An Empirical Evaluation of the User Interface Energy Consumption of React Native and Flutter

Erik Blokland

***Abstract*—Energy efficiency is a growing area of concern for mobile developers, as good battery life is highly desired by end users of mobile devices. While many developers work to increase their app's energy efficiency during development, there is not much information available about the energy efficiency of the different app frameworks on the market. As the choice of a framework must be made before the start of development, and cannot be easily changed later on, information about these frameworks is crucial to allow developers to optimize their apps for efficiency. In this paper, we compare the energy use of the React Native and Flutter frameworks while performing User Interface tasks to the native Android API. While we were unable to draw a conclusion about whether one of these frameworks is more or less efficient than the baseline app, we were able to identify certain UI actions that were consistently more or power-hungry than average, and found that the energy use tendencies of these actions tended to be consistent between different frameworks and devices. We also found that measuring overall energy use between separate test runs was inconsistent, and further research may be necessary to identify the best method to isolate the energy use of a single app.**

I. Introduction

Energy efficiency is an area of significant importance to mobile developers, as their software is run on devices with limited battery capacity. There are a number of aspects to the energy efficiency of an app, such as the display, sensor use, and network use [1]. Research has been done into many different energy saving approaches, such as optimizing the efficiency of the display [2], as well as devising methods to give programmers a better overview of their app's power consumption [3].

Nearly all app developers will want to release their app on at least Android and iOS, as these two mobile operating systems account for the vast majority of smartphone users. To this end, cross-platform app frameworks, such as React Native, Flutter, and others attempt to ease the burden of creating multi-platform apps by adopting the concept of "Write Once, Run Anywhere" [4].

However, once the framework of choice is decided upon, switching to a different framework requires significant effort, as the framework API and even the programming language may be different. Therefore, the choice of framework is a critical one for developers, but at the same time must be made without prior knowledge of the performance of their particular (envisioned) app on each framework. Empirical studies of the qualities of the available frameworks can help developers make an informed choice on which framework is right for their needs. Willocx et al. [5] compare the performance of a number of frameworks in comparison to each other and a "native" app, using the standard development tools (such as the Android ADK), for a given platform, considering aspects such as the CPU load under a certain task, the time taken to load a page, and the amount of RAM consumed by the app. Angulo and Ferre [6] compared the user experience of two frameworks with a baseline "native" app, comparing the overall satisfaction of the users and which app they preferred to use.

However, in spite of the importance of energy efficiency, there are no studies comparing the energy use of different frameworks. As app frameworks are not easily changed after the development process has begun, this information is critical to developers seeking to optimize the energy efficiency of their app. Additionally, there are many different elements combined to make an app's UI, and choices made in this area could also influence the energy consumption of the app.

In this paper, we measure the energy use of React Native, Flutter, and the Android API when performing UI tasks. We performed these measurements by creating skeleton apps using each of the frameworks that implement a selection of UI elements, and running a test script using MonkeyRunner [7], a testing utility allowing programmatic testing of app UIs through an API supporting all types of gestures. This script exercises each of these elements in turn over a period of time. The energy use was measured by leveraging the built-in Android power monitoring utilities [8], which allowed us to measure the battery charge level at different points of the test script.

We found that there was a clear difference in energy usage between different types of UI action. Furthermore, these differences were consistent between devices and framework, implying that some UI actions inherently cost more energy than others. One of the most consistently energy-expensive actions was scrolling a list, with both text and image contents consuming consistently more energy than average. As scrolling views often form the basis of an app, optimizations to these elements could have a noticeable impact on the energy consumption of real-world apps.

We were unable to determine whether or not a particular framework is more or less efficient than others, as the average energy consumption was inconsistent between different devices and different test runs on the same device. We believe that the inconsistency could be caused by background tasks being run on the device, but were unable to conclusively determine the cause.

II. Background

There are a number of Android app frameworks available for use, and these frameworks use a variety of techniques to display the user interface of an app. We chose the two frameworks considered in this study in part due to the differences in how each displays UI elements. Additionally, each framework and the baseline app use a different programming language, which could lead to further differences in the energy consumption. We have summarized the differences between the app frameworks in the subsections below.

A. Android API

The Android API [9] uses a collection of elements combined by a developer into Activities, which form the basis of an app. Each screen of an app can be defined by an Activity, but can also be created by combining independent fragments [10] into an activity, and these fragments can allow an app to change its UI without transitioning between Activities. Android apps using the Android ADK are written in Java or Kotlin, and executed by the Android Runtime (ART) [11] on all Android versions 5 or higher. ART has been shown by Chen and Zong to be significantly more energy efficient than the older Dalvik Java Virtual Machine used in older versions of Android [12], and all tests of the baseline app will be performed on devices using this version of the JVM.

B. React Native

React Native has been shown to perform equal or better to a native Android app in terms of response time, with near-equal results in memory usage and framerate consistency, but with higher CPU usage [13]. To render the UI, React Native uses "native bridges" to convert UI elements written in JSX to those exposed by the Android API [14]. Unlike the Android API, React Native uses JavaScript for the app logic and user interface, using the JavaScriptCore JS engine, using JIT compilation on our Android test devices [15]. Georgiou, et al. found that JavaScript has a more favorable Energy Delay Product when compared to Java [16].

C. Flutter

Flutter uses the Dart programming language, and application code is Ahead-Of-Time compiled and built into an APK [17] using the Android NDK. In contrast to React Native, Flutter does not use the UI elements built into Android, but instead renders the UI independently of the underlying system in a single view [18]. This core difference could affect the energy efficiency of the app, as inefficiencies in the Android API could be avoided by the independent rendering, or vice versa.

III. Methodology

The goal of this paper is to measure the energy consumption of the React Native and Flutter app frameworks while performing UI tasks and compare them to the native Android API. To this end, we propose the following research questions:

- RQ1: How much energy do different types of UI actions consume?
- RQ2: Is there a difference between the frameworks in terms of energy consumption?

We tested the energy consumption of the different frameworks by implementing a similar app in each framework. We then performed automated testing on each app to exercise the UI elements, and measured the energy consumption of the phone with Android's built-in battery statistic services.

A. Experimental App

1) Feature Selection: In devising the testing app, we needed to decide on a set of UI elements to include. We chose to only include elements that are provided by all of the frameworks in order to simplify the app and make a fair comparison between the app frameworks being tested. We also chose to include (physics-based) animations and different transitions between screens. To determine which elements we would include, we analyzed a selection of commonly used Android apps, and determined which basic UI elements (such as transitions, animations, and lists) were present.

