On the Impact of the Critical CSS Technique on the Performance and Energy Consumption of Mobile Browsers

Kalle Janssen, Tim Pelle, Lucas de Geus, Reinier van der Gronden
Tanjina Islam, Ivano Malavolta
Vrije Universiteit Amsterdam, The Netherlands
{ k6.janssen | t.pelle | l.j.de.geus | r.m.vander.gronden }@student.vu.nl, { t.islam | i.malavolta }@vu.nl

ABSTRACT

Context. Due to the growing popularity of smartphones, mobile web browsing is more popular than ever with users desiring fast loading web apps and low energy usage. A technique that might improve the run-time performance and reduce the energy consumption of this action is the Critical CSS technique.
Goal. The goal of this research is to analyze the impact of the Critical CSS technique for the purpose of evaluating the impact on run-time performance and energy consumption from the point of view of a developer in the context of Android mobile web apps.
Method. To assess the impact of the Critical CSS technique, 40 web apps were served with and without Critical CSS on a mobile Android device. For each website, the energy consumption, load time, first paint and, first contentful paint were measured.
Results. Applying the Critical CSS technique had a medium effect size on the first paint for Google Chrome, and on the first contentful paint for Google Chrome and Mozilla Firefox, the effect size is small. Therefore, we can claim that applying critical CSS to web apps served to Android mobile devices has a small but positive effect on their run-time performance. The loading time difference for Google Chrome was small to negligible. Finally, the energy consumption for Google Chrome and Mozilla Firefox, and the loading time and first paint for Mozilla Firefox showed no significant differences.
Conclusions. Depending on the characteristics of the web application, it is advisable to apply the Critical CSS technique to enhance the run-time performance (*e.g.,* to ensure a fast loading time of the web app) of Android mobile web apps. Moreover, the experimental results show that applying the Critical CSS technique tends to have no significant impact on the energy consumption of mobile web apps on Android.

ACM Reference Format:
Kalle Janssen, Tim Pelle, Lucas de Geus, Reinier van der Gronden and Tanjina Islam, Ivano Malavolta. 2022. On the Impact of the Critical CSS Technique on the Performance and Energy Consumption of Mobile Browsers . In *The International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE 2022), June 13–15, 2022, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3530019.3530033

EASE 2022, June 13–15, 2022, Gothenburg, Sweden
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9613-4/22/06…$15.00
https://doi.org/10.1145/3530019.3530033

1 INTRODUCTION

Using a smartphone has become a popular way of surfing the web. As of August 2021, mobile browsing has exceeded desktop browsing by 16.64 percentage points[1]. Over time, web apps have increasingly become more sophisticated. Many have added more value by supporting functionalities and content types such as videos, maps, and interactive elements. This is great for the usability and user experience, however, it does come at a price. Loading the full content of a requested web app adds overhead to all parties involved in this process, from the end-user's device to the router, and backend servers. All these processes require electricity and processing power and therefore tend to have an impact on their sustainability [1] and run-time performance [2]. While the content of web apps increases it is also the case that the amount of users is growing, and will continue to do so[2]. Scaling all the available content to satisfy these new consumers also affects the amount of electricity and computation power required by the user's smartphone as well as the infrastructure that enables the user to load a web app.

