applications and Figure 4.5 the Efficiency Score (power per benchmark point) normalised to Google Chrome. In both Figures the horizontal dotted lines represent the average of the benchmarks performed for that application.

Browser	Power (W)	Score	Efficiency	Normalised
Brave	2.248	33.87	66.380	0.995
Chrome	2.304	34.53	66.732	1.000
Edge	2.393	34.77	68.822	1.031
Firefox	2.068	24.52	84.354	1.264
Opera	2.255	34.53	65.319	0.979

Table 4.5: MotionMark Benchmark Results

Browser	Power (W)	Score	Efficiency	Normalised
Brave	2.253	956	2.358	1.109
Chrome	2.294	1079	2.125	1.000
Edge	2.177	967	2.252	1.059
Firefox	2.032	825	2.463	1.159
Opera	2.169	1055	2.056	0.967

Table 4.6: Wirple Benchmark Results

Just like in the JavaScript benchmarks the average power used for each benchmark would suggest that Firefox performs very well for graphics benchmarks in terms of energy used. While that is true, it does so at a performance hit. For Graphics, Firefox and Edge have performed significantly worse compared to the other browsers for the MotionMark benchmark in terms of the efficiency score. While Chrome, Brave and Edge are all based on the Chromium JavaScript engine, the energy results show that they are handling graphics differently internally.

Based on the average efficiency score across the two benchmarks, Chrome performs the best, followed by Opera, Brave, Edge and Firefox.

Performance Benchmarks

2 performance benchmarks were chosen to test the energy efficiency of performance related tasks for each browser. The benchmarks tested for each browser are:

- **Speedometer 2.0** [45] - A performance benchmark created by Apple's Webkit that tests uses simulates user interaction which test's the browser's responsiveness.

- **Basemark Web 3.0** [46] - This benchmark, created by Basemark tests performance on new web standards and features.

The results for Speedometer and Basemark averaged over 3 trials are in Tables 4.7 and 4.8 respectively. More detailed results for each trial are in Appendix A.5.4. Figure 4.6 shows the average power measured for each applications and Figure 4.7 the Efficiency Score (power per benchmark point) normalised to Google Chrome. In both Figures the horizontal dotted lines represent the average of the benchmarks performed for that application.

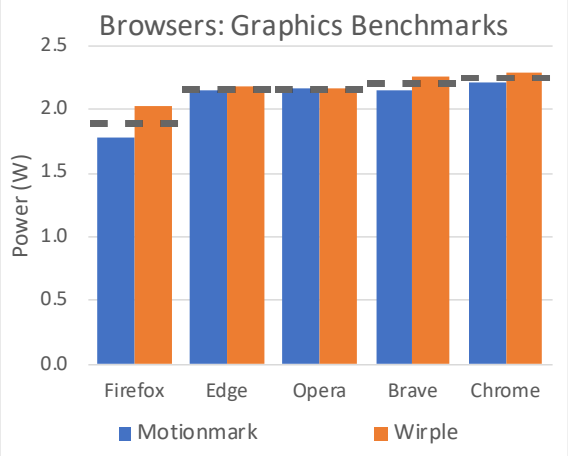


Figure 4.4: Energio Browsers Results: Graphics Benchmarks

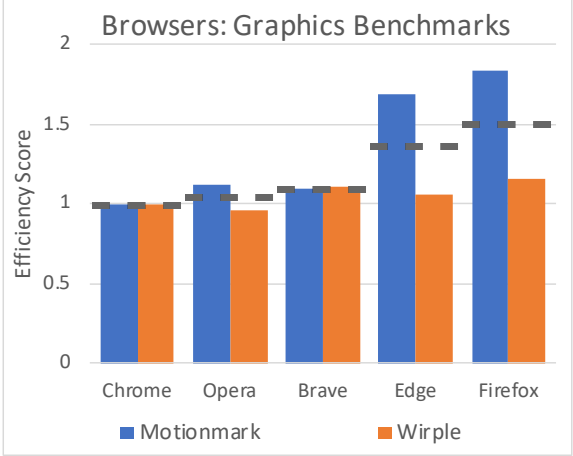


Figure 4.5: Energio Browsers Results: Efficiency Score Graphics Benchmarks

Browser	Power (W)	Score	Efficiency	normalised
Brave	2.055	20.7	99.419	0.868
Chrome	2.141	18.7	114.510	1.000
Edge	2.171	18.7	116.286	1.016
Firefox	1.859	18.4	100.832	0.881
Opera	2.079	21.5	96.832	0.846

Table 4.7: Speedometer Benchmark Results

Browser	Power (W)	Score	efficiency	normalised
Brave	3.042	230.79	13.180	1.050
Chrome	3.086	245.94	12.546	1.000
Edge	2.982	235.74	12.648	1.008
Firefox	2.965	181.81	16.308	1.300
Opera	3.073	252.60	12.170	0.970

Table 4.8: Basemark Benchmark Results

For the performance benchmark Chrome performs the best in terms of the efficiency score, followed by Opera, Brave, Edge and lastly Firefox. It is once again the case here that Firefox performs the best in terms of energy solely when based on the energy consumed during benchmark execution. However, when the benchmark performance scores are taken into account as well, Firefox clearly performs worse than the rest of the browsers. It can also be observed that Edge performs much worse on the MotionMark compared to the other 2 chromium browsers, with the performance comparable to Firefox.

All Benchmarks

The results for all of the benchmarks run together can be seen in Figure 4.8 and the combined efficiency scores in Figure 4.9. The dotted lines on both figures represent the averages across all benchmarks.

Just as for all individual benchmarks it can be seen that Firefox performs the best in terms of the energy used during the benchmarks, but does so at a significant performance hit. When considering the average efficiency score, Opera performs the best, followed by the 3 Chromium based browsers, Chrome, Brave and Edge, and finally Firefox performs the worst.

These results match quite well what the actual performance was when loading top 100 webpages meaning that testing the energy usage of browsers for specific tasks is representative of real loads, while providing more detailed analysis into what sort of workloads an application is energy ineffi-

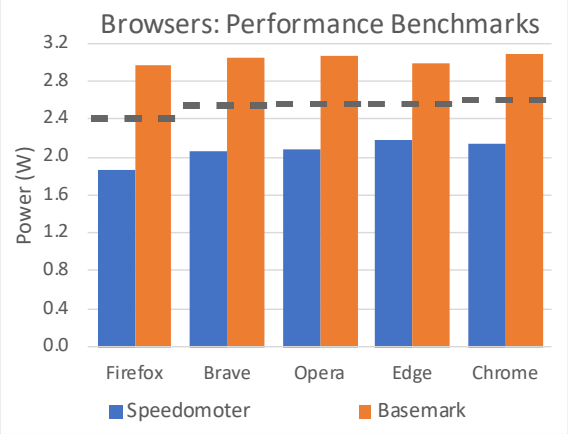


Figure 4.6: Energio Browsers Results: Performance Benchmarks

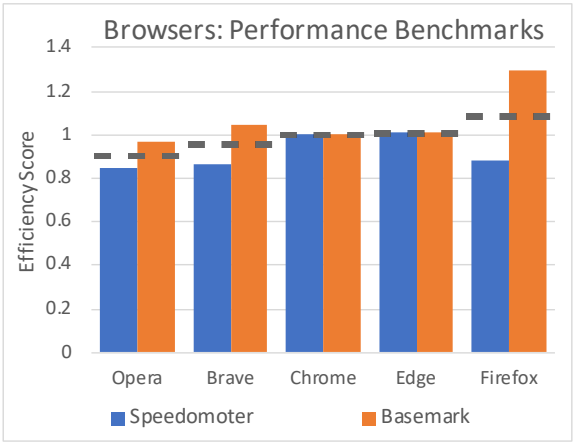


Figure 4.7: Energio Browsers Results: Efficiency Score Performance Benchmarks

cient on. A full energy analysis like this is especially useful to developers who can see that which parts of their browser drain the most battery and have the most room for improvement.

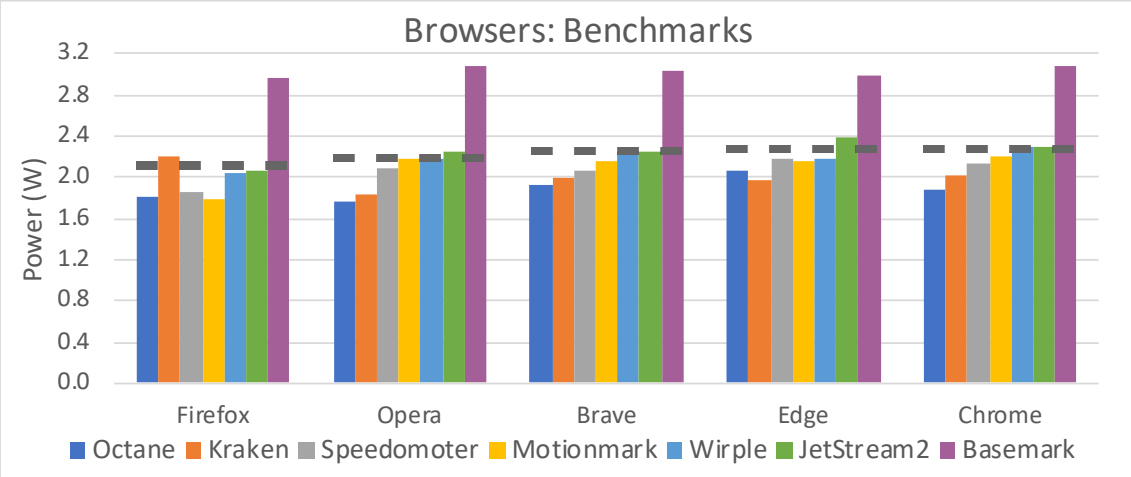


Figure 4.8: Energio Browsers Results: All Benchmarks

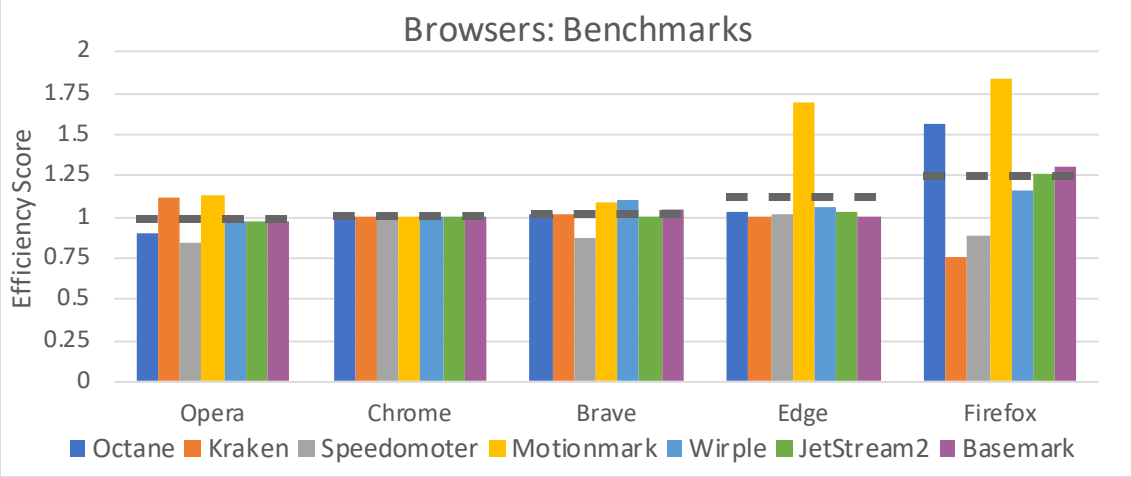


Figure 4.9: Energio Browsers Results: Efficiency Score for All Benchmarks

4.1.3 Reddit Applications

Reddit Applications serve as an another good category to benchmark applications against each other as all applications use the same API and are able to present the same data. 5 Reddit applications - BaconReader, Red Reader, Reddit Official and Relay and rif is fun - were chosen as previous work (Aeon) have tested these applications allowing Energio's results to be compared directly to past findings.