2) Tested Features: We have chosen to test the following features:

- Text Lists
- Image Lists
- Modal Dialogs
- Buttons
- Linear Animations
- Physics-Based Animations
- Navigation Drawer
- Spinner Dialogs

3) Architecture: As the goal of this project was to measure the energy use of the GUI alone, we designed the app to have as little internal logic as possible. To this end, text and images displayed are contained in the app itself, to minimize dependency on loading from external sources. The app consists of different activities reached via the navigation drawer of a start page. Figures 1, 2, and 3 show the home page of each of these apps.

B. Testing Process

We used an automated testing process to exercise the UI elements to ensure that the results would be reliable between runs. The test consists of a set of discrete sections, each section testing a specific part of the UI (e.g. Scrolling with images, physics-based animations, etc.). By separating the tests into sections, we can compare the energy consumption per-activity, and combine the partial results into an overall view of the frameworks' energy consumption.

The tests will be run using Monkeyrunner [7], over a wireless ADB connection on a non-congested 5GHz WiFi network, where the devices are placed in close proximity to the wireless access point. We chose to use Monkeyrunner as it supports scripted touch events at a specific point on the screen, allowing us to write a single test script that works on each app without any modifications. Each app's UI is positioned in exactly the same location to support this method. The test script will periodically record the

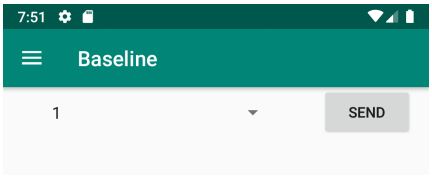

Figure 1. Baseline App Home Page

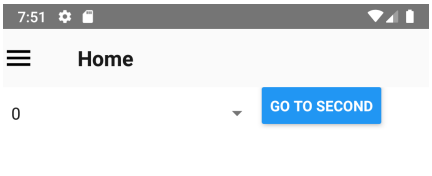

Figure 2. React Native App Home Page

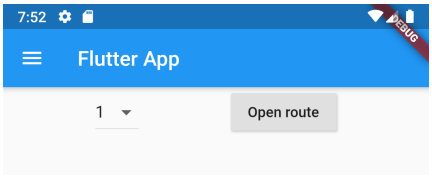

Figure 3. Flutter App Home Page

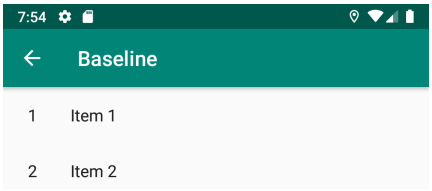

Figure 4. Baseline App Text Scrolling Page

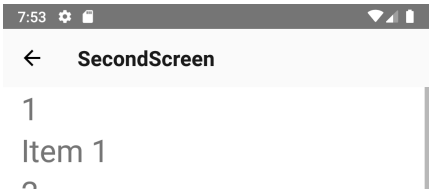

Figure 5. React Native App Text Scrolling Page

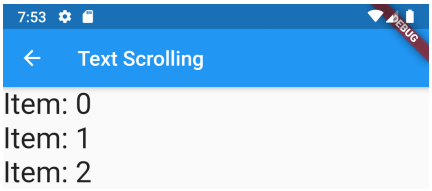

Figure 6. Flutter App Text Scrolling Page

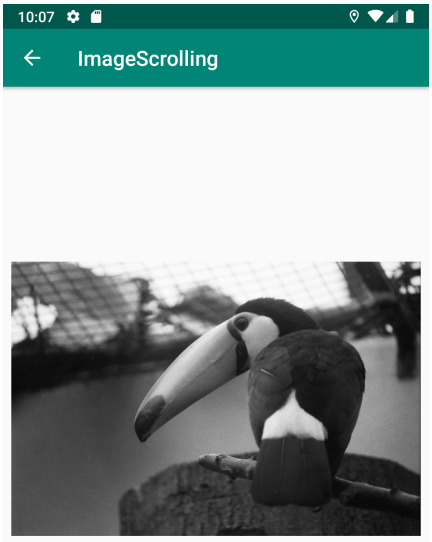

Figure 7. Baseline App Image Scrolling Page

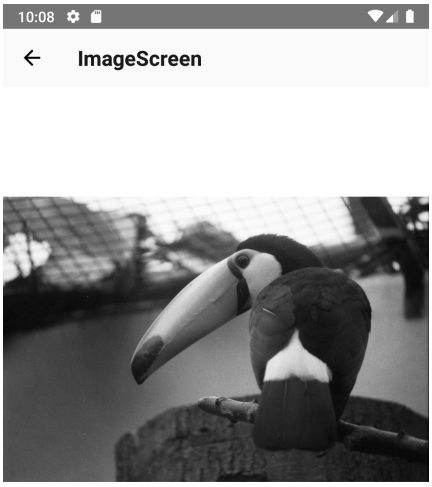

Figure 8. React Native App Image Scrolling Page

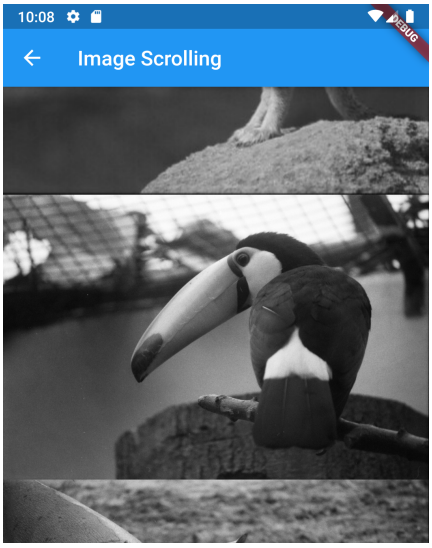

Figure 9. Flutter App Image Scrolling Page

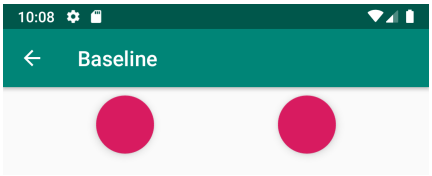

Figure 10. Baseline App Animation Page

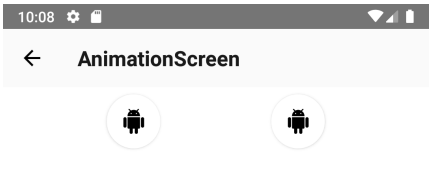

Figure 11. React Native App Animation Page

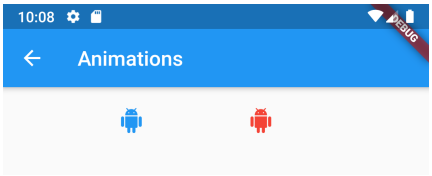

Figure 12. Flutter App Animation Page

battery's charge level by first scheduling an update using the command `service call batteryproperties 4`, and then poll the battery statistics using the command `dumpsys battery`. This method allows us to retrieve the battery charge level at a resolution of about one second on our two Android 9 devices. However, we were unable to achieve this level of accuracy on the Android 8 device (Motorola G5 Plus), which lead to the measurements of individual sections being less precise, as displayed in Figures 23, 24, and 25 in Appendix B. This effect is caused by the shorter run-time of individual sections when compared to that of the whole test. Due to battery constraints, it was not feasible to lengthen the run-time of individual sections as the run-time of the overall test would exceed the battery life of some devices.