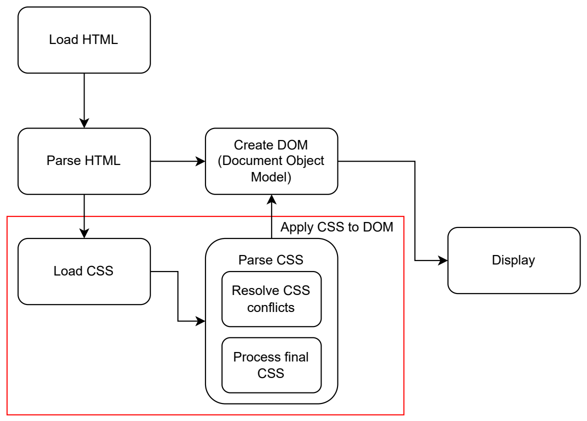


Figure 1: Visual representation of the CSS execution process

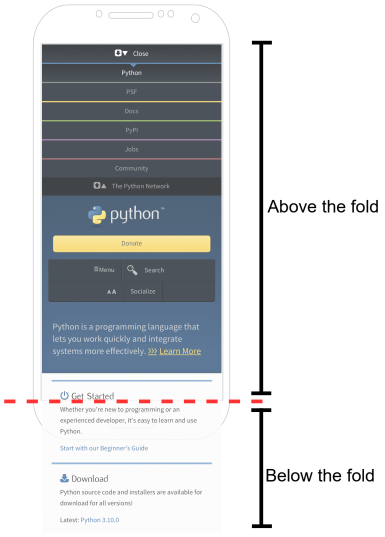
Any loaded web app is a collection of files together constituting a page. Most web apps consist of three main file types: Hypertext Markup Language (HTML), JavaScript (JS), and Cascading Style Sheets (CSS)[3] [4]. A simplified graph displaying the functioning of web app rendering is shown in Figure 1. HTML can be used to add text, sections, and a simple structure to your web app. JS is a programming language and can be used to add more complicated

[1]https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet
[2]https://www.statista.com/forecasts/1146844/internet-users-in-the-world
[3]https://www.w3.org/standards/webdesign/htmlcss
[4]https://www.w3.org/standards/webdesign/script

functions to your web app. For example, some functions could add calculations, allowing a user to hide specific HTML content, or send network requests to run external programs. Finally, CSS is used to style web apps by being able to change style related attributes such as colors, locations, and fonts of HTML elements.

In practice, not all the CSS of a web app is needed at all times. Only for the visible part of the web app, the CSS is required to let it function properly. The rendering of web apps is blocked until the CSS files are requested, received, downloaded, and processed by the browser. In Figure 1, all elements related to CSS are shown in the red frame. By reducing the amount and size of the files, and thereby reducing the time it takes to render the page, the perceived run-time performance can be increased[5].

Critical CSS is a technique that marks all the non-essential CSS, which is the content below-the-fold. The below-the-fold content includes all content a user can not see when the page is loaded initially (see Figure 2). Jovanovski and Zaytsev conducted an experiment to see how inlining critical CSS creates a significant decrease in the time to first render of a web app using 909 out of the 1000 most popular web apps from Alexa [3]. Based on the results, they concluded that on average the initial page rendering of a web app is 1.96 seconds faster when applying the Critical CSS technique. Therefore, by applying this technique the amount of work a browser has to do to show an initial page can be reduced. It is common to inline the critical-path of the CSS, which means that the critical CSS file is placed in the *<head>* element of the HTML document directly while the non-critical CSS file is loaded asynchronously (by a JS call). By doing so, the browser does not have to load all the CSS at once.



Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

Start with our Beginner's Guide

Download

Python source code and installers are available for download for all versions!

Latest: Python 3.10.0

Figure 2: Visual representation of below/above the fold content on the python.org site

Extracting all the above-the-fold CSS manually can become quite time consuming depending on the complexity of the web app. For example, the homepage of Amazon.com uses 1499 lines of CSS for the initial load, manually checking whether all these lines are used for above or below the fold content can take a while. Fortunately, there exist many open source tools that can automatically extract the critical-path of web apps. Three popular free tools are: CriticalCSS [6], Critical [7] and Penthouse [8] with 1600, 9300 and 2500 stars on GitHub respectively.

The goal of this research is to analyze the impact of implementing such Critical CSS on run-time performance and energy consumption in the context of consumer Android web apps. We discuss related work in Section 2. Then, the setup and execution of the experiment is discussed in Section 3 and Section 4 respectively. We show the results of our experiments in Section 5. Discussion and research threats are discussed in Section 6 and Section 7 respectively. Finally, we present our conclusions in Section 8.

2 RELATED WORK

The impact of critical CSS on battery usage of mobile devices while browsing the web has not been researched as much as other aspects of mobile web browsing. For example, battery usage while mobile browsing and how this can be measured have been researched extensively. The impact of Critical CSS has also been researched separately but not in combination with the impact on run-time performance and energy consumption of mobile devices. In this section, we will be summarizing relevant research, describing how our research differs from these papers, and how it builds upon them.