To benchmark Reddit Browsers to each other a public but restricted subreddit '\r\energio' was created. This allows for the creation of a Reddit page that can be static and will test all applications for the same content. Using any other page would risk that interaction from external users would change what is displayed and unfairly judging applications to each other. The '\r\energio' was populated with a variety of content representative of a typical Reddit page - text posts, shared posts, image uploads, GIFs, polls and shared external links.

The UI automation scripts were written using AndroidViewClient to be able to assert whether the script has reached the correct view. As the applications are differently organised, the initialisation scripts navigated through the application first start up and through the menus to be one click away from the energio page for the 'Scroll Energio' test and one click away from loading the tested task for the rest. 5 different tasks were chosen to be tested to see mimic typical reddit browsing experience:

1. **Scroll Energio** - opens the '\r\energio' subreddit and scrolls through it until it reaches the end of it.
2. **Open shared** - opens a shared post that links to a different subreddit.
3. **Open post** - opens a simple reddit post with text and comments.
4. **Open PNG** - opens an image that is part of an uploaded post.
5. **Open GIF** - opens a GIF image that is shared on the platform.

The results with average power across 3 trials can be seen in table 4.9 and more detailed results for individual trials can be seen in Appendix A.5.5. Figure 4.10 shows the average power measured for each applications. The horizontal dotted line represent the average of the benchmarks performed for that application.

Browser	Open Post Power (W)	Open Shared Power (W)	Open PNG Power (W)	Scroll Energio Power (W)	Open GIF Power (W)
BaconReader	1.040	1.102	1.113	1.363	1.877
Reddit	1.163	1.070	1.115	1.648	1.655
RedReader	0.980	0.987	1.451	1.426	1.770
Relay	0.822	0.905	1.359	1.351	1.581
rif is fun	0.831	1.047	0.952	1.372	1.796

Table 4.9: Reddit Applications Results

From the results it can be seen that rif is fun has performed the best in terms of energy, followed by Relay, BaconReader, Reddit Official and final RedReader. Reddit official is the only application that loads posts on the feed in the expanded form - images take the full width of the screen. This is evident in the Scroll Energio test, where due to this Reddit Official take the highest average power. On the other hand Reddit Official performs really well in opening GIFs, because while other applications use Android's Webview, Reddit launches a Google Chrome instance which is more efficient.

4.1.4 Establishing a rating

As discussed in Section 2.7, there are many ways of rating applications to be able to compare between them. Energio results can be represented in the following ways:

- **Measurement Ratings** - The average power measurements from each benchmark can be compared directly between applications for each tasks. To compare applications based on a

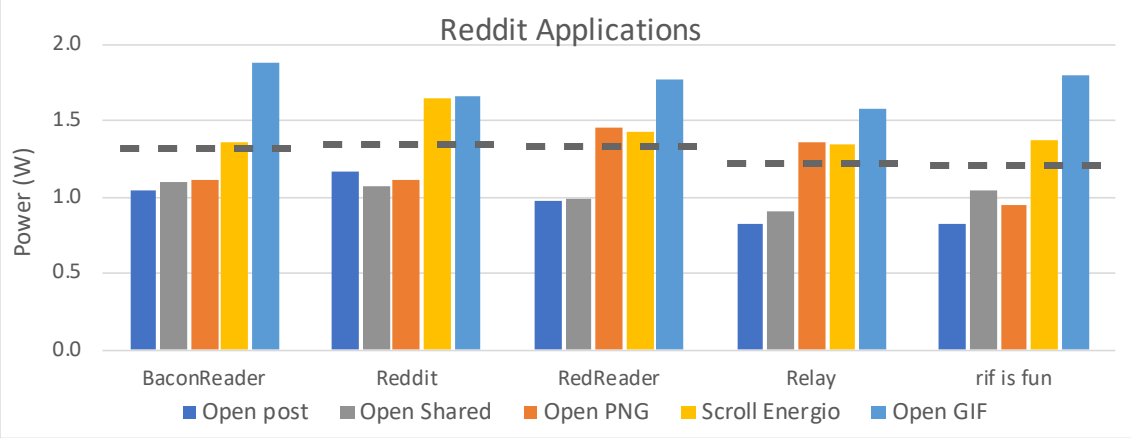


Figure 4.10: Energio Reddit Applications Results

set of task, a sum of average power for each task can be used or the average of all power averages. The averages of averages approach is better as it is allows more tasks to be added to the benchmarks without affecting the scale of the results.

- **Quantised Letter Ratings** - Just like European Union does for it's energy rating, a quantisation of results into buckets can be performed and a letter rating assigned to each bucket.

- **Percentage of Defeated Users** - Like Antutu benchmark provides the percentage of defeated users to give context of the performance, the percentage of defeated applications can also be part of a rating.

- **Relative Ratings** - The same approach as DXOMARK can be taken when results of particular tasks are normalized in relation to a specific marker. In the case of browsers, the benchmarking results were normalised in relation to Google Chrome.

Table 4.10 shows the relative rating, letter rating and percentage of defeated users approach for creating a rating for browsers. The Letter ratings for these ratings were arbitrarily chosen to be in the following ranges <900 → A, 900-1000 → B, 1000-1050 → C and >1050 → D. Just as before, more applications would have to be tested to make such range letter rating justifiable. The relative ratings can be displayed similar to how DXOMARK presents their results, showing the overall rating and relative performance for each tested task. This is shown in Figure 4.11.

Table 4.11 shows the measurement rating, letter rating and percentage of defeated users approach for creating a rating for Reddit applications. The letter ratings for these ratings ere arbitrarily chosen to be in the following ranges: <1.2 → A, 1.2-1.25 → B, 1.25-1.3 → C and >1.3 → D. More applications would have to be tested to make such range letter rating justifiable. Further to this a visualisation of a letter rating, along with measurement results is presented in a EU inspired rating in Figure 4.12.

Browser	**JavaScript Rating**	**Graphics Rating**	**Performance Rating**	**Overall Rating**	**Letter Rating**	**Defeated Apps (%)**
Brave	1013	1102	959	1025	B	60
Chrome	1000	1000	1000	1000	B	80
Edge	1018	1374	1012	1135	B	40
Firefox	1198	1500	1090	1263	D	20
Opera	998	1046	908	984	A	100

Table 4.10: Browser Applications Ratings Based on Benchmarks

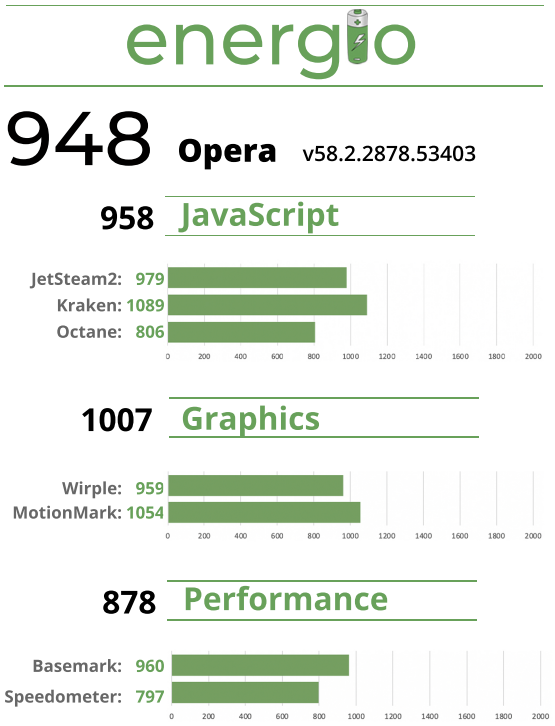


Figure 4.11: DXO Mark Inspired Energy App Rating: Opera

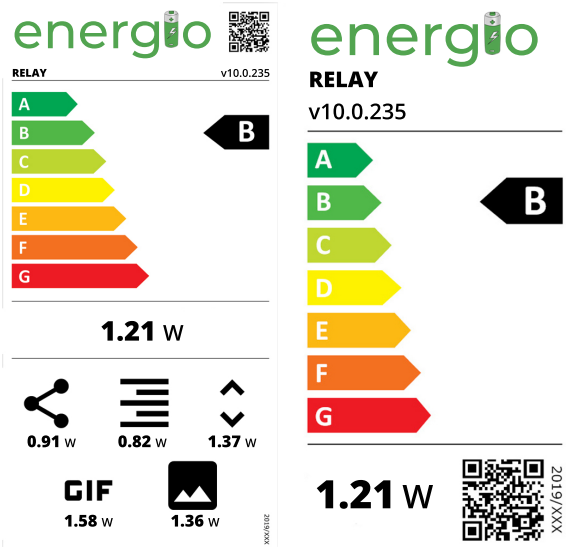


Figure 4.12: EU Inspired Energy App Rating: Relay

Reddit App	Sum Rating	Average Rating	Letter Rating	Defeated Apps (%)
BaconReader	6.495	1.299	C	60
Reddit	6.651	1.330	D	40
RedReader	6.613	1.323	D	20
Relay	6.017	1.203	B	80
rif is fun	5.998	1.200	A	100

Table 4.11: Reddit Applications Ratings

4.2 GreenHub

4.2.1 Google Pixel Results

Since the Energio benchmark platform measures the energy consumption on a Google Pixel 2 it would be beneficial to compare the results from the GreenHub dataset for this device. While 23 Google Pixel 2 devices are in the devices dataset and therefore enrolled to GreenHub, there are 0 samples present for these devices. This would imply that while Pixel 2 users have downloaded the GreenHub BatteryHub, the app was never kept in the background while using other applications. When extending the range to the whole Pixel line up there are 90 enrolled devices with 5588 samples for all applications. Out of the 10 applications tested through Energio, only 2 are present in the dataset - Google Chrome with 186 samples and Mozilla Firefox with 186 samples. Running the query from Section 3.4.1 that calculates the power drawn by these applications gives results in Table 4.12, where 'Average Power 1' refers to the average power found using the discharge method and 'Average Power 2' refers to the average power found using the voltage, current method.