C. Energy Consumption Data Collection

Di Nucci, et al. have shown that software monitoring of power consumption on Android is feasible, with a small variation from hardware monitoring [3]. We will use Android's built in battery monitoring service, which provides us with the battery charge level in μAh [8]. Although the recorded power consumption includes energy used by all components of the device, we will be able to compare power between

apps on the same hardware, as we expect the power draw from hardware not being studied (such as the display itself) to remain consistent between different frameworks.

1) Testing Protocol: Each device used is first factory reset, to prevent user-installed apps from affecting the power consumption in an unpredictable way. Each phone is running the latest (at the time of testing) official Android OS version available from their respective manufacturers, as listed below. During the test, unnecessary services such as Bluetooth and GPS were disabled to limit outside influence on energy consumption, screen brightness was manually set to 50%, and background processes were disabled in the developer options. The current battery charge level is measured at the beginning of each run of the test. As the Android energy monitoring utilities can only be run while on battery power, we chose to use wireless ADB in spite of its potential confounding effects. We made this choice as some of the available testing devices cannot be 'rooted' to provide system level access, which is a requirement to disable charging while using ADB over USB. We expect the power usage of the WiFi module to be consistent between runs on a single device, and do not intend to directly compare results between different devices.

The test script is run 20 times per app, per device. These runs occur sequentially, without any pause in between runs. The first 10 runs of the script are intended as a 'warm-up' to allow run-time optimizations to occur without influencing the measurements, and are therefore not included in the final results.

2) Testing Devices: We tested with a small selection of devices to verify differences measured between apps across different Android versions and device hardware. By testing with multiple devices, we reduce the chance that the outcome of our study will be influenced by the particular hardware of a single device.

We used the following phones in our study:

Device	OS Version
Nokia 7 plus	Android 9, April 2019 security patch
Google Pixel XL	Android 9, May 2019 security patch
Motorola G5 Plus	Android 8, February 2019 security patch

D. Data Analysis

For each run, the energy consumption is found by comparing the charge level in μAh of the battery at the beginning and end of the test. The 10 runs are then averaged together, and compared to the other two apps by way of a Wilcoxon T test. The individual sections are analyzed by comparing their average energy usage relative to the overall average for that app-device combination. This delta can then be compared to other apps on that device, using a Wilcoxon T test, and to other devices.

IV. Results

A. RQ1: Is there a difference between the frameworks in terms of energy consumption?

To answer this research question, we recorded the charge level (in μAh) of the device battery at the beginning and end of each run of the test, as well as a timestamp allowing us to precisely measure the amount of charge that was consumed by the device to execute a run.

Each device tested resulted in a different outcome for the efficiency ranking of the different apps, and in most cases, except for the tests run on the Motorola device, these results were statistically significant at α=0.05 using a Wilcoxon test. However, as each device tested resulted in a different outcome for each app, we are unable to draw significant conclusions as to whether or not a particular framework performs better or worse than another framework or the baseline. While there may be differences between the apps, these differences appear to be largely outweighed by outside factors, potentially background tasks running on the device during testing or variable hardware consumption (such as the wireless module).

The individual results per device are reported in the following sections:

1) Google Pixel XL: As shown in Figure 13, we found that the baseline app ran far more efficiently on the Pixel device than either of the frameworks, using only 64% of the energy of the next lowest app, Flutter. This difference would be clearly noticeable to the user, and, if caused by the app itself, would give the baseline Android API a clear energy consumption advantage. However, we did not record similar performance on other test devices, where the different apps had a more similar overall result, indicating potential outside interference in the results.

2) Nokia 7 Plus: In contrast to the Pixel XL, the results obtained from the Nokia 7 Plus show that the React Native app used the least energy, and the Baseline app used the most. However, these results are much closer than those from the Pixel, with the most power-hungry app only consuming an average of approximately 13% more energy than the least power-hungry.

3) Motorola G5 Plus: The results obtained from the Motorola G5 Plus are the most consistent out of the three devices tested with respect to the difference between frameworks, with the most efficient framework (Flutter) using only approximately an average of 8% more energy during the test than the least efficient (React Native).

RQ1 summary: Due to variations between different runs of our test, we were unable to definitively answer RQ1. Each app placed in each position on one of the three devices, and all of these differences were statistically significant. These results could indicate that device hardware plays a greater role than software for energy use, but could also indicate an unaccounted for source of interference.