Thiagarajan et al. [1] proposed a tool to measure the energy consumption of different web elements, including CSS and Javascript files. With this tool, they measured the consumption of these elements for popular sites. It was shown that for Amazon.com 17% of the energy consumption was caused by CSS files while for Gmail it was only 3%. They discovered that the energy consumption caused by CSS files is highly dependent on how many items have to be styled using CSS rules. However, this research was published in 2012 and many claims that are made, such as the claim that companies have very limited optimization for mobile phones, may not be completely true anymore. It also focused on all elements required to render web apps while we only focus on the effect on run-time performance and energy consumption of using critical CSS. The last difference is that the research made use of hardware to measure the energy consumption while our research uses software based energy consumption measurements.

Bui et al. [4] performed a trade-off analysis between the energy and run-time performance in mobile web browsers. They discovered that mobile browsers are largely focused on run-time performance. In their research, they aimed to reduce the energy consumption of mobile web browsers without or minimally increasing loading times for mobile web apps. They managed to save 24.4% battery usage for Chromium on average without increasing load times and 10.5% of battery usage saved for Firefox with a 1.69% increase in load times. These power savings were made by combining Network-aware Resource Processing, Adaptive Content Painting, and Application-Assisted Scheduling. Their research proves that big energy savings can still be made but they have not tested the energy and run-time

[5]https://web.dev/extract-critical-css/

[6]https://github.com/filamentgroup/criticalCSS
[7]https://github.com/addyosmani/critical
[8]https://github.com/pocketjoso/penthouse

performance impact of critical CSS. Similar to the experiment of Bui et al., in this paper, we aim to analyze the trade-off between the energy and run-time performance but for the critical CSS.

Jovanovski and Zaytsev [3] have shown, for 909 out of the 1000 most popular web apps of 2016, that on average the CSS that is not inlined adds 1.96 seconds to the time it takes to render an initial web app. To find and extract the CSS that can be inlined they have created a tool that has three key functions: 1) find and extract CSS files from HTML documents, 2) find and extract CSS rules that are relevant for a given resolution (used to extract above-the-fold critical CSS), 3) remove external CSS scripts and split up the needed CSS files by inlining the critical CSS and use JS to load in the remaining CSS asynchronously. Our research differs from the described research as we focus on the possible change in energy consumption and run-time performance due to critical CSS. The tool provided by the research can be valuable as it is able to perform a key part of our research, namely to add critical CSS to web apps.

3 STUDY DESIGN

3.1 Goal and Research Questions

The goal of this study is to *analyze the impact of the Critical CSS technique for the purpose of evaluating the impact on run-time performance and energy consumption from the point of view of a developer in the context of Android mobile web apps.*

To achieve this goal, we identify two main areas it could affect - run-time performance and energy consumption. Therefore, we refine our goal into the following two research questions:

RQ1: *What is the impact of the Critical CSS technique on the run-time performance of Android mobile web apps?*

To measure this, we calculate in milliseconds (ms) the first paint, showing any elements on the screen, as well as the first contentful paint, showing actual content such as text or images on the screen. Finally, the loading time of the entire web app is also measured. The main advertised benefit of the Critical CSS technique is that the initial load time of a web app can be reduced [3]. We investigate this using this research question.

RQ2: *What is the impact of the Critical CSS technique on the energy consumption of Android mobile web apps?*

We measure this using software estimates of Android device energy consumption in Joule (J) during our experiments. For any new tool, it is also important to question the impact on energy consumption. In the context of critical CSS, it is not immediately clear whether it should have a beneficial or detrimental effect on energy consumption. Based on the implementation, it could mean less CSS is parsed, but it could also mean that redundant CSS is fetched from a server. In this research question, we investigate this.

Since the goal is to analyze the impact of the Critical CSS technique on the run-time performance and energy consumption of mobile web apps, the main perspective will come from the developers who have an interest in knowing what the exact influence is of Critical CSS on the web apps that they develop. Another group is the end-users. They are not directly interested in this research, however, they do benefit from any gains that may be made in the usage of their mobile Android devices.