Model	App Name	Importance	Number of Samples	Average Discharge (%/s)	Average Power 1 (W)	Average Power 2 (W)
Pixel 2 XL	Google Chrome	Service	190	0.010	2.812	2.041
Pixel 2 XL	Firefox	Service	112	0.011	11.135	2.107
Pixel XL	Google Chrome	Service	26	0.036	64.319	6.159

Table 4.12: GreenHub Results for Browsers on Pixel Devices

The results for the average power using the first method are not in the expected range of 0.2W-4W measured through Energio. This can be explained by the fact that this method considers the battery percentage drop over periods when the app was open. To get better results using this method, only data with longer opening times can be considered. Extending the query to only take into account continuous samples where the application was open for at least 30 seconds gives the results in table 4.13:

Model	App Name	Importance	Number of Samples	Average Discharge (%/s)	Average Power 1 (W)	Average Power 2 (W)
Pixel 2 XL	Google Chrome	Service	26	0.001	0.270	2.748
Pixel 2 XL	Firefox	Service	17	0.004	5.613	2.090

Table 4.13: GreenHub Filtered Results For Browsers on Pixel devices

While considering only samples where the app was open for at least 30 seconds, makes the method of discharge closer in magnitude to power measured through Energio, the results are still in ranges that are not reasonable (even running intensive benchmarks in Energio used 3W of power and a device in stand by used 0.2W). On the other hand, the approach of using voltage and current to calculate power gives results more acceptable results (except for Pixel XL) even with so few samples. It should be mentioned however, that the samples for browsers in Greenhub for Pixel devices are running as services instead of in the foreground. This should in turn mean a lower average power, but here results are as high as the power measured in benchmarks. When these browsers are run as a service, the power consumption is more representative of an another applications running in the foreground. This makes the results above not very accurate.

4.2.2 All devices Results

Browsers

The energy usage can be calculated across all devices and only when the desired application is in the foreground. Across the whole dataset for when the application is in the foreground and the phone is discharging there are 186 samples for Brave, 59 samples for Edge, 708 samples for Opera, 7012 samples for Firefox and 100,032 samples for Google Chrome.

The query from section 3.4.1 returns 165 rows of devices and the average power for applications. The full results are in Table A.20. From that table, the weighted average of average power weighted by the number of samples can be calculated to get the average power across all devices. The devices with more samples are given higher weighting as more samples should mean more accurate results. Table 4.14 shows the weighted average power for both methods. Measurement with Energio have always had all energy measurements in the range between 0.2 and 4W. As can be seen the average power in this table outside of this expected range. On inspection of the results Table 4.14 it can be noticed that 5 devices have the average power different by 3 orders of magnitude. Such a difference is most likely because current was stored in the table in amps instead of milliamps. However, to avoid speculation, all results where the average power (using method 2) is outside of the 0.2 to 4 Watts range are considered as outliers. Table 4.15 shows the average power without these outliers.

App Name	Number of Samples	Weighted Average 1 Power (W)	Weighted Average 2 Power (W)
Google Chrome	14967	27.803	29.751
Firefox	3519	0.100	1.532
Opera	41	6.386	3.267
Edge	23	5.210	1.502
Brave	12	3 760.969	1 457.874

Table 4.14: Greenhub: Browsers Query Average Power Results

App Name	Number of Samples	Weighted Average 1 Power (W)	Weighted Average 2 Power (W)
Google Chrome	13647	0.978	2.528
Firefox	3519	0.100	1.532
Opera	41	6.386	3.267
Edge	23	5.210	1.502

Table 4.15: Greenhub: Browsers Query Average Power Results with Samples Where Power Between 0.2 and 4 W

These results match in magnitude more what was found in Energio. Using these results, the browsers would be ranked as Edge, Firefox, Chrome, Opera. This does not match what was found through Energio. The results also vary among each other, which could be indicative of the workloads at time of measurement being different, hardware and software running these application being different, or errors in Greenhub sampling results.

Reddit

When it comes to the Reddit applications there is only one application that has samples for when it is running in the foreground - Reddit Official. Therefore, to calculate the average power all possible states are allowed. This brings number of samples to 26,098 for Reddit Offical, 10 for rif is fun, 437 for BaconReader, 1 for Relay and 2 for RedReader. The results using the SQL query from Section 3.4.1 are in Table A.21. Just as for browsers the weighted average of average power results weighted by number of samples can be calculated for each power calculation approach. This is shown in Table 4.16 below.

App Name	Number of Samples	Weighted Average 1 Power (W)	Weighted Average 2 Power (W)
Reddit	9715	2.190	2.027
rif is fun	2	635.040	2.544
RedReader	2	539.460	0.041

Table 4.16: GreenHub Results for Reddit Applications

These results show that only Reddit has enough samples to get an idea of an energy estimate. Interestingly, both the discharge approach and the voltage current approach results show close results. With enough samples this is something that one would expect to happen as the discharge method has much smaller precision. With more trials the results should converge towards the correct result by the law of big numbers. But at the same time as shown through the Browser results in Section 4.2.2 even more samples don't show such convergence.

4.2.3 GreeenHub Summary

Overall, it can be seen that the second approach of using voltage and current to calculate average power drawn gives more reasonable results.

While GreenHub is continuously gathering information, the released dataset not been updated since August 2019. This means that an approach of validating Energio measurements by recording

measurements while at the same time running the GreenHub BatteryHub could not be done and analysis had to rely on past data. Unfortunately, the dataset does not contain enough samples for all applications to make valid conclusions.

Also, as the results in Tables A.21 and A.20 show, the variability in results is very high and therefore even average power calculations across the devices are hard to establish. This can be accredited to results being coupled to external factors (other applications running, network, storage etc.), to varying hardware and software. Analysing the Greenhub dataset, therefore also shows how important keeping as many factors constant is in getting consistent results. The Energio platform is able to keep many factors constant across different tests and so results from Energio are much more consistent.

4.3 Greenspector

On their blog, Greenspector published a comparison and rating of mobile browsers, which contains a hardware energy measurement benchmark. They performed the tasks and measurements on Samsung S7 Smartphone, Android 8 on Wi-Fi with 50% brightness. These results can be tested against the results from using Energio to validate that the platform works and can be used for building comparison benchmarks like Greenspector have done.

4.3.1 Greenspector App Mark vs Energio

Greenspector App Mark considers 5 axes when establishing a rating for a mobile app, and does not focus solely on energy use. The score therefore covers more aspects of mobile applications than just energy.

Energio uses units of power (W) rather than charge (mAh) that Greenspector uses as explained in section 3.3.7. In summary, using power units is better for comparison across benchmark and different devices.

Energio is open source and as a benchmarking platform is available to anyone. Also, results that other people have gathered are openly accessible to anyone to view. While Greenspector have published 3 studies on their blog an a report on 1000 applications on the market, the company works on a consulting basis and does not give access to their benchmarking platform. The results published are also partial and do not give a full overview of the tests performed.

4.3.2 Navigation Benchmark

The methodology in the navigation benchmark for Greenspector was to open 6 websites (Wikipedia, Youtube, Pintreset, Walmart, Apple and NY Times) with a 20 second time in between. As seen in Figure 4.13, the test ranks the 5 browsers in the following order - Opera, Chrome, Brave, Edge and Firefox. This is very close to what was found in the browsing section 4.1.2 of Energio results, which found the order to be Opera, Brave, Chrome, Edge and Firefox. As the number of websites tested and the websites loaded are different, a small difference in results is to be expected.

4.3.3 Kraken Benchmark

Greenspector have also looked at Kraken benchmark and have compared browsers on their measure of efficiency - mAh consumed per a second of a benchmark. As $Energy\ (J) = 3.6 \times Voltage\ (V) \times Charge\ (mAh)$, their efficiency rating resembles more of a power measure and should be compared with the average power consumed in Energio benchmarks. Greenspector's results can be seen in Figure 4.14.

In Greenspector's results, the 5 browsers that were compared through Energio, rank in the following order - Opera, Firefox, Brave, Chrome and Edge. Energio for average power ranked these browsers as Opera, Edge, Chrome, Brave and Firefox. The order of results do not match what was found through Energio. This difference in results could be accredited to different devices used, different operating systems and different application versions. As systems can updated, the underlying hardware or even application software can have different performance. The fact that more demanding load like a JavaScript benchmark is more hardware intensive could then explain why Kraken results are uncorrelated, while navigation results are.

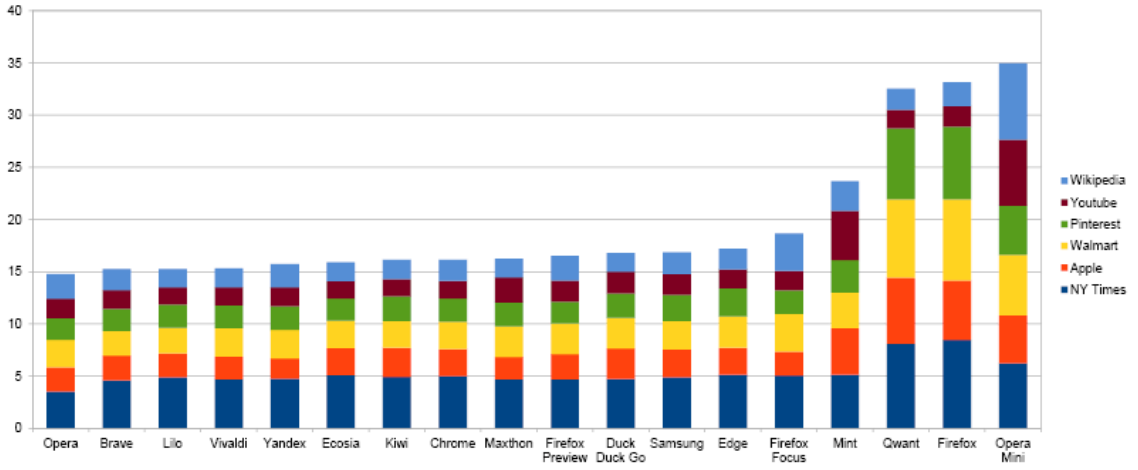


Figure 4.13: Greenspector - Browser Navigation source:
https://greenspector.com/en/what-are-the-best-web-browsers-to-use-in-2020/

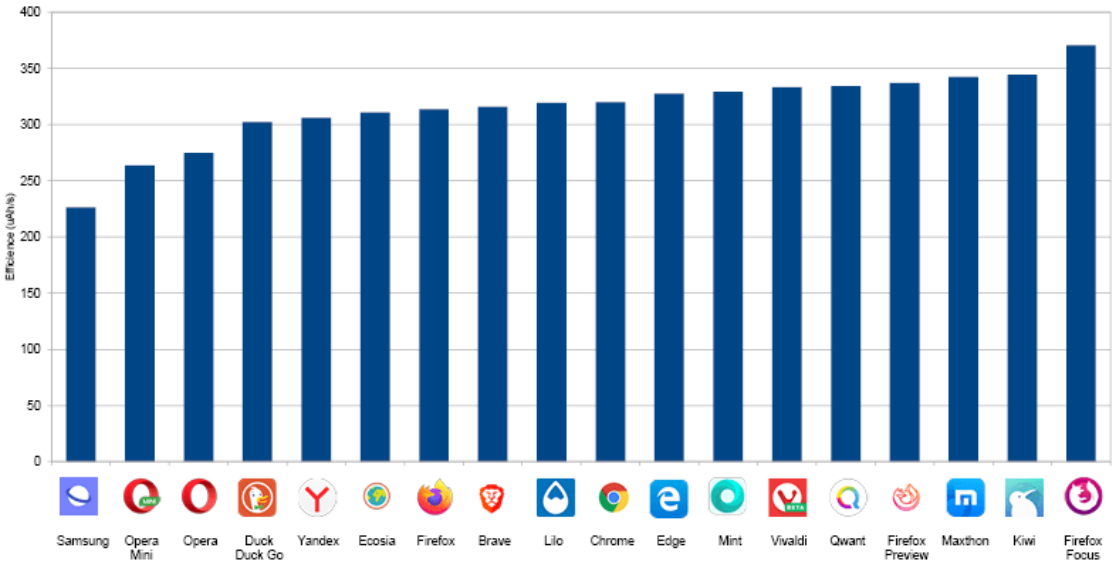


Figure 4.14: Greenspector - Kraken Efficiency source:
https://greenspector.com/en/what-are-the-best-web-browsers-to-use-in-2020/

4.4 Aeon

Aeon tested applications by running DroidMate-2, which explores different application paths, on Orka for 30 minutes. The results from Aeon [27] are in the table below:

Reddit App	Emulator Energy (J)	Device Energy (J)	Trepn Energy (J)
BaconReader	304.6	288.49	255.804
Reddit	447.87	377.07	287.304
RedReader	151.09	224.94	101.046
Relay	931.84	867.59	335.286
rif is fun	203.79	265.90	210.132

Table 4.17: Aeon Results

In his paper, D. Tsiang, concludes that Orka overestimates the power consumption of applications, but the relative results of Orka on software estimation are correct relative to another software estimation method - Trepn.