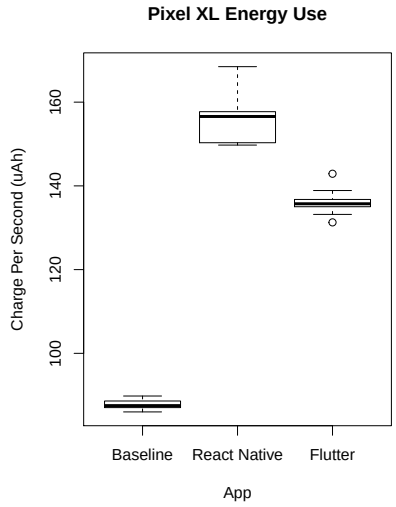

Figure 13. Energy use on a Pixel XL

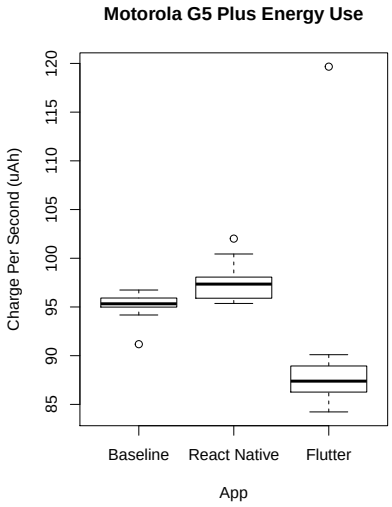

Figure 14. Energy use on a Motorola G5 Plus

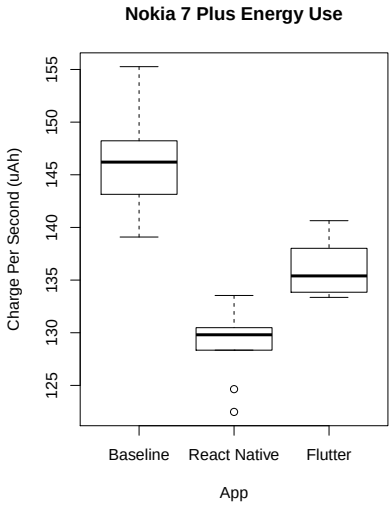

Figure 15. Energy use on a Nokia 7 Plus

B. RQ2: How much energy do different types of UI actions consume?

To answer this research question, we separated the test into seven distinct sections, and collected time and battery charge level data for each section. This approach allows us to compare the energy usage of different app sections relative to the overall use of the app, and compare these relative differences to the other apps and devices. Due to the lower accuracy of measurements on the Motorola G5 Plus, its data has not been included in this section.

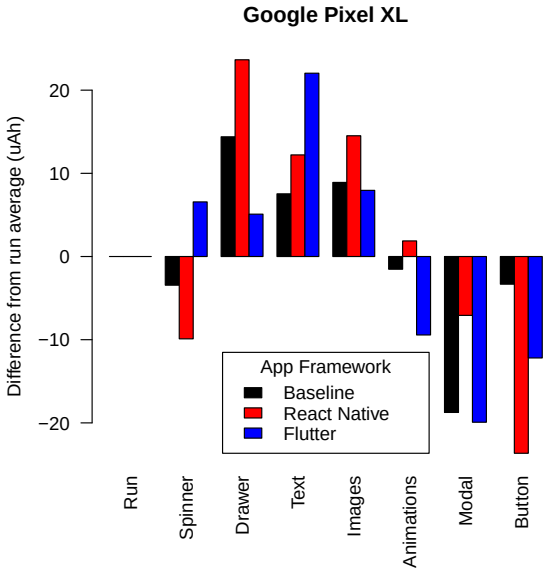

Figure 16. Relative energy use of app sections on a Pixel XL

We found that there were evident trends regarding the power usage of different parts of the app. In the React Native and baseline apps, the action of opening and closing the navigation drawer was, in all but one test run, the most demanding section of the test, and in eight out of the nine tested phone/app combinations, more power-hungry than the overall average. Image scrolling also consumed more energy than average in eight out of nine cases, and recorded the highest out of all sections on two of those cases. Figures 17, 19, 18, 20, 21, and 22, in Appendix B, show these trends on the two phones, a Google Pixel XL and Nokia 7 Plus, where these trends could be reliably identified.

Notably, while text scrolling was typically above the average in all apps, it was the highest consuming section of the Flutter app on both the Pixel XL and Nokia test devices, by a considerable margin. This result may be related to noticeable scrolling "stutter" observed while manually testing the app that did not occur in either of the other apps, indicating a potential issue with the framework's underlying implementation. This result can be observed in Figure 16, which shows the relative results of each section of each app collected from the Pixel test device.

The most consistently energy efficient test section was the button test, which performed more energy efficiently than average in seven of the nine tests, five of which were significant at an α=0.05 level using a Wilcoxon T test. The data collected from the Motorola device was too noisy to provide a meaningful measurement in this regard, and contributed both of the two instances of above average energy use. Additionally, the modal dialog, animation, and spinner menu sections also used consistently less energy than average, in eight, seven, and eight out of the nine test combinations, respectively. However, in only two cases, or three for the animation section, were these results significant, indicating that these sections may not have as strong of an effect on the energy use of the app as the more energy-heavy sections, as well as the consistently lower button section.

Additionally, we compared the different apps to each other

by measuring the difference in energy use of each section from the overall test average. By comparing the results in this manner, we noticed two clear trends. Firstly, the text scrolling sections of the Flutter app had a larger delta than the other two apps in all cases, while the image scrolling sections of the Flutter app had a lower delta. Secondly, the button section of the React Native app had a smaller delta than the other two apps in all cases. Furthermore, there were no instances in which a section of an app used consistently more energy than the overall average while using less energy than average in a different framework. We therefore conclude that the energy use of a UI element, relative to the overall energy use of its app, is not dependent on the UI framework used.

RQ2 summary: We identified clear trends regarding the energy use of specific app sections relative to the overall mean, with the navigation drawer, text scrolling, and image scrolling consistently using more energy than average. The button section consistently used less energy than average, as did the animations, modal dialogs, and spinner, though with less significance than the button section. Furthermore, these trends generally remained consistent between different devices and app frameworks, suggesting that some UI elements inherently consume more energy than others.

V. Discussion and Related Work

A. Energy Use of UI Elements

Our results show that there is a link between the type of UI element being exercised and the energy use of the device. Some of the high energy use sections, such as the navigation drawer, are only occasionally used during user interaction with a real app, so optimizations made here may not greatly affect the overall power consumption of an app. Other sections, such as text and image scrolling, are more frequently used, and in many cases form the basis of the app UI. Considering that these sections were consistently over-average in terms of energy usage in our tests, app developers could benefit in the future from optimizations made to these elements, or by choosing a framework that implements these elements in a more efficient manner than another.

B. Measurement Interference

During our initial trials, we noticed that some apps used far more energy than others on a particular device, but not on other devices. When running additional trials to confirm the results, the outcome was much more similar to that of the other apps. We concluded that an outside factor was influencing our results, and attempted to mitigate this with a 'warm-up' round of tests, but we were unable to fully remove this influence. During testing, we disabled Bluetooth, GPS, adaptive brightness, and disabled background processes in the Android developer options. While WiFi is a potential source of the observed interference, we do not believe it to be the cause, as all tests were performed in a similar environment, connected at short range to a non-congested network. Notably, measurements taken from the same test run were generally consistent, even within the 20-run tests over a three hour time span. We hypothesize that the interference comes from either background tasks running on the device that may have begun during the charge cycle between trials, or from inconsistent scheduling choices made by the device, such as whether to run the app on the 'big' or 'little' cores of the CPU. Some studies, such as that done by DiNucci et al, [3], collect more information about the system state and resource utilization, and model the power usage of an app based off of that information. In contrast, our study solely considers the battery charge level reported by the system. When comparing results between multiple apps, it may also be useful to reduce the time between these trials, as our observations were that individual trials in a set of runs produced relatively consistent results. In future studies, it may be useful to interleave trials, such that instead of testing a single app before charging the phone, a portion of each app's total trials are run between each charging cycle. However, this method does not address the underlying cause of the interference, and may not produce consistent results between cycles.