3.2 Subjects Selection

To select the subjects of this experiment, we consider the entire population of Android mobile devices, Android versions, web browsers, and web apps. After that, a sample from the population is taken to perform the experiments. The population of distinct Android mobile devices is immense. In 2015, over 24,000 unique Android devices were released[9]. Sampling a meaningful amount of unique android devices is nearly impossible. Instead, a single device is tested. The experiments are performed on a Huawei Nexus 6P device. Similarly, the number of Android versions is also large. There were 20 main Android versions released between 2008 and 2020 with many smaller versions in between[10]. The Android version the tests are performed on is Android 6.0.1, a single Android version is tested to avoid complicating the experiment. Furthermore, we expect that the specific Android version has no impact on browsing the web. Another important decision is which browsers are tested. The top six browsers combined have 98.4% market share with Google Chrome having roughly 64.3% market share, Safari 24.2%, Samsung Internet 5.3%, Opera 2.2%, UC Browser 1.9% and Mozilla Firefox 0.5%[11]. Furthermore, Google Chrome, Mozilla Firefox, and Safari make up the top three most commonly used browsers for both initial development and testing by CSS developers[12]. From these statistics, the device we are using, and Android Runner, we decide to test Google Chrome and Mozilla Firefox as these are the top two browsers the Nexus 6P and Android Runner supports properly. The browsers considered in our experiment are Google Chrome 94.0 and Mozilla Firefox 93.0.

Table 1: Inclusion and exclusion criteria for subjects

Criteria	Description
Inclusion	Top 1000 of Tranco list
Exclusion #1	<style>tag in HTML head
Exclusion #2	Not downloadable with web app download tools

We randomly select 40 web apps from the Tranco list [5], an aggregation of four existing lists (*i.e.,* Alexa, Umbrella, Quantcast, and Majestic) which is stable and it has been designed for reducing the effort in replicating studies based upon it [6]. However, the Tranco list has the 1 million most popular web apps in descending order. Since this research wants to discover the real-world impact of the Critical CSS technique, it is logical to test it on the web apps that are being accessed the most and not on web apps with only a few visits per day. Therefore, 40 web apps from the top 1,000 of the Tranco list are randomly selected. Before deciding to include a certain web app, we firstly check if it does not already use the Critical CSS technique. This check is performed by checking if there is a $< style >$ tag present in the head of the HTML file using a Python script, this script is available in the replication package as explained in Section 3.7. The absence of this tag indicates that the Critical CSS technique is not used on the web app.

[9]https://www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2015-08/2015_08_fragmentation_report.pdf
[10]https://www.javatpoint.com/android-versions
[11]https://gs.statcounter.com/browser-market-share/mobile/worldwide
[12]https://2021.stateofcss.com/en-US/

3.3 Experimental Variables

To answer the research questions defined in Section 3.1 we consider the usage of the Critical CSS technique as our independent variable. There are two treatments for Critical CSS: *The Critical CSS technique is not applied* and *The Critical CSS technique is applied.* At every run of the experiment, a copy of the original web app is served in case the treatment is *not applied*, whereas for the *applied* treatment a modified version of the web app where the Critical CSS technique is used is served. This modified version of the web app is made using the Critical[13] tool. There is one blocking factor, namely the type of browser. Browsers might use different methods of rendering web apps and therefore we investigate them separately. The browsers considered in this research are Google Chrome and Mozilla Firefox as discussed in Section 3.2.

Table 2: Dependent variables considered in this research

Name	Description	RQ
Loading time (lt)	Time (in ms) until the web app is loaded in its entirety in seconds	RQ1
First paint (fp)	Time (in ms) until anything is visually different from the previous screen.	RQ1
First contentful paint (fcp)	Time (in ms) until the first text or image is painted.	RQ1
Energy consumption (e)	Energy consumption (in J) by the mobile browser running on the mobile device for completely loading a web app.	RQ2

The dependent variables are shown in Table 2. They function as an objective evaluation of the impact of the Critical CSS technique on energy consumption and run-time performance of mobile web apps. To measure the *loading time*, *first paint* and *first contentful paint* the PerfumeJS plugin[14] in AndroidRunner is used. To measure the energy consumption the Trepn plugin[15] is used.

3.4 Experimental Hypotheses

In this research, we aim to reason about the impact of the implementation of the Critical CSS technique, on each of the dependent variables in Table 2. As the browser has been identified as a potential blocking factor, we investigate this impact for each browser separately.

From the experiment, we calculate population means μ_{ijk} with $i \in \{fp, fcp, e\}$ each of the dependent variables, $j \in \{c, f\}$ each browser Google Chrome and Mozilla Firefox respectively, and $k \in \{0, 1\}$ a boolean indicating whether the Critical CSS technique was applied to this sample.