The results of Orka do not match what was found through Energio. Orka found these 5 Reddit applications to rank as RedReader, rif is fun, BaconReader, Reddit Official and Relay. The test done through Energio found the order to be rif is fun, Relay, BaconReader, Reddit Official and Red Reader.

The differences in the results could firstly be accredited to the fact that Energio measured the energy consumption through hardware, while Orka estimates the energy using API calls. It should be mentioned that Orka estimates the energy of only the running application, while Energio measures the energy consumption of the whole system. Orka uses an API energy call approach from 2014 to estimate energy usage, which could be inaccurate in Android 10. The energy results above where also gathered on a Samsung Galaxy S6 running Android 7, which could also influence the results.

Chapter 5

Conclusion

This project introduced the first open source energy benchmark platform that instead of energy estimates uses actual hardware measurements on a Google Pixel 2. The Energio website hosted on the local computer serves as a user interface with the benchmarking system and isolates the complexity of the underlying measurement system. The platform allows anyone to upload their UI automation scripts to get the energy measurements for it. The main feature of Energio is that scripts can be grouped together to form a benchmark that puts the energy usage of these applications against each other.

By evaluating two categories of applications - Browsers and Reddit applications, the Energio platform was verified to work, be capable of gathering results reliably and offer ways of identifying failures in the uploaded scripts or the platform. UI automation scripts were written using Monkeyrunner and AndroidViewClient to test these two categories of applications. This exercise has shown that the platform can be used remotely without access to the devices, and results are still consistent.

The findings from these benchmarks are that Opera preforms the best in terms of Energy, followed by Google Chrome, Brave, Edge and Firefox. By writing benchmarks that cover a wide range of tasks applications perform, the most battery inefficient parts of the applications can be identified too. For example, Firefox was found to be particularly inefficient when running JavaScript related tasks. Over the 6 tasks performed on 5 Reddit applications rif is fun performed the best, followed by Relay, BaconReader, RedReader and the official Reddit app.

Further to creating the Energio platform, an alternative way of getting energy performance of applications of applications was explored - data analysis of the GreenHub dataset. Two ways of extracting the power usage were explored - a battery discharge approach and a voltage current approach. The estimation of energy usage of applications based on the average current and average voltage over the course of running the application has shown more consistent results. However even the power estimation using this technique has shown too much variance across devices to be considered a good estimation of relative battery usage of applications. The query and ideas provided in this project can serve as a starting point for decoupling different factors and accounting for noise. Removing corrupted data may also be required.

5.1 Energio Discussion

1. **scalability & parallelization** - the benchmarking platform is scalable, but would need require more Otii devices along with a smartphone to be running in parallel. In terms of requests the web server could handle, the limitation for scalability is how much the network the computer is on can handle and how powerful the computer itself is.

2. **cost** - The costs of the simplest implementation is £455 for the Otii power monitor about £100 for the smartphone. It also requires a computer and optionally a domain to host the website. Using a hardware approach is therefore more expensive than using software estimation techniques, but the cost is not so high that it prevents people from setting up the platform themselves.

3. **time** - By comparison to other software approaches like Orka, the hardware benchmark is a lot more time consuming. It requires someone to write UI automation scripts. The

execution time is also high in comparison. The time it takes to perform a measurement is highly dependent on how long the UI automation script takes to run. The biggest weakness in terms of time execution is the queue. If there is a queue of benchmarks waiting to be executed then the time taken is a lot longer. This can be solved with the scalability considerations above.

4. **accuracy** - From the comparisons to results from Greenspector it can be seen that Energio's measurements are partially verified. Since the platform uses hardware measurements, the capabilities of the Otii power monitor define how accurate the platform is.

5. **portability** - The web server can be deployed anywhere, but has to be on a computer on an open network. The setup cannot be placed in the cloud, because access to special hardware needs to be done. It would be possible to host the website in the cloud and then have a separate web server waiting for local requests.

6. **availability** - One of the weakest sides of this implementation is that the website is only available when the computer is on and test can only conducted when both the Otii and the phone are connected.

5.2 Future Work

5.2.1 Testing more applications

In this project a limited number of applications were tested to test the feasibility of such tool for establishing an energy rating. However, creating a benchmarking tool where anyone can upload scripts for energy usage to be tested creates an opportunity to create new scripts and extend the benchmark tests to different categories and applications.

5.2.2 Continuous Integration Energy Readings

The platform could be integrated directly into git control to enable developers to commit UI automation scripts that are run every major merge, commit or on a set frequency basis. This would allow developers to see whether their change have impact on energy usage and catch any energy related issues before deployment. While there exist software tools like Energy Profiler that comes with Android Studio to warn about potential energy usage spikes, in practice the energy consumed could differ and with hardware measurements these issues would be avoided.

5.2.3 Deploying the website in the cloud

While there is a strict constraint where energy measurements have scheduled on a computer with direct connection to Otii, the website could be deployed anywhere. The website could be hosted in the cloud to make it capable of handling more traffic, while the measurement queue could be hosted on a local computer. The website could send requests to an open REST API provided by the web server of the computer to execute only requests associated with energy measurement and Otii controls.

5.2.4 Support for other operating systems and devices

In this project only energy benchmarking for Android smartphones was considered. However provided that UI automation scripts exist for an operating system and the device running it can be powered through the power monitor, the benchmarking platform could easily be extended to support other devices. For example, a windows laptop could be powered by the power monitor and AutoIt UI automation scripts could be run. Or IOS devices could be tested using Xcode's UI automation testing tool.

5.2.5 Django Channels

Currently the progress of benchmarks does not refresh automatically on the users screen and instead pages have to be refreshed manually to fetch updated information from the database. There exists

a project called Django Channels that offers web socket like functionality using ASGI and the Redis Server to communicate updates from the server to a user's screen.

5.2.6 Remote installation of APK packages

To further allow tests to be done remotely without access to the phone. A UI automaton script that searches for the required package name on the Play Store could be built to install the desired package. An alternative would be to allow users to upload any APK they want to the Energio platform, but preventative measures might need to be taken to prevent malicious users from installing malware APKs.

5.2.7 GreenHub Database

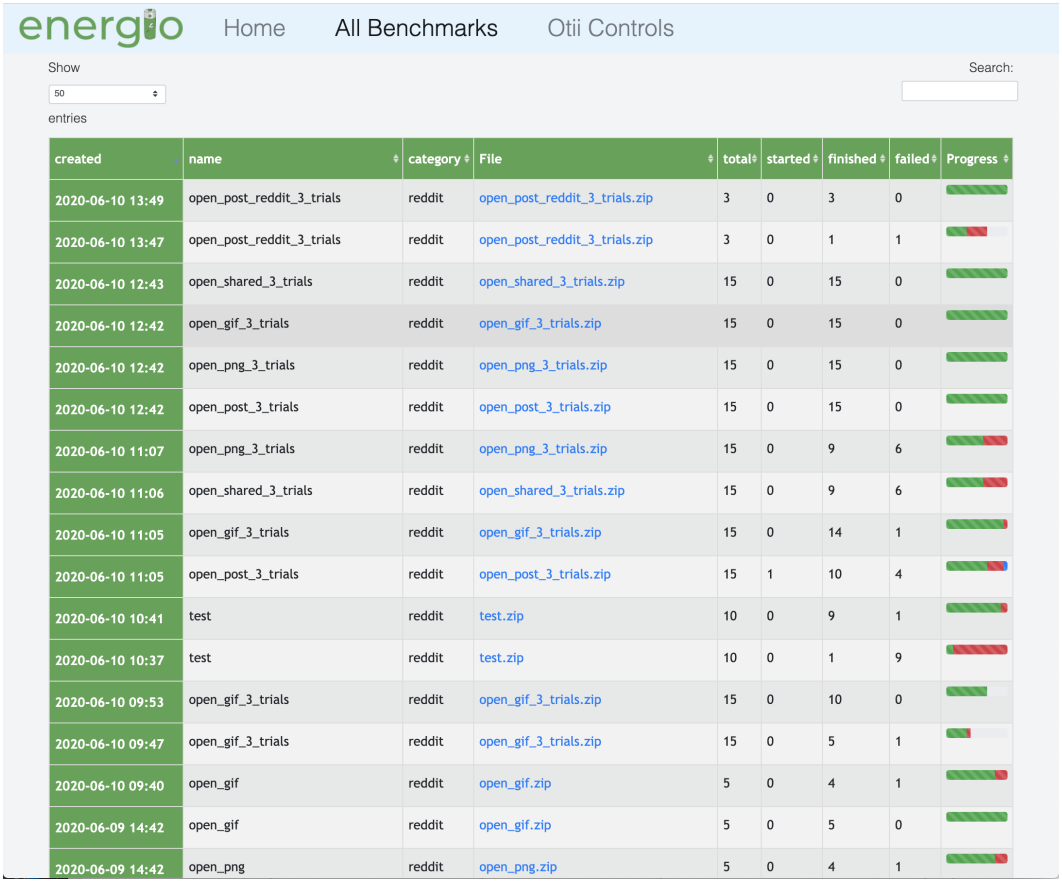
Setting up a faster database and optimizing the query built in this project would allow for the analysis of the whole table. The major bottleneck for analysis in the limited time frame was that even with indexing, even queries limited to 100 thousand samples took hours to run.

5.2.8 GreenHub Analysis

While this project built two ways for querying power usage of applications from GreenHub, the impact of other factors and decoupling of them from results was not taken into account. Further studies into the dataset could reveal the high variability of results across devices.