C. Related Work

There have been a number of studies over the energy consumption of mobile devices and apps, as well as comparisons between native apps and app frameworks. Some studies comparing mobile frameworks focus on user interaction compared to native apps [6], while others focus on performance aspects or a mixture of both [19][13]. There are a number of studies on how users value battery life [20][21], showing the importance of energy efficiency in apps. Many studies have been done over energy efficiency of mobile apps, including the following:

Li et al. [1] analyzed the power usage of a number of publicly available Android apps, identifying the amount of energy consumed by different types of apps, the energy consumed by different smartphone components, and what parts of an app consume the most energy. They also identified the granularity of measurement needed to accurately measure method and API calls made by an app.

Chen and Zong [12] developed an energy profiler to measure the energy consumption of android apps, and tested the impact of programming language choice, using C, C++, and Java, on energy efficiency. They also evaluated the differences between the ART and Dalvik virtual machines, and measured the impact of different implementation choices.

Ma et al. [19] compare multiple web apps to native apps in multiple aspects, including energy consumption. They found that the energy use of web apps was often close to that of native apps if the web cache was enabled, and that they were sometimes more efficient at certain tasks than the native counterparts. However, this was often attributable to differences in the operation of the app, such as the communication protocol or background tasks performed by the native app, rather than a fundamental difference between web and native apps.

Linares-Vásquez et al. [2] investigated the energy consumption of app UIs in terms of the display hardware. They created a tool, GEMMA, that leverages the property of Organic Light Emitting Diode displays where the power consumption of the display is coupled to the colors displayed; some colors consume less power than others, with a pixel displaying true black consuming no power. They found that, for this type of display, significant energy use reduction can be achieved while keeping the end user experience acceptable.

Hansson and Vidhall [13] studied the React Native framework in-depth, touching on both performance and user experience. Although they did not study the energy efficiency of the framework, they did collect data on the CPU utilization while performing various tasks, and higher CPU usage often indicates higher energy consumption. They found that while React Native typically had equal or very close performance to that of the Android and iOS native apps, it generally utilized the CPU more than the comparable native app. This could indicate that React Native will use more energy than a native app to perform the same function, but unfortunately we were unable to answer this question due to the aforementioned shortcomings in our testing methodology.

DiNucci et al. [3] created a tool that analyzes the energy use of Android apps at the method level, allowing developers to identify areas of high energy consumption, without the need for special hardware instrumentation. Their work showed that their software estimation tool, using the information provided by Android's built-in power monitoring utilities, provides an accurate estimation of energy consumption when compared to hardware instrumentation. As the authors showed that Android's utilities provide adequately accurate estimations, we chose to use them in our study, to reduce the complexity caused by hardware instrumentation. However, we did not analyze the same system information, instead relying solely on the battery charge information provided by the Android battery service.

VI. Threats to Validity

A. Internal Validity

1) Measuring Technique: When measuring the charge level of a device during testing, the `scheduleUpdate` function of the `batteryproperties` android service [22] was called prior to reading the battery level. On the two Android 9 devices, this method yielded a battery level that was accurate to about a second, providing a reasonably accurate measure of the true battery level. However, on the Android 8 device, the battery charge level did not update immediately after calling `scheduleUpdate`, meaning that the error present in the individual section measurements is much higher than that of the other two devices.

However, this error is much smaller in the overall measurement of the app's energy use, as the longer running time of the full test overshadowed the update time of the battery charge level in comparison to the individual section measurements. As stated in our results, we did not take the results from the Android 8 (Motorola) testing device into account when answering research questions relating to individual sections of the app, so we do not expect this to influence our results.

2) Unexpected Outliers: We initially used a shorter experimental setup in which the test script was run only 10 times per app, without "warm-up" runs prior to the 10 runs of interest. When analyzing our results, we noticed that some sets of runs had an abnormally high energy use compared to the other runs on the same device, higher than would be expected to be attributable to the difference between app frameworks. Upon re-running the tests for that device/app combination, the outcome was more in line with that of the other two apps on that device. To reduce the chances of this effect occurring, we added 10 "warm-up" runs to our testing method. These runs were intended to reduce the variation that could potentially occur from background tasks that were running prior to the test (as many phones required charging between tests) and run-time optimizations.

B. External Validity

1) Generalization to other devices: In this study, we do not attempt to normalize the energy consumption of different devices to remove the influence of hardware and software variations. However, the differences we found between UI elements were consistent between the two devices we were able to gather sufficient data from. We expect these differences to remain consistent if tested on other devices.

Both the baseline and React Native apps rely on the built-in Android API UI elements to render their UI. As the implementation of these elements is not necessarily consistent between Android versions, we cannot say with certainty that our results will be applicable to older versions of Android.

2) Generalization to Other Apps: In each of the apps used in this paper, standard UI elements from the frameworks' APIs were used. Therefore, we expect our results to be valid for real-world apps. However, the energy consumption of the UI of an app is only a component of an app's overall energy consumption, and the additional functions that a typical app would include, such as internet access, may overshadow the energy consumption of the UI and could lead to different results than found by our study.

VII. Conclusions and Future Work

In this paper, we investigated the power use of React Native, Flutter, and the Android API when rendering UI elements. We focused on three primary research questions: Whether there is a difference between the three apps in terms of overall energy consumption (RQ1), how much energy do different types of UI actions consume (RQ2), and if there is a difference between UI action energy consumption, whether it is consistent between different types of UI action. (RQ3) To test these questions, we constructed three apps, using the native Android app as our baseline, and wrote a testing script using monkeyrunner [7] to exercise each of the studied UI actions individually. We found that there is a difference in energy consumption between different UI actions, with some actions, such as opening and closing the navigation drawer, using consistently

more energy than other actions, such as opening and closing a modal dialog. Our findings here lead us to conclude for RQ2 that there is indeed a difference in energy consumption between UI actions. Furthermore, these results were, for the most part, consistent between app frameworks (with limited exceptions as mentioned in our conclusions), answering RQ3. For RQ1, we were unable to reach any conclusions over differences in energy consumption between the frameworks, as each device resulted in different test outcomes, often with very wide margins between different apps. Furthermore, re-testing an app would often result an entirely different outcome. From these results, we concluded that our testing methodology did not do enough to reduce the impact of outside influence on the energy consumption results. However, most results were consistent during a test run, indicating that charging the phone between testing different apps could be a source of interference (for example, by activating background tasks). A possible solution to this problem would be to interleave apps instead of only testing one app per charge cycle, by programmatically switching apps at pre-defined points in the test.