For the run-time performance variables *fp, fcp* we can reasonably assume that the introduction of the Critical CSS technique only improves on the values of the variables. To this end, we execute one-sided statistical tests on the hypotheses,

$$H_{0,ij}: \quad \mu_{ij0} = \mu_{ij1}, \qquad \forall i \in \{fp, fcp\}, j \in \{c, f\}$$
$$H_{a,ij}: \quad \mu_{ij0} < \mu_{ij1}. \qquad \forall i \in \{fp, fcp\}, j \in \{c, f\}$$

For the energy consumption variable it is unclear whether the Critical CSS technique imposes a positive or negative effect. To this end, we execute two-sided statistical tests on the hypotheses,

$$H_{0,ej}: \quad \mu_{ej0} = \mu_{ej1}, \qquad \forall j \in \{c, f\}$$
$$H_{a,ej}: \quad \mu_{ej0} \neq \mu_{ej1}. \qquad \forall j \in \{c, f\}$$

3.5 Experiment Design

To measure the effect of applying Critical CSS on the dependent variables, a full 2x2 factorial design is performed in which for each subject the dependent variables are measured for each combination of factors. Thus, each web app is served once with and once without the Critical CSS technique in each selected browser, resulting in 160 runs for a single trial. Collecting energy consumption, first paint times, first contentful paint times, and loading times can be inconsistent in the real world. To mitigate this, each trial is performed 10 times and the results of all 10 trials are saved. To serve the web app we made use of the Flask[16] framework with which the HTML files can be served locally. This eliminates the possibility for network related factors to affect the run-time performance or energy consumption. Furthermore, to ensure that no bias exists in the order of execution, each web app is randomly served with or without the Critical CSS technique.

3.6 Data Analysis

The collected data is a collection of numerical values for each dependent variable per subject in each browser and with or without the treatment applied. This data is then analyzed quantitatively.
Exploration. The collected data is analyzed using descriptive statistics and boxplots to get an indication for the first paint times, first contentful paint times, energy consumption, and loading times.
Check for normality. The data is tested to see if it can be approximated as normally distributed for the dependent variables in both browsers. Testing for normality is initially done using density plots. After that, Q-Q plots are used which should show a diagonal line for normally distributed data and any other line for non-normally distributed data. The Shapiro-Wilk test is also used for the same purpose. An α of 0.05 is used. So, this test should return a p-value of > 0.05 for normally distributed data and < 0.05 otherwise.
Normality assumed. Should the data be normally distributed, paired t-tests are used to test if the mean of the differences of the populations is 0. In this case the two populations are the complete samples of subjects to which the Critical CSS is either applied or withheld. The populations are tested for first paint times, first contentful paint times, energy consumption, and loading times. The paired t-test also provides information on whether the impact of the Critical CSS technique is positively or negatively correlated with the dependent variables. If there is a significant difference between the two populations according to the paired t-test, Cohen's d is used to determine the effect size.

[13]https://github.com/addyosmani/critical
[14]https://github.com/S2-group/android-runner/tree/master/AndroidRunner/Plugins/perfume_js
[15]https://github.com/S2-group/android-runner/tree/master/AndroidRunner/Plugins/trepn
[16]https://flask.palletsprojects.com/en/2.0.x/

Normality not assumed. If the data is not normally distributed, the Wilcoxon Signed Rank test is used to test whether there is a difference between the populations. With a p-value below 0.05, we reject the null hypothesis of equal population means. If there is a significant difference between the populations, Cliff's delta is used to determine the effect size.

3.7 Study Replicability

The method, execution and collected data for this research can be found in the replication package uploaded on Github[17]. The replication package can be used to verify the results and conclusions of this research.

4 EXPERIMENT EXECUTION

The execution of our experiment is divided into three main steps. Firstly, we prepare the subjects used in this experiment. Then, we setup the infrastructure for executing the runs. Finally, we run the experiment and perform the measurement.

4.1 Preparation

To be able to serve the selected web apps locally the Resources-Saver[18] tool will be used, which is available for Google Chrome and Mozilla Firefox. This tool is capable of locally downloading the front-end of third-party web apps. For each locally-stored web app, we manually check whether it is completed and/or contains any errors. Then, for each saved web app the critical CSS technique is applied using Critical[19]. A Python script is written to automate this process using the Command-line-interface (CLI) version of Critical. It takes a folder containing the *index.html* files as input and for each of those files creates a *critical.index.html* file. In addition, UTF-8 encoding is enforced on the files due to the nature of our local web app hosting. The result of this phase is to have, for each subject of the experiment, a pair of subjects: the original one and the one where the CSS technique is applied.