Appendix A

A.1 Energio Pages Screenshots



created	name	category	File	total	started	finished	failed	Progress
2020-06-10 13:49	open_post_reddit_3_trials	reddit	open_post_reddit_3_trials.zip	3	0	3	0	
2020-06-10 13:47	open_post_reddit_3_trials	reddit	open_post_reddit_3_trials.zip	3	0	1	1	
2020-06-10 12:43	open_shared_3_trials	reddit	open_shared_3_trials.zip	15	0	15	0	
2020-06-10 12:42	open_gif_3_trials	reddit	open_gif_3_trials.zip	15	0	15	0	
2020-06-10 12:42	open_png_3_trials	reddit	open_png_3_trials.zip	15	0	15	0	
2020-06-10 12:42	open_post_3_trials	reddit	open_post_3_trials.zip	15	0	15	0	
2020-06-10 11:07	open_png_3_trials	reddit	open_png_3_trials.zip	15	0	9	6	
2020-06-10 11:06	open_shared_3_trials	reddit	open_shared_3_trials.zip	15	0	9	6	
2020-06-10 11:05	open_gif_3_trials	reddit	open_gif_3_trials.zip	15	0	14	1	
2020-06-10 11:05	open_post_3_trials	reddit	open_post_3_trials.zip	15	1	10	4	
2020-06-10 10:41	test	reddit	test.zip	10	0	9	1	
2020-06-10 10:37	test	reddit	test.zip	10	0	1	9	
2020-06-10 09:53	open_gif_3_trials	reddit	open_gif_3_trials.zip	15	0	10	0	
2020-06-10 09:47	open_gif_3_trials	reddit	open_gif_3_trials.zip	15	0	5	1	
2020-06-10 09:40	open_gif	reddit	open_gif.zip	5	0	4	1	
2020-06-09 14:42	open_gif	reddit	open_gif.zip	5	0	5	0	
2020-06-09 14:42	open_png	reddit	open_png.zip	5	0	4	1	

Figure A.1: Energio Benchmarks Page

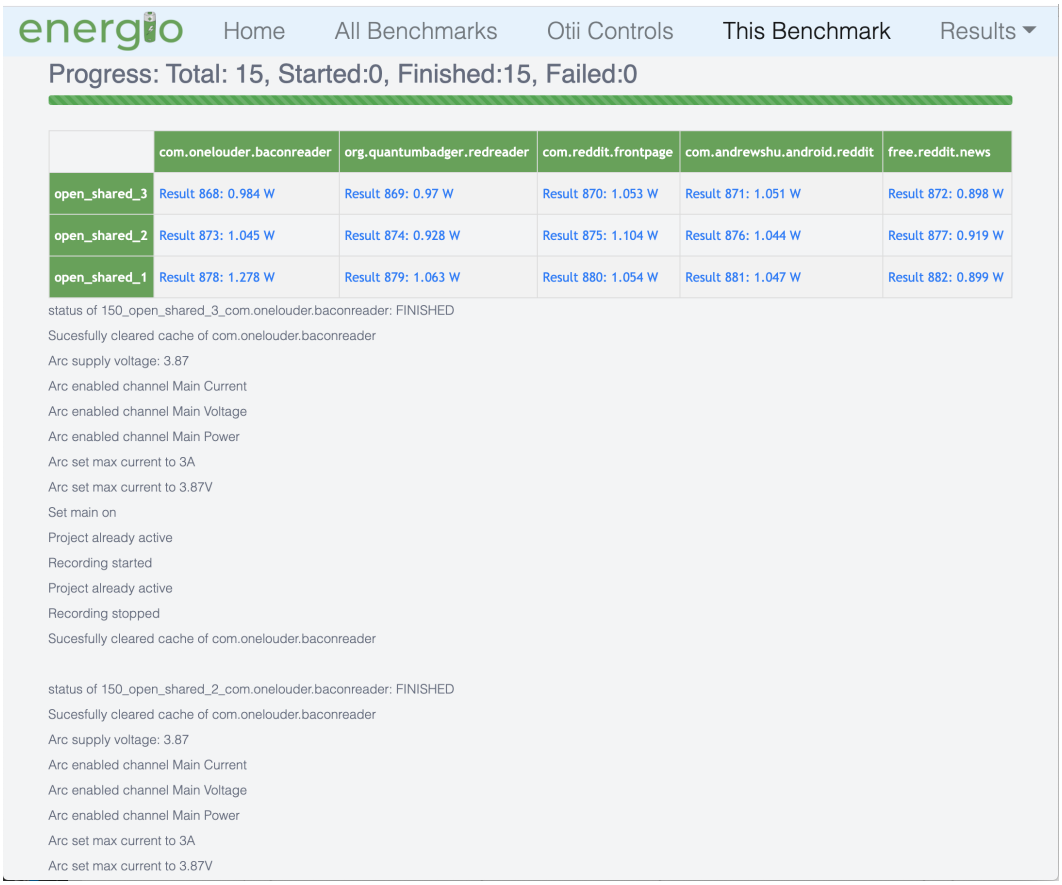


Figure A.2: Energio Benchmark Page

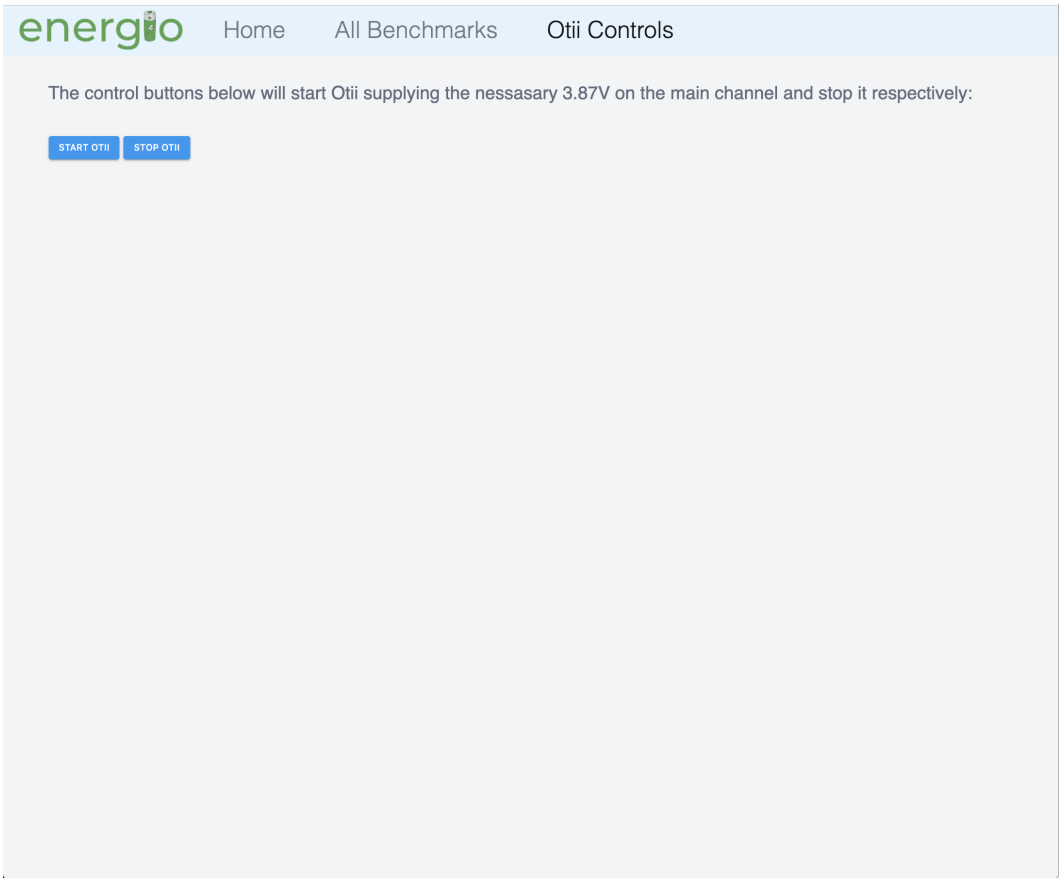


Figure A.3: Energio Otii Page

A.2 Greenhub SQL query

```
DROP TABLE IF EXISTS result;
CREATE TABLE result(model TEXT(65535), name TEXT(65535), num_samples int,
    avg_discharge FLOAT);
Insert ignore into result SELECT model, name,num_samples,avg_discharge from (
    with apps as (
    SELECT samples.id as id, importance, battery_details.voltage as voltage,
    current_now, devices.capacity as capacity, name, samples.device_id as device_id
    , battery_level, app_processes.created_at as d, model
    FROM app_processes JOIN samples ON app_processes.sample_id = samples.id JOIN
    devices ON samples.device_id = devices.id JOIN battery_details ON
    battery_details.sample_id = samples.id
    WHERE name in ('com.microsoft.emmx', 'com.brave.browser', 'com.android.chrome',
     'com.opera.browser', 'org.mozilla.firefox') and battery_state='Discharging'
    limit 1000000
),
t_start as (
    select *
from (
    SELECT T1.id,
        T1.device_id as device_id,
        T1.battery_level as battery_level,
        T1.d as d,
        IFNULL(TIME_TO_SEC(TIMEDIFF(T1.d, Max(T2.d))),5000) AS TimeDiff
    FROM apps T1
        LEFT JOIN apps T2
        ON T1.device_id = T2.device_id
            AND T2.d < T1.d
    GROUP BY T1.device_id, T1.d
    ) as x
where x.TimeDiff > 40
),
t_end as (
    select *
from (SELECT T1.id,
        T1.device_id as device_id,
        T1.battery_level as battery_level,
        T1.d as d,
        IFNULL(TIME_TO_SEC(TIMEDIFF(T1.d, MIN(T2.d))),-5000) AS TimeDiff
    FROM apps T1
        LEFT JOIN apps T2
        ON T1.device_id = T2.device_id
            AND T2.d > T1.d
    GROUP BY T1.device_id, T1.d) as x
where x.TimeDiff < -40
),
t_boundaries as (
    SELECT
    T1.device_id as device_id,
    T1.d as t1,
    MIN(T2.d) AS t2,
    TIME_TO_SEC(TIMEDIFF(T2.d, T1.d)) as time_diff
FROM t_start T1
    LEFT JOIN t_end T2
    ON T1.device_id = T2.device_id
        AND T2.d >= T1.d
GROUP BY T1.device_id, T1.d
HAVING time_diff > 10
)
select
    model,
    name,
    importance,
    sum(count) as num_samples,
    sum(total_discharge) / sum(TimeDiff) as avg_discharge,
    sum(total_discharge) / sum(TimeDiff) * capacity * 3.6 / sum(TimeDiff) * AVG(
    voltage) as avg_power,
    AVG(voltage) * AVG(current_now) / -1000 as avg_power2

    FROM (
        select t1.device_id as device_id,
            count(*) as count,
```

```
            model,
            capacity,
            name,
            voltage,
            current_now,
            importance,
            (MAX(battery_level) - min(battery_level)) as total_discharge,
            MAX(battery_level) as max,
            min(battery_level) as min,
            GROUP_CONCAT(battery_level),
            TIME_TO_SEC(TIMEDIFF(t_boundaries.t2,t_boundaries.t1)) AS TimeDiff
        FROM apps t1
            INNER JOIN t_boundaries on t1.d BETWEEN t_boundaries.t1 and
    t_boundaries.t2
        where t1.device_id = t_boundaries.device_id
        and current_now < 0
        group by t_boundaries.t1
        having TimeDiff<>0
    ) as y
where y.total_discharge > 0
group by y.model, y.name, y.importance) as t;
SET @sql = NULL;
SELECT
  GROUP_CONCAT(DISTINCT
     CONCAT(
       'MAX(IF(name = ''',
   name,
   ''', avg_discharge, NULL)) AS ',
   CONCAT("'",name,"'")
 )
   ) INTO @sql
FROM result;

SET @sql = CONCAT('SELECT model, ', @sql, ' FROM result GROUP BY model');

PREPARE stmt FROM @sql;
EXECUTE stmt;
```

Listing A.1: Greenhub SQL query

A.3 Package Names and Version Codes of Applications Tested

A.3.1 Browsers

App Name	Package Name	Version Code
Brave	com.brave.browser	1.8.112
Edge	com.microsoft.emmx	45.03.4.4958
Firefox	org.mozilla.firefox	68.8.1
Google Chrome	com.android.chrome	83.0.4103.83
Opera	com.opera.browser	58.2.2878.53403