Future work could examine more aspects of app frameworks than only the UI rendering. As each framework tested in this paper uses a different native programming language, and interacts with the underlying Android API differently, energy efficiency differences could exist in all aspects of an app. Investigation into the cause of the measurement inconsistency noticed in our results could also provide useful information for accurately measuring and comparing the energy use of different apps, or different versions of an app.

REFERENCES

[1] D. Li, S. Hao, J. Gui, and W. G. Halfond, "An empirical study of the energy consumption of android applications," in *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 121–130, IEEE, 2014.

[2] M. Linares-Vásquez, G. Bavota, C. E. B. Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Optimizing energy consumption of guis in android apps: a multi-objective approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 143–154, ACM, 2015.

[3] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Petra: A software-based tool for estimating the energy profile of android applications," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, IEEE, May 2017.

[4] M. Q. Huynh, P. Ghimire, and D. Truong, "Hybrid app approach: could it mark the end of native app domination?," *Issues in Informing Science and Information Technology*, vol. 14, pp. 049–065, 2017.

[5] M. Willocx, J. Vossaert, and V. Naessens, "Comparing performance parameters of mobile app development strategies," in *Proceedings of the International Workshop on Mobile Software Engineering and Systems - MOBILESoft '16*, ACM Press, 2016.

[6] E. Angulo and X. Ferre, "A case study on cross-platform development frameworks for mobile applications and ux," in *Proceedings of the XV International Conference on Human Computer Interaction*, p. 27, ACM, 2014.

[7] "Monkeyrunner." https://developer.android.com/studio/test/monkeyrunner/.

[8] https://developer.android.com/studio/command-line/dumpsys.html.

[9] https://developer.android.com/reference.

[10] https://developer.android.com/guide/components/fragments.

[11] https://source.android.com/devices/tech/dalvik.

[12] X. Chen and Z. Zong, "Android app energy efficiency: The impact of language, runtime, compiler, and implementation," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, IEEE, Oct 2016.

[13] N. Hansson and T. Vidhall, "Effects on performance and usability for cross-platform application development using react native," Master's thesis, Linköping University, Human-Centered systems, 2016.

[14] T. Majchrzak and T.-M. Grønli, "Comprehensive analysis of innovative cross-platform app development frameworks," in *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

[15] https://facebook.github.io/react-native/docs/javascript-environment.

[16] S. Georgiou, M. Kechagia, P. Louridas, and D. Spinellis, "What are your programming language's energy-delay implications?," in *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, (New York, NY, USA), pp. 303–313, ACM, 2018.

[17] https://flutter.dev/docs/resources/faq.

[18] https://medium.com/@kelvinma/exploring-android-ui-frameworks-f4b87ff81326).

[19] Y. Ma, X. Liu, Y. Liu, Y. Liu, and G. Huang, "A tale of two fashions: An empirical study on the performance of native apps and web apps on android," *IEEE Transactions on Mobile Computing*, vol. 17, p. 990–1003, May 2018.

[20] M. V. Heikkinen, J. K. Nurminen, T. Smura, and H. Hämmäinen, "Energy efficiency of mobile handsets: Measuring user attitudes and behavior," *Telematics and Informatics*, vol. 29, p. 387–399, Nov 2012.

[21] S. Hosio, D. Ferreira, J. Goncalves, N. van Berkel, C. Luo, M. Ahmed, H. Flores, and V. Kostakos, "Monetary assessment of battery life on smartphones," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, ACM Press, 2016.

[22] https://android.googlesource.com/platform/frameworks/base/+/refs/tags/android-vts-9.0_r9/core/java/android/os/IBatteryPropertiesRegistrar.aidl.

APPENDIX A
EXPERIMENTATION RESOURCES

1) Testing Script: The Python script used with MonkeyRunner to perform the tests in this paper has been made available at http://doi.org/10.5281/zenodo.3260257.

The testing script consists of two Python files, one of which contains the test logic itself, and another definitions file that contains the methods used to interact with individual parts of the app. This system is necessary to handle potential differences in the UI layout between different devices; only the definitions file must be changed in order to run the test on a different device. (Alternatively, a new definitions file can be created, and the import in the main testing script changed).

To run the test script, a suitable Java runtime environment must be installed (in this paper, Java 8 was used). Navigate to the Android SDK tools directory, and launch the script using the instructions found in the `readme.txt` file included with the testing scripts.

2) Apps: Each of the apps used in this paper have been included in the above release. To compile an app, import the project into Android Studio if applicable (Flutter, Baseline) and then proceed with the standard compilation procedure for the framework. The framework (aside from the baseline app) and an appropriate Android SDK must be installed. More detailed instructions can be found in the readme files included with each of the three apps.