4.2 Setup

The goal of the experiment is to measure all the dependent variables for each of the factor combinations. A visual representation of the setup of our main experiment is shown in Figure 3. We have connected our Android device with a Raspberry Pi to charge and control the device, as well as collect the measures related to the dependent variables of our experiment. To prevent any connection issues that may affect the outcome of the experiment, we make sure to connect the Raspberry Pi to the local network by an Ethernet cable. The two devices are connected with each other through a USB cable. Furthermore, we make sure to avoid charging the Android device during execution. We achieve this behaviour by programmatically disabling the USB ports from charging during the runs and enabling it in between the runs.

For orchestrating the execution of all the runs of the experiment, we use Android Runner [7], a wrapper around Android SDK, Android Debug Bridge (ADB), and MonkeyRunner. Android Runner is a Python framework that is used for automatically executing measurement-based experiments on native and web apps running on Android devices. Android Runner measures the run-time performance metrics, *i.e.,* loading, first paint, and first contentful paint times via a dedicated plugin. Additionally, the battery power consumption of the Android device is measured by Android Runner via the Trepn profiler, an accurate widely-used software-based power profiler for Android apps [8].

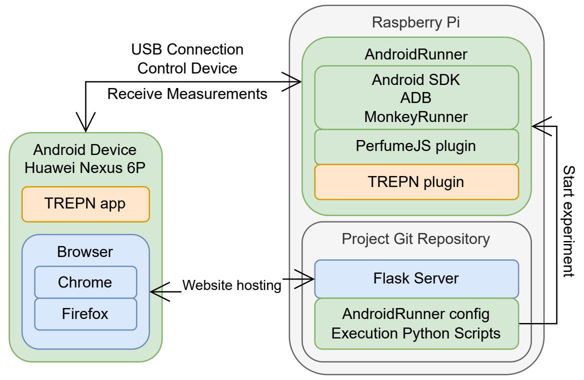


Figure 3: A visual representation of the experiment setup

We control the setting of the independent variables by deciding which version of each subject must be loaded at each run (*i.e.,* its original version or the one with the Critical CSS technique applied to). For this, the Chrome and Firefox browsers need to be installed on the Android device. The prepared web apps also need to be hosted for the device to access. To this end, we locally host a Flask web server on the Raspberry Pi that the device can access. Flask is a simple web application framework based on Python, it allows for quick and easy deployment of web apps. To simulate real-world usage, the network speed of the Raspberry Pi is throttled to 20 megabits per second, which is the typical real-world speed of 4G LTE [20], using Wonder Shaper [21]. Wonder Shaper is a script that can be used to throttle both upload and download speeds to a certain value as specified by the user.

4.3 Measurement

At the start of each run, Android Runner will be used to initiate the experiment. All parameters required for measuring the dependent variables, as described in Section 3.3, are specified in the *config.json* file. run-time performance measures are collected using the PerfumeJS plugin of Android Runner, whereas energy is used via its Trepn plugin.

At the beginning of the each run, the local Flask server is started. Once started, another Python script is executed which ensures that each web app is being randomly served once. To minimize the effect of older data on loading the new web app, the cache of the browser is cleared before each run. Furthermore, we decided that between each web app serving, one minute of idle time will be used to ensure that the battery of the device can charge.

[17]https://github.com/S2-group/EASE-2022-energy-critical-css-rep-pkg
[18]https://github.com/up209d/ResourcesSaverExt
[19]https://github.com/addyosmani/critical
[20]https://www.4g.co.uk/how-fast-is-4g/
[21]https://github.com/magnific0/wondershaper

Table 3: Descriptive statistics of loading time (lt), first paint (fp), first contenful paint (fcp) in milliseconds, and energy consumption (e) in Joules