Table A.1: Browsers: Package Names and Version Codes

A.3.2 Reddit Applications

App Name	Package Name	Versin Code
BaconReader	com.onelouder.baconreader	5.7.0
Reddit	com.reddit.frontpage	2020.20.1
RedReader	org.quantumbadger.redreader	1.9.11
Relay	free.reddit.news	10.0.235
rif is fun	com.andrewshu.android.reddit	4.16.19

Table A.2: Reddit Applications: Package Names and Version Codes

A.4 Top 100 most visited websites on mobile devices

	Website		Website		Website
1	google.com	35	reddit.com	69	livejasmin.com
2	facebook.com	36	msn.com	70	foxnews.com
3	youtube.com	37	news.yahoo.co.jp	71	tribunnews.com
4	baidu.com	38	ebay.com	72	cookpad.com
5	ucnews.in	39	accuweather.com	73	google.de
6	wikipedia.org	40	rakuten.co.jp	74	zhihu.com
7	twitter.com	41	whatsapp.com	75	craigslist.org
8	xvideos.com	42	bbc.com	76	redtube.com
9	instagram.com	43	mail.ru	77	amazon.de
10	pornhub.com	44	bbc.co.uk	78	duckduckgo.com
11	xnxx.com	45	bing.com	79	anybunny.tv
12	yahoo.co.jp	46	fandom.com	80	twitch.tv
13	google.com.br	47	ok.ru	81	news.google.com
14	yandex.ru	48	office.com	82	dailymail.co.uk
15	yahoo.com	49	weather.com	83	ameblo.jp
16	bit.ly	50	irs.gov	84	pornhubpremium.com
17	naver.com	51	bongacams.com	85	nytimes.com
18	youtu.be	52	tiktok.com	86	archiveofourown.org
19	amazon.com	53	quora.com	87	cgtn.com
20	xhamster.com	54	auone.jp	88	chaturbate.com
21	worldometers.info	55	daum.net	89	livedoor.jp
22	globo.com	56	youporn.com	90	microsoftonline.com
23	ucweb.com	57	taobao.com	91	google.co.in
24	smt.docomo.ne.jp	58	amazon.co.jp	92	goo.ne.jp
25	zoom.us	59	walmart.com	93	nbryb.com
26	taboola.com	60	sohu.com	94	linkedin.com
27	live.com	61	imdb.com	95	netflix.com
28	qq.com	62	line.me	96	news.naver.com
29	samsung.com	63	fc2.com	97	tsyndicate.com
30	cnn.com	64	covid19india.org	98	hurriyet.com.tr
31	vk.com	65	sogou.com	99	syosetu.com
32	pinterest.com	66	theguardian.com	100	apple.com
33	outbrain.com	67	wordpress.com		
34	uol.com.br	68	paypal.com		

Table A.3: Top 100 most visited websites on mobile devices

A.5 Energio Results

A.5.1 Browsing Top Websites

Browsing Top 5

Browser	**Trial 1 Power (W)**	**Trial 2 Power (W)**	**Trial 3 Power (W)**	**Average Power (W)**
Brave	1.432	1.480	1.437	1.450
Edge	1.642	1.597	1.580	1.606
Firefox	1.501	1.561	1.626	1.563
Google Chrome	1.492	1.439	1.457	1.463
Opera	1.421	1.397	1.428	1.415

Table A.4: Browsers: Browsing Top 5 Task Results

Browsing Top 10

Browser	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)
Brave	1.537	1.491	1.611	1.546
Edge	1.649	1.702	1.726	1.692
Firefox	1.657	1.788	1.811	1.752
Google Chrome	1.574	1.605	1.625	1.601
Opera	1.441	1.496	1.439	1.459

Table A.5: Browsers: Browsing Top 10 Task Results

Browsing Top 100

Browser	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)
Brave	1.694	1.672	1.709	1.692
Edge	2.004	1.914	1.953	1.957
Firefox	2.303	2.224	2.283	2.270
Google Chrome	1.963	1.947	1.917	1.942
Opera	1.637	1.759	1.598	1.665

Table A.6: Browsers: Browsing Top 100 Task Results

A.5.2 JavaScript Benchmarks

Octane

Browser	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)	Trial 1 Score	Trial 2 Score	Trial 3 Score	Average Score
Brave	1.873	2.013	1.885	1.924	10521	10558	10571	10550
Chrome	1.768	1.985	1.890	1.881	10567	10506	10475	10516
Edge	2.118	2.076	2.030	2.075	11337	11364	11302	11334
Firefox	1.791	1.838	1.803	1.811	6406	6489	6481	6459
Opera	1.682	1.763	1.813	1.753	10990	10588	10996	10858

Table A.7: Browsers: Octane Results

Kraken

Browser	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)	Trial 1 Score	Trial 2 Score	Trial 3 Score	Average Score
Brave	1.988	2.018	2.013	2.006	3826.3	3789	3812.4	3809.2
Chrome	1.998	2.014	2.011	2.008	3722.6	3728.5	3705	3718.7
Edge	1.924	1.960	2.042	1.975	3669.7	3667.7	3636.6	3658
Firefox	2.268	2.115	2.231	2.205	3304.5	3305.2	3292.2	3300.6
Opera	1.827	1.757	1.910	1.831	3881.9	3795	3795.1	3824.0

Table A.8: Browsers: Kraken Results

JetStream2

Browser	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)	Trial 1 Score	Trial 2 Score	Trial 3 Score	Average Score
Brave	2.261	2.235	2.249	2.248	33.583	34.000	34.029	33.871
Chrome	2.319	2.296	2.297	2.304	34.357	34.547	34.674	34.526
Edge	2.397	2.381	2.401	2.393	34.797	34.719	34.796	34.771
Firefox	2.011	2.105	2.088	2.068	24.245	24.690	24.612	24.516
Opera	2.282	2.226	2.258	2.255	34.533	34.600	34.451	34.528

Table A.9: Browsers: JetStream2 Results

A.5.3 Graphics Benchmarks

MotionMark

Browser	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)	Trial 1 Score	Trial 2 Score	Trial 3 Score	Average Score
Brave	2.165	2.127	2.173	2.155	44.770	57.220	56.600	52.863
Chrome	2.205	2.218	2.212	2.212	60.290	58.620	59.320	59.410
Edge	2.174	2.093	2.187	2.151	31.280	30.770	40.600	34.217
Firefox	1.804	1.793	1.756	1.784	26.860	26.530	24.660	26.017
Opera	2.213	2.162	2.137	2.171	57.400	49.240	48.840	51.827

Table A.10: Browsers: MotionMark Results

Wirple

Browser	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)	Trial 1 Score	Trial 2 Score	Trial 3 Score	Average Score
Brave	2.225	2.284	2.251	2.253	864	1019	984	956
Chrome	2.318	2.295	2.269	2.294	1062	1080	1096	1079
Edge	2.242	2.285	2.005	2.177	932	1045	924	967
Firefox	2.067	2.053	1.975	2.032	851	760	864	825
Opera	2.209	2.157	2.140	2.169	1055	1030	1079	1055

Table A.11: Browsers: Wirple Results

A.5.4 Performance Benchmarks

Speedometer

Browser	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)	Trial 1 Score	Trial 2 Score	Trial 3 Score	Average Score
Brave	2.054	2.049	2.061	2.055	20.800	20.500	20.700	20.667
Chrome	2.138	2.147	2.139	2.141	18.400	18.800	18.900	18.700
Edge	2.179	2.175	2.158	2.171	18.500	18.600	18.900	18.667
Firefox	1.820	1.875	1.881	1.859	18.600	18.200	18.500	18.433
Opera	2.077	2.078	2.081	2.079	21.400	21.500	21.500	21.467

Table A.12: Browsers: Speedometer Results

Basemark

Browser	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)	Trial 1 Score	Trial 2 Score	Trial 3 Score	Average Score
Brave	3.018	3.036	3.071	3.042	226.65	237.04	228.67	230.79
Chrome	2.962	3.178	3.117	3.086	242.63	249.51	245.68	245.94
Edge	2.891	2.934	3.120	2.982	229.76	238.17	239.29	235.74
Firefox	2.848	2.963	3.084	2.965	183.56	183.76	178.12	181.81
Opera	2.975	3.067	3.177	3.073	253.91	262.38	241.49	252.59

Table A.13: Browsers: Basemark Results

A.5.5 Reddit Applications Energy Results

Open PNG Task

Reddit App	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)
BaconReader	1.083	1.109	1.146	1.113
Reddit	1.150	1.120	1.076	1.115
RedReader	1.196	1.980	1.176	1.451
Relay	1.462	1.320	1.294	1.359
rif is fun	0.975	0.949	0.931	0.952

Table A.14: Reddit: Open PNG Task Results

Open Post Task

Reddit App	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)
BaconReader	1.073	1.017	1.031	1.040
Reddit	1.141	1.165	1.182	1.163
RedReader	0.999	0.990	0.950	0.980
Relay	0.822	0.864	0.780	0.822
rif is fun	0.832	0.833	0.829	0.831

Table A.15: Reddit: Open Post Task Results

Open GIF Task

Reddit App	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)
BaconReader	1.859	1.863	1.909	1.877
Reddit	1.717	1.550	1.698	1.655
RedReader	1.768	1.759	1.784	1.770
Relay	1.553	1.574	1.615	1.581
rif is fun	1.778	1.839	1.772	1.796

Table A.16: Reddit: Open GIF Task Results

Open Shared Task

Reddit App	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)
BaconReader	0.984	1.045	1.278	1.102
Reddit	1.053	1.104	1.054	1.070
RedReader	0.970	0.928	1.063	0.987
Relay	0.898	0.919	0.899	0.905
rif is fun	1.051	1.044	1.047	1.047

Table A.17: Reddit: Open Shared Task Results

Scrolling Energio Task

Reddit App	Trial 1 Power (W)	Trial 2 Power (W)	Trial 3 Power (W)	Average Power (W)
BaconReader	1.314	1.380	1.394	1.363
Reddit	1.650	1.636	1.657	1.648
RedReader	1.489	1.473	1.315	1.426
Relay	1.373	1.327	1.352	1.351
rif is fun	1.327	1.473	1.315	1.372