APPENDIX B
ADDITIONAL GRAPHS

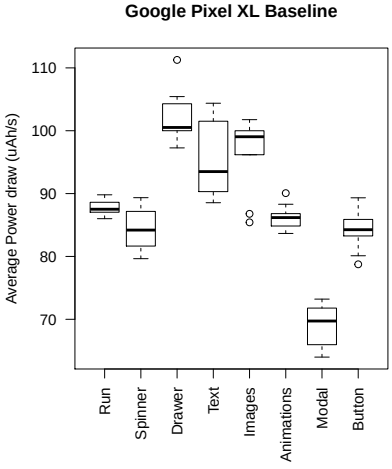

Figure 17. Energy use of the Baseline app

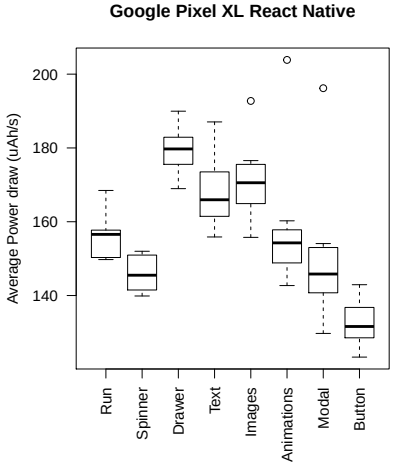

Figure 18. Energy use of the React Native app

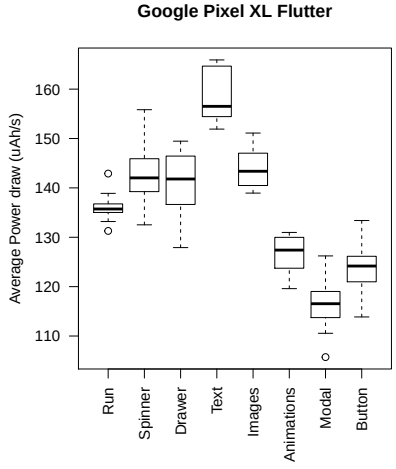

Figure 19. Energy use of the Flutter app

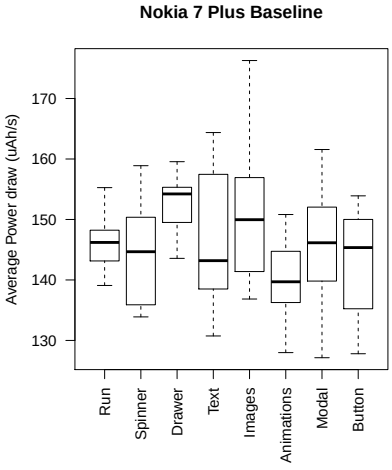

Figure 20. Energy use of the Baseline app

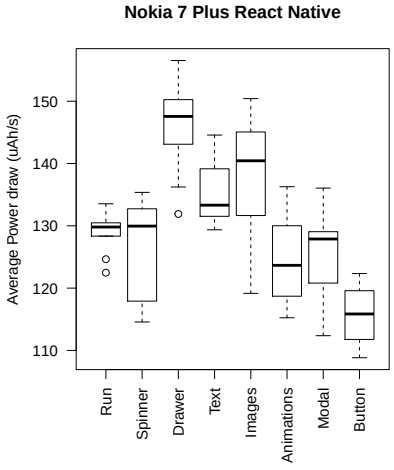

Figure 21. Energy use of the React Native app

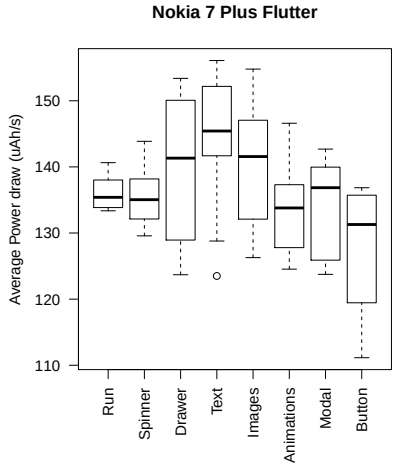

Figure 22. Energy use of the Flutter app

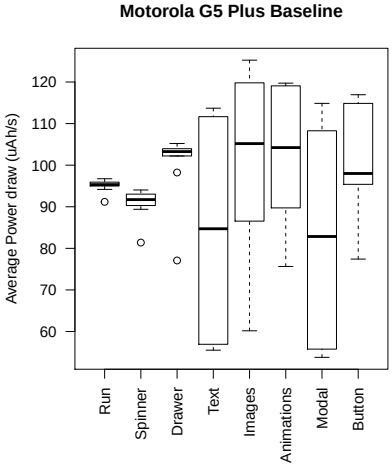

Figure 23. Energy use of the Baseline app

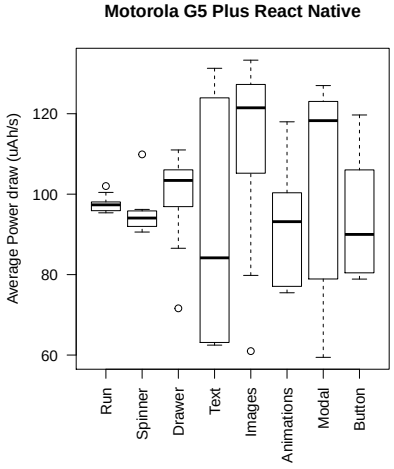

Figure 24. Energy use of the React Native app

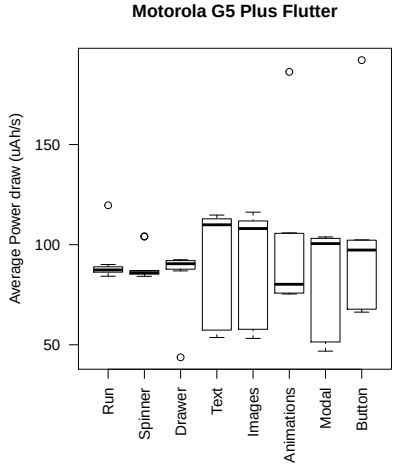

Figure 25. Energy use of the Flutter app

APPENDIX C
DATA TABLES

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Minimum	86.01	79.65	97.27	88.55	96.18	83.66	63.98	80.11
Lower Quartile	87.03	81.65	100.01	90.31	96.18	84.84	65.96	83.26
Median	87.51	84.19	100.52	93.50	99.04	86.18	69.73	84.26
Upper Quartile	88.62	87.17	104.29	101.52	100.00	86.81	71.78	85.89
Maximum	89.81	89.35	105.43	104.38	101.76	88.30	73.21	89.33

Table I
PIXEL BASELINE (μAH/S)

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Minimum	149.74	139.87	168.97	155.87	155.76	142.70	129.73	123.31
Lower Quartile	150.31	141.50	175.56	161.47	164.92	148.84	140.74	128.54
Median	156.57	145.50	179.73	165.95	170.56	154.27	145.82	131.61
Upper Quartile	157.72	150.96	182.90	173.51	175.54	157.81	153.01	136.77
Maximum	168.48	152.00	189.96	187.05	176.57	160.25	154.08	142.93

Table II
PIXEL REACT NATIVE (μAH/S)

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Minimum	133.19	132.52	127.91	151.90	138.94	119.60	110.55	113.86
Lower Quartile	135.02	139.24	136.64	154.43	140.48	123.73	113.73	120.98
Median	135.72	142.02	141.80	156.51	143.37	127.40	116.54	124.17
Upper Quartile	136.76	145.91	146.44	164.66	147.03	129.98	119.02	126.15
Maximum	138.89	155.83	149.46	165.92	151.10	130.98	126.22	133.39

Table III
PIXEL FLUTTER (μAH/S)