Table A.18: Reddit: Scrolling Energio Task Results

A.6 GreenHub Results

A.6.1 Browsers

model	App Name	Number of Samples	Average Time Running (s)	Average Discharge (%/s)	Average Power 1 (W)	Average Power 2 (W)
A574BL	Google Chrome	3052	24.680	0.008	0.053	2.153
ASUS_Z007	Google Chrome	1080	58.627	0.004	0.026	2.333
Lenovo A6010	Google Chrome	1057	35.395	0.007	0.151	3.863
ASUS_T00J	Google Chrome	833	29.083	0.006	(null)	2.438
Andromax C46B2H	Google Chrome	589	34.878	0.005	(null)	2.124
ASUS_T00F	Google Chrome	567	28.733	0.006	(null)	2.605
ONE A2001	Google Chrome	357	49.833	0.002	0.092	2.589
Z557BL	Google Chrome	330	37.563	0.006	0.436	2.337
Andromax A A16C3H	Google Chrome	308	17.692	0.008	(null)	3.977
LGMP260	Google Chrome	304	34.353	0.005	0.166	1.636
XT1060	Google Chrome	272	80.091	0.003	0.090	2.398
5056N	Google Chrome	267	39.000	0.002	0.037	2.492
5056A	Google Chrome	252	36.682	0.003	0.135	3.178
GT-I9195	Google Chrome	217	24.588	0.013	0.404	2.334
Andromax B26D2H	Google Chrome	205	19.000	0.010	(null)	3.322
VS986	Google Chrome	187	24.037	0.003	0.194	2.658
GT-N8000	Google Chrome	178	131.158	0.002	0.069	0.004
LGMS210	Google Chrome	171	18.080	0.010	753.809	1 698.280
SH-05F	Google Chrome	156	43.000	0.004	(null)	1.697
LG-TP260	Google Chrome	143	17.750	0.009	1.566	2.077
LG-H810	Google Chrome	133	34.286	0.010	1.580	3.158
LG-M153	Google Chrome	133	22.539	0.010	1.191	2.397
XT1585	Google Chrome	113	25.429	0.007	2.001	3.745
ARK Benefit A3	Google Chrome	111	69.250	0.009	(null)	6.083
SM-T561M	Google Chrome	111	20.539	0.004	1.009	0.004
RIDGE FAB 4G	Google Chrome	107	24.143	0.006	1.536	3.895
QMobile Z9	Google Chrome	105	25.091	0.005	0.488	2.507
ASUS_Z002	Google Chrome	102	32.000	0.005	0.957	2.775
ASUS_T00G	Google Chrome	98	45.833	0.005	(null)	1.817
VS501	Google Chrome	96	15.857	0.009	3.179	1.587

model	App Name	Number of Samples	Average Time Running (s)	Average Discharge (%/s)	Average Power 1 (W)	Average Power 2 (W)
Andromax A16C3H	Google Chrome	90	35.412	0.002	(null)	3.726
Moto G (5)	Google Chrome	89	15.900	0.007	1.784	1.661
C6833	Google Chrome	87	19.263	0.005	0.564	3.210
HUAWEI SCC-U21	Google Chrome	87	19.750	0.007	1.332	3.978
Redmi Note 3	Google Chrome	82	33.750	0.008	1.695	2.373
Z851M	Google Chrome	79	43.500	0.009	1.352	2.749
Moto E (4)	Google Chrome	77	29.100	0.007	0.885	1.917
Z981	Google Chrome	77	38.250	0.004	0.512	3.714
GT-P5200	Google Chrome	76	41.000	0.005	0.957	0.004
VS980 4G	Google Chrome	76	32.889	0.006	0.776	0.114
Lenovo A6000	Google Chrome	74	15.286	0.008	2.414	1.865
Z558VL	Google Chrome	74	19.000	0.007	2.618	2.685
GT-I9190	Google Chrome	69	20.900	0.006	0.702	1.645
LG-K120	Google Chrome	67	22.556	0.005	0.665	1.454
Z958	Google Chrome	66	15.750	0.013	4.447	2.051
General Mobile 4G Dual	Google Chrome	65	17.000	0.010	(null)	4.300
LGL158VL	Google Chrome	62	32.100	0.004	0.533	2.589
Z220	Google Chrome	57	28.667	0.007	1.248	1.831
ASUS_Z00ED	Google Chrome	56	31.250	0.007	1.823	5.284
A0001	Google Chrome	53	20.444	0.007	1.642	0.004
Z839	Google Chrome	50	28.333	0.004	(null)	2.919
LG-H815	Google Chrome	49	30.667	0.008	3.403	2.377
Lenovo A2020a40	Google Chrome	48	39.286	0.003	0.314	3.705
LM-X210(G)	Google Chrome	48	20.000	0.013	4.318	0.651
SM-T560	Google Chrome	48	22.000	0.005	2.723	0.004
SM-G3815	Google Chrome	47	25.800	0.006	1.262	3.000
Ilium LT500	Google Chrome	42	26.571	0.005	(null)	7.727
Aquaris M5	Google Chrome	41	19.600	0.005	2.089	2.757
Lenovo K33b36	Google Chrome	41	16.250	0.010	6.062	2.437
RS988	Google Chrome	41	11.000	0.014	15.939	2.482
Z959	Google Chrome	41	26.750	0.005	1.871	6.530
LG-H950	Google Chrome	40	19.000	0.006	3.105	3.240
LGUS215	Google Chrome	39	22.800	0.006	1 904.810	1 863.580
E2363	Google Chrome	38	33.500	0.009	4.576	2.559
LM-X410.F	Google Chrome	34	36.667	0.004	1.410	1.536
GT-I9192	Google Chrome	33	36.000	0.004	0.966	3.027
Andromax A26C4H	Google Chrome	32	35.000	0.009	(null)	3.780
ASUS_A009	Google Chrome	32	75.500	0.003	0.737	1.678
Z3001S	Google Chrome	32	52.333	0.009	1.603	2.196
LG-D855	Google Chrome	31	25.667	0.003	1.729	0.034
L50	Google Chrome	30	20.500	0.006	1.834	1.994
SM-N930F	Google Chrome	30	22.000	0.005	3.503	0.003
LT30p	Google Chrome	29	12.000	0.018	19.358	2.030
MI 5s	Google Chrome	29	15.500	0.026	11.895	2.499
YD201	Google Chrome	29	65.750	0.000	0.056	3.191

model	App Name	Number of Samples	Average Time Running (s)	Average Discharge (%/s)	Average Power 1 (W)	Average Power 2 (W)
LG-D802	Google Chrome	28	24.000	0.003	1.738	0.100
LG-D410	Google Chrome	26	42.500	0.002	0.935	1.328
Andromax B16C2H	Google Chrome	25	20.333	0.006	(null)	6.156
E6653	Google Chrome	25	25.333	0.007	3.647	1.332
HUAWEI G7-L03	Google Chrome	25	44.000	0.003	1.161	0.805
ONEPLUS A3003	Google Chrome	25	15.500	0.011	15.596	1.554
Redmi Note 5A	Google Chrome	25	28.000	0.005	3.804	2.467
VS425PP	Google Chrome	25	20.000	0.004	(null)	1.935
A577VL	Google Chrome	24	20.000	0.005	4.994	3.149
Alcatel_5044C	Google Chrome	24	17.000	0.006	3.401	1.833
Aquaris X5 Plus	Google Chrome	24	49.333	0.001	0.456	2.813
E6853	Google Chrome	24	20.000	0.016	12.349	2.562
LGM-K121K	Google Chrome	24	47.500	0.003	1.151	2.489
Moto G (5S) Plus	Google Chrome	24	42.000	0.007	(null)	1.509
SO-02F	Google Chrome	24	21.250	0.015	5.307	2.611
XT1045	Google Chrome	24	34.000	0.017	4.725	1.833
GT-S7275R	Google Chrome	23	52.400	0.002	0.144	1.474
VF-895N	Google Chrome	22	31.667	0.002	0.605	3.283
305SH	Google Chrome	19	15.750	0.005	2.309	1.736
GT-N8013	Google Chrome	19	33.571	0.005	1.793	0.004
LG-M703	Google Chrome	19	17.000	0.005	5.956	2.139
Moto G (4)	Google Chrome	18	17.000	0.009	20.292	3.522
C103	Google Chrome	17	27.333	0.007	4.694	2.873
LGT02	Google Chrome	17	42.000	0.007	11 650.000	2 669.710
HS-L671	Google Chrome	16	13.333	0.006	(null)	3.980
XT1058	Google Chrome	16	16.000	0.009	19.305	2.263
ASUS_Z00RD	Google Chrome	15	28.000	0.002	1.017	2.268
Redmi Note 4	Google Chrome	15	23.500	0.007	9.042	1.351
C6743	Google Chrome	14	40.000	0.004	2.466	5.147
XT1033	Google Chrome	14	18.500	0.004	2.926	2.926
RAINBOW LITE 4G	Google Chrome	13	67.000	0.002	0.693	4.399
DLI-TL20	Google Chrome	12	12.000	0.037	132.548	3.380
E2306	Google Chrome	12	13.000	0.016	37.147	3.913
K010	Google Chrome	12	20.500	0.024	38.535	2.137
SM-T561	Google Chrome	12	38.000	0.002	1.065	0.003
LG-H440AR	Google Chrome	11	15.000	0.013	12.230	2.646
Redmi 4X	Google Chrome	11	36.500	0.004	2.912	1.900
SM-N910F	Google Chrome	11	47.000	0.003	0.559	0.004
GT-N8020	Google Chrome	9	83.500	0.002	1.056	0.004
XT1032	Google Chrome	9	16.000	0.008	13.737	1.314
Karbonn Aura 1	Google Chrome	8	22.000	0.010	(null)	5.692
LG-M430	Google Chrome	8	13.000	0.011	35.024	1.846
C6603	Google Chrome	7	14.000	0.001	1.566	2.846
KYOCERA-C6742	Google Chrome	7	16.000	0.004	7.649	3.149

model	App Name	Number of Samples	Average Time Running (s)	Average Discharge (%/s)	Average Power 1 (W)	Average Power 2 (W)
Lenovo A6020l36	Google Chrome	7	43.000	0.001	0.540	7.899
S4035_4G	Google Chrome	7	32.000	0.009	(null)	3.347
ASUS_X014D	Google Chrome	6	16.000	0.001	0.546	1.618
GT-I9301I	Google Chrome	6	61.500	0.009	2.049	0.289
LS-5016	Google Chrome	6	14.000	0.004	8.379	1.851
LGUS110	Google Chrome	5	20.500	0.006	4.053	1.667
MotoE2	Google Chrome	5	27.000	0.001	1.747	2.135
SAMSUNG-SM-N910A	Google Chrome	5	25.000	0.004	7.630	0.004
STARADDICT 6	Google Chrome	5	14.000	0.006	(null)	1.772
X10	Google Chrome	5	12.000	0.053	85.680	5.016
6055K	Google Chrome	4	32.000	0.001	1.197	0.030
Coolpad A8	Google Chrome	4	52.000	0.010	7.629	5.446
G620S-L01	Google Chrome	4	21.500	0.000	0.289	1.369
MI 2S	Google Chrome	4	35.000	0.001	0.435	3.078
Neffos Y5	Google Chrome	4	15.000	0.003	9.012	2.682
SM-T113NU	Google Chrome	4	16.500	0.003	3.920	0.004
UL40	Google Chrome	4	24.000	0.001	2.024	2.631
6070K	Google Chrome	3	20.000	0.002	3.078	1.136
Aquaris X5	Google Chrome	3	13.000	0.002	5.053	1.260
C5502	Google Chrome	3	33.000	0.001	0.607	1.835
Coolpad 5267	Google Chrome	3	18.000	0.008	(null)	1.323
GT-I8730	Google Chrome	3	59.000	0.004	2.151	2.360
LG-M250	Google Chrome	3	11.000	0.015	47.451	3.314
LG-MS770	Google Chrome	3	11.000	0.002	3.621	1.121
RIDGE 4G	Google Chrome	3	11.000	0.003	8.483	5.904
Z899VL	Google Chrome	3	39.000	0.000	0.270	1.672
5027B	Google Chrome	2	18.000	0.019	18.617	2.661
ASUS_T00I	Google Chrome	2	16.000	0.003	2.450	2.269
K014	Google Chrome	2	13.000	0.001	5.455	0.004
Le 1 Pro	Google Chrome	2	21.000	0.004	10.736	1.525
Lenovo K33a48	Google Chrome	2	12.000	0.001	2.738	2.307
LS-4505	Google Chrome	2	13.000	0.002	2.968	2.550
Moto Z2 Play	Google Chrome	2	11.000	0.073	294.188	1.590
ONE E1003	Google Chrome	2	14.000	0.001	1.679	2.411
Redmi 5A	Google Chrome	2	19.000	0.019	40.403	2.537
SM-J327T1	Google Chrome	2	17.000	0.006	12.377	0.004
XT1580	Google Chrome	2	17.000	0.001	3.794	1.669
LGMS210	Brave	12	13.000	0.002	3 760.970	1 457.870
GT-I9195	Edge	23	18.500	0.022	5.210	1.502
ASUS_Z011D	Opera	41	18.333	0.009	6.386	3.267
GT-I9192	Firefox	3295	40.776	0.004	0.008	1.422
HTC 10	Firefox	155	16.000	0.007	1.520	3.261
GT-I9195	Firefox	25	28.400	0.011	1.978	3.550
LG-M150	Firefox	22	28.000	0.004	(null)	2.892
Redmi 4X	Firefox	20	36.000	0.002	0.678	1.987
ASUS_Z002	Firefox	2	17.000	0.005	13.134	3.610

Table A.20: Greenhub: Browsers Query Power Results

A.6.2 Reddit Applications

model	App Name	Import-ance	Number of Samples	Average Time Running (s)	Average Discharge (%/s)	Average Power 1 (W)	Average Power 2 (W)
Nexus 5	Reddit	Service	3498	10.527	0.019	0.345	1.886
ONE A2003	Reddit	Service	1715	20.535	0.018	0.394	3.406
ONEPLUS A3003	Reddit	Service	1088	12.254	0.014	0.784	1.537
MI 5	Reddit	Service	1085	7.197	0.031	2.418	1.391
S60	Reddit	Service	782	22.058	0.010	0.454	1.120
Redmi 4X	Reddit	Service	423	42.643	0.007	0.663	1.068
RIDGE FAB 4G	Reddit	Service	240	16.500	0.010	1.268	3.691
Lenovo A6020a40	Reddit	Service	208	8.808	0.037	6.110	2.746
LML413DL	Reddit	Service	160	18.000	0.016	5.351	1.897
A574BL	Reddit	Service	146	13.700	0.009	1.671	2.033
A0001	Reddit	Service	90	13.533	0.013	2.637	0.004
Aquaris X5 Plus	Reddit	Service	84	12.429	0.021	10.360	2.243
RIDGE FAB 4G	Reddit	Foreground	78	14.167	0.008	3.635	3.760
LG-TP450	Reddit	Service	69	13.000	0.015	6.114	1.611
Redmi 4X	Reddit	Background	20	22.000	0.007	18.639	1.501
Mi 4i	Reddit	Service	15	3.667	0.016	64.106	3.083
LG-K550	Reddit	Service	7	28.000	0.011	15.511	1.733
Redmi Note 3	Reddit	Background	4	1.000	0.030	1 627.130	1.566
ONE A2001	Reddit	Service	3	1.000	0.020	948.024	0.818
MI 5	rif is fun	Service	2	2.000	0.030	635.040	2.544
Redmi Note 3	RedReader	Background	2	1.000	0.010	539.460	0.041

Table A.21: Greenhub: Reddit Query Power Results

Bibliography

[1] Wilke C, Richly S, Götz S, Piechnick C, Aßmann U. Energy Consumption and Efficiency in Mobile Applications: A User Feedback Study. 2013;Available from: https://ieeexplore.ieee.org/abstract/document/6682059.

[2] Heikkinen MV, Nurminen JK, Smura T, Ham H. Energy efficiency of mobile handsets: Measuring user attitudes and behavior. Telematics and Informatics, Volume 29, Issue 4. 2012;p. 387–399. Available from: https://www.sciencedirect.com/science/article/pii/S0736585312000068.

[3] Somavat P, Jadhav S, Namboodiri V. Accounting for the energy consumption of personal computing including portable devices. Publication:e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking. 2010 April;p. 141–149.

[4] GREENSPECTOR. Playstore Efficiency Report 2019;. [Accessed 10 Jun, 2020]. Available from: https://greenspector.com/wp-content/uploads/2020/01/Playstore_Efficiency_Report_2019_EN.pdf.

[5] Paradiso JA, Starner T. Energy scavenging for mobile and wireless electronics. 2005;Available from: https://ieeexplore.ieee.org/document/1401839.

[6] Halpern M, Zhu Y, Reddi VJ. Mobile CPU's Rise to Power: Quantifying the Impact of Generational Mobile CPU Design Trends on Performance, Energy, and User Satisfaction. 2016;Available from: https://ieeexplore.ieee.org/document/7446054.

[7] Android is for everyone;. [Accessed 22 Jan, 2020]. Available from: https://www.android.com/everyone/.

[8] Mobile Operating System Market Share Worldwide; 2019. [Accessed 20 Jun, 2020]. Available from: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[9] Qoitech. How to measure smartphone energy consumption (Samsung S7 Example). 2018 February;[Accessed 22 Jan, 2020]. Available from: https://www.qoitech.com/blog/how-to-measure-smartphone-energy-consumption.

[10] Ding F, Xia F, Zhang W, Zhao X, Ma C. Monitoring Energy Consumption of Smartphones. 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing. 2011 Oct;.

[11] Zhang L, Tiwana B, Dick RP, Qian Z, Mao ZM, Wang Z, et al. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones . 2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). 2010 Oct;.

[12] Noureddine A, Bourdon A, Rouvoy R, Seinturier L. Runtime monitoring of software energy hotspots. 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. 2012 Sept;p. 160–169. Available from: https://ieeexplore.ieee.org/document/6494916.

[13] Linares-Vásquez M, Bavota G, Bernal-Cárdenas C, Oliveto R, Penta MD, Poshyvanyk D. Mining energy-greedy API usage patterns in Android apps: an empirical study. MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories. 2014 May;p. 2–11. Available from: https://dl.acm.org/doi/10.1145/2597073.2597085.

[14] Westfield B, Gopalan A. Orka: A new technique to profile the energy usage of android applications. 2016 5th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS). 2016 April;p. 213–224.

[15] Cornet A, Gopalan A. A Software-based Approach for Source-line Level Energy Estimates and Hardware Usage Accounting on Android. ENERGY 2018, The Eighth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies. 2018 May;p. 32–37.

[16] UI/Application Exerciser Monkey;. [Accessed 22 Jan, 2020]. Available from: https://developer.android.com/studio/test/monkey.

[17] Jamrozik K, Zeller A. DroidMate: A Robust and Extensible Test Generator for Android. In: 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft); 2016. p. 293–294.

[18] monkeyrunner;. [Accessed 22 Jan, 2020]. Available from: https://developer.android.com/studio/test/monkeyrunner.

[19] monkey recorder;. [Accessed 22 Jan, 2020]. Available from: https://android.googlesource.com/platform/sdk/+/ics-mr0/monkeyrunner/scripts/monkey_recorder.py.

[20] UI Automator;. [Accessed 22 Jan, 2020]. Available from: https://developer.android.com/training/testing/ui-automator.

[21] Milano DT. Culebra Tester;. [Accessed 22 Jan, 2020]. Available from: http://culebra.dtmilano.com/.

[22] Energy labels. 2020;[Accessed 22 Jan, 2020]. Available from: https://europa.eu/youreurope/business/product-requirements/labels-markings/energy-labels/index_en.htm.

[23] Understanding The EU's New Energy Efficiency Labels. 2019 March;[Accessed 22 Jan, 2020]. Available from: https://industryeurope.com/understanding-the-eus-new-energy-efficiency-labels/.

[24] Glick K. I/O 2019: New features to help you develop, release, and grow your business on Google Play. 2019;[Accessed 22 Jan, 2020]. Available from: https://android-developers.googleblog.com/2019/05/whats-new-in-play.html.

[25] Goldstein K. Some experimental observations concerning the influence of colors on the function of the organism. 1942;p. 147–151. Available from: https://journals.lww.com/ajpmr/Citation/1942/06000/SOME_EXPERIMENTAL_OBSERVATIONS_CONCERNING_THE.2.aspx.

[26] Stone NJ, English AJ. Task type, posters, and workspace color on mood, satisfaction, and performance. Journal of Environmental Psychology. 1998;p. 175–185. Available from: https://www.sciencedirect.com/science/article/pii/S0272494498900846.

[27] Tsiang D. Developing an Energy Efficiency Rating Framework for Android Applications [master's thesis]; 2019.

[28] Derudder K. GREENSPECTOR App Mark;. [Accessed 10 Jun, 2020]. Available from: https://greenspector.com/en/greenspector-app-mark/y.

[29] Philippot O. What are the best web browsers to use in 2020?; 2019. [Accessed 10 Jun, 2020]. Available from: https://greenspector.com/en/what-are-the-best-web-browsers-to-use-in-2020/.

[30] Leboucq T. Which video conferencing mobile application to reduce your impact?; 2020. [Accessed 10 Jun, 2020]. Available from: https://greenspector.com/en/which-video-conferencing-mobile-application-to-reduce-your-impact/.

[31] Derudder K. What's the carbon impact for social network applications?; 2020. [Accessed 10 Jun, 2020]. Available from: https://greenspector.com/en/social-media-carbon-impact/.

[32] Qoitech. Otii Enterprise;. [Accessed 10 Jun, 2020]. Available from: https://www.qoitech.com/products/enterprise.

[33] Django. Django overview;. [Accessed 10 Jun, 2020]. Available from: https://www.djangoproject.com/start/overview/.

[34] Django. Security in Django;. [Accessed 10 Jun, 2020]. Available from: https://docs.djangoproject.com/en/3.0/topics/security/.

[35] NGINX. What is NGINX?;. [Accessed 10 Jun, 2020]. Available from: https://www.nginx.com/resources/glossary/nginx/.

[36] uWSGI. Setting up Django and your web server with uWSGI and nginx;. [Accessed 10 Jun, 2020]. Available from: https://uwsgi-docs.readthedocs.io/en/latest/tutorials/Django_and_nginx.html.

[37] rq. Django-RQ. GitHub; 2020. [Accessed 10 Jun, 2020]. Available from: https://github.com/rq/django-rq.

[38] Driessen V. Redis Queue;. [Accessed 10 Jun, 2020]. Available from: https://python-rq.org/.

[39] Redis. Introduction to Redis;. [Accessed 10 Jun, 2020]. Available from: https://redis.io/topics/introduction.

[40] team V. Retiring Octane;. [Accessed 12 Jun, 2020]. Available from: https://v8.dev/blog/retiring-octane.

[41] Mozilla. Kraken JavaScript Benchmark;. [Accessed 12 Jun, 2020]. Available from: https://krakenbenchmark.mozilla.org/.

[42] webkit. In-Depth Analysis;. [Accessed 12 Jun, 2020]. Available from: https://browserbench.org/JetStream/in-depth.html.

[43] webkit. About MotionMark 1.1;. [Accessed 12 Jun, 2020]. Available from: https://browserbench.org/JetStream/in-depth.html.

[44] Debouvere B. Wirple;. [Accessed 12 Jun, 2020]. Available from: https://www.wirple.com/.

[45] webkit. About Speedometer 2.0;. [Accessed 12 Jun, 2020]. Available from: https://browserbench.org/Speedometer2.0/.

[46] Basemark. Basemark Web 3.0;. [Accessed 12 Jun, 2020]. Available from: https://web.basemark.com/.