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Minimum	139.09	133.89	143.57	130.74	136.85	127.99	127.14	127.80
Lower Quartile	143.14	135.89	149.51	138.50	141.39	136.26	139.81	135.24
Median	146.20	144.67	154.22	143.19	149.97	139.70	146.15	145.35
Upper Quartile	148.23	150.37	155.32	157.46	156.92	144.74	152.04	150.01
Maximum	155.27	158.89	159.56	164.37	176.28	150.82	161.57	153.91

Table IV
NOKIA BASELINE (μAH/S)

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Minimum	128.35	114.57	136.23	129.36	119.16	115.24	112.36	108.83
Lower Quartile	128.35	117.91	143.10	131.53	131.65	118.70	120.81	111.76
Median	129.81	129.96	147.56	133.31	140.44	123.65	127.89	115.85
Upper Quartile	130.48	132.73	150.26	139.15	145.06	130.01	129.06	119.58
Maximum	133.54	135.37	156.53	144.57	150.42	136.28	136.06	122.34

Table V
NOKIA REACT NATIVE (μAH/S)

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Minimum	133.36	129.57	123.69	128.79	126.28	124.54	123.75	111.14
Lower Quartile	133.85	132.12	128.93	141.68	132.10	127.79	125.90	119.45
Median	135.40	135.04	141.32	145.43	141.56	133.79	136.86	131.29
Upper Quartile	138.02	138.18	150.08	152.18	147.06	137.31	139.96	135.72
Maximum	140.64	143.87	153.39	156.09	154.79	146.60	142.69	136.85

Table VI
NOKIA FLUTTER (μAH/S)

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Minimum	94.18	89.39	102.22	55.51	60.18	75.64	53.79	77.42
Lower Quartile	94.99	90.30	102.22	56.93	86.54	89.72	55.77	95.39
Median	95.33	91.72	103.26	84.71	105.18	104.23	82.85	98.00
Upper Quartile	95.92	93.03	103.96	111.66	119.80	119.07	108.27	114.86
Maximum	96.74	94.04	105.22	113.70	125.25	119.73	114.86	116.94

Table VII
MOTOROLA BASELINE (μAH/S)

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Minimum	95.36	90.58	86.55	62.47	79.81	75.49	59.43	78.88
Lower Quartile	95.90	91.98	96.90	63.13	105.21	77.08	78.91	80.44
Median	97.35	94.06	103.41	84.17	121.47	93.16	118.28	89.99
Upper Quartile	98.07	95.84	106.05	123.95	127.28	100.35	123.06	106.04
Maximum	100.44	96.21	110.99	131.29	133.32	117.99	127.00	119.69

Table VIII
MOTOROLA REACT NATIVE (μAH/S)

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Minimum	84.23	84.17	86.90	53.63	53.17	75.37	46.81	66.30
Lower Quartile	86.26	85.30	87.76	57.33	57.74	75.82	51.41	67.78
Median	87.39	86.00	90.48	109.92	108.06	80.22	100.53	97.31
Upper Quartile	88.94	87.07	92.03	112.91	111.83	105.66	103.14	102.24
Maximum	90.10	87.07	92.49	114.77	116.21	105.88	103.86	102.39

Table IX
MOTOROLA FLUTTER (μAH/S)

	Baseline	React Native	Flutter
Minimum	86.01	149.74	133.19
Lower Quartile	87.03	150.31	135.02
Median	87.51	156.57	135.72
Upper Quartile	88.62	157.72	136.76
Maximum	89.81	168.48	138.89

Table X
PIXEL OVERALL (μAH/S)

	Baseline	React Native	Flutter
Minimum	139.09	128.35	133.36
Lower Quartile	143.14	128.35	133.85
Median	146.20	129.81	135.40
Upper Quartile	148.23	130.48	138.02
Maximum	155.27	133.54	140.64

Table XI
NOKIA OVERALL (μAH/S)

	Baseline	React Native	Flutter
Minimum	94.18	95.36	84.23
Lower Quartile	94.99	95.90	86.26
Median	95.33	97.35	87.39
Upper Quartile	95.92	98.07	88.94
Maximum	96.74	100.44	90.10

Table XII
MOTOROLA OVERALL (μAH/S)

	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Pixel Baseline	1.85E-02	1.08E-05	7.58E-05	1.47E-02	4.33E-02	1.08E-05	1.15E-02
Pixel React	1.50E-03	1.08E-05	2.09E-03	7.25E-04	6.31E-01	8.93E-03	1.08E-05
Pixel Flutter	6.84E-03	6.30E-02	1.08E-05	2.06E-04	1.08E-05	1.08E-05	4.33E-05
Nokia Baseline	6.31E-01	6.84E-03	6.31E-01	2.18E-01	6.30E-02	1.00E+00	7.39E-01
Nokia React	9.71E-01	2.17E-05	6.84E-03	2.88E-02	8.92E-02	1.65E-01	1.08E-05
Nokia Flutter	7.96E-01	5.79E-01	2.32E-02	4.81E-01	3.53E-01	9.12E-01	2.88E-02
Moto Baseline	4.87E-04	1.50E-03	1.43E-01	9.71E-01	4.81E-01	1.43E-01	8.92E-02
Moto React	6.84E-03	1.05E-01	3.93E-01	2.32E-02	5.29E-01	1.65E-01	6.84E-01
Moto Flutter	1.65E-01	1.43E-01	7.96E-01	7.96E-01	3.53E-01	3.53E-01	7.39E-01

Table XIII
WILCOXON P-VALUES

	Run	Spinner	Drawer	Text Scrolling	Image Scrolling	Animations	Modals	Button
Pixel Baseline	87.72	84.28	102.11	95.26	96.63	86.20	68.98	84.40
Pixel React	156.02	146.12	179.67	168.23	170.54	157.89	148.94	132.36
Pixel Flutter	136.10	142.66	141.19	158.13	144.05	126.66	116.17	123.90
Nokia Baseline	145.97	144.28	152.89	146.25	152.62	140.00	145.62	143.22
Nokia React	128.99	126.30	145.81	134.93	138.03	124.26	125.36	115.61
Nokia Flutter	136.04	135.71	139.50	144.01	140.29	133.85	133.92	127.73
Moto Baseline	95.12	90.85	100.38	84.62	99.33	100.40	82.52	100.72
Moto React	97.70	95.16	99.05	91.76	111.34	91.14	104.98	94.91
Moto Flutter	90.50	89.34	85.52	89.81	89.33	96.44	85.72	96.03

Table XIV
SECTION AND RUN MEANS FOR EACH APP/DEVICE COMBINATION (μAH/S)