CSS Technique	Without Critical CSS								With Critical CSS							
Browser	Chrome				Firefox				Chrome				Firefox			
Variable	lt	fp	fcp	e	lt	fp	fcp	e	lt	fp	fcp	e	lt	fp	fcp	e
Mean	1743	1456	1495	9.9	1963	1178	1440	18.5	1542	1117	1178	10.5	1745	1204	1081	19.0
Standard Dev	1197	774	784	8.4	1457	596	1231	15.9	1043	531	542	8.3	1333	683	979	15.5
Minimum	272	544	544	0.1	224	315	211	2.1	270	315	454	0.8	249	330	152	2.2
25% Quantile	939	1022	1044	4.9	840	777	625	9.8	765	758	791	5.6	755	766	427	10.0
Median	1482	1359	1401	7.4	1598	1121	1006	13.4	1319	1011	1087	8.0	1322	1069	741	14.9
75% Quantile	1994	1611	1676	11.4	2744	1405	1791	20.5	1863	1262	1293	12.5	2460	1391	1236	21.3
Maximum	8326	6517	6517	50.0	9522	4996	9346	116.8	5443	4089	4089	55.0	5946	4996	5599	91.6

5 RESULTS

The results section is divided into three parts. Firstly, the collected data is explored to get an initial idea of how the data is distributed. Secondly, each combination of dependent variables, treatments and, blocking factors is checked for normality in three steps. Thirdly, the hypotheses are tested.

5.1 Data exploration

The first step in the data analysis is to further understand our data by exploration. In Table 3, we show descriptive statistics of each browser and dependent variable, as well as whether the Critical CSS technique has been applied.

Looking at the means, we can see that across all subjects applying the Critical CSS technique improved the run-time performance but also increased energy usage. We also notice that the standard deviation is quite high concerning the mean, giving the first indication that this data is not normally distributed since negative consumption or time values should not be possible.

Additionally, for the Firefox browser with the Critical CSS technique applied, the mean first paint time is larger than the first contentful paint time, putting into question the validity of this data column as first paint should always finish loading before first contentful paint.

In Figure 4, we show the distribution of loading time, first paint, first contentful paint times, and the energy usage respectively for the Critical CSS technique when applied and not applied. The boxplots appear non-symmetrical and have many outliers which is another indication that the distribution is not normal. In Section 5.2, this is formally investigated.

These box plots further support our findings from the descriptive statistics that the loading, first paint, and first contentful paint times seem to slightly improve by applying the Critical CSS technique. In Section 5.3, this is formally investigated across all hypotheses.

5.2 Normality checks

As explained in Section 3.6, testing for normality is an essential step before testing the hypotheses. Depending on whether the collected data is assumed to be normally distributed or not, either paired t-tests or Wilcoxon Signed Rank tests are used. Testing for normality was done in three steps.

Density plots. Firstly, density plots were made to see if the data followed a bell curve. Due to the combination of dependent variables, treatments, and blocking factors a total of 8 density plots, 4 for Chrome and 4 for Firefox, have been constructed using the collected data. In each density plot, both the Critical CSS applied and not applied can be seen to compare the differences. The density plots for Chrome and Firefox can be seen in Figure 5. In these figures, the difference for each dependent variable along with its treatment can be seen and easily compared. The density plots for all combinations do not seem to resemble a bell curve. Thus, it is unlikely that the underlying data follows a normal distribution.

Note also how these density plots give another indication of the impact of applying the Critical CSS technique. Most notably, the energy measurements are fairly similar across both groups, as well as the first paint time of the Firefox browser. How statistically significant the other differences are, will be discussed in Section 5.3.

Q-Q plots. Q-Q plots are also made to check the normality of the obtained data. 16 Q-Q plots have been made using the collected data and are available in the replication package (see Section 3.7).

Shapiro-Wilk. Finally, the Shapiro-Wilk test was used. The results of these tests can be seen in Table 4. The values in this table represent the p-values of the Shapiro-Wilk test. As can be seen in the table, all p-values are below 0.05. So, according to the Shapiro-Wilk test, the data is not normally distributed.

Table 4: P-values of the Shapiro-Wilk tests

CSS Technique	Without Critical CSS		With Critical CSS	
Browser	Chrome	Firefox	Chrome	Firefox
Loading time	< 2.2e-16	3.071e-16	< 2.2e-16	< 2.2e-16
First paint	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
First contentful paint	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16
Energy usage	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16

Conclusion. Combining the above three steps for normality testing, we can conclude that there is no statistical evidence that the data comes from a normal distribution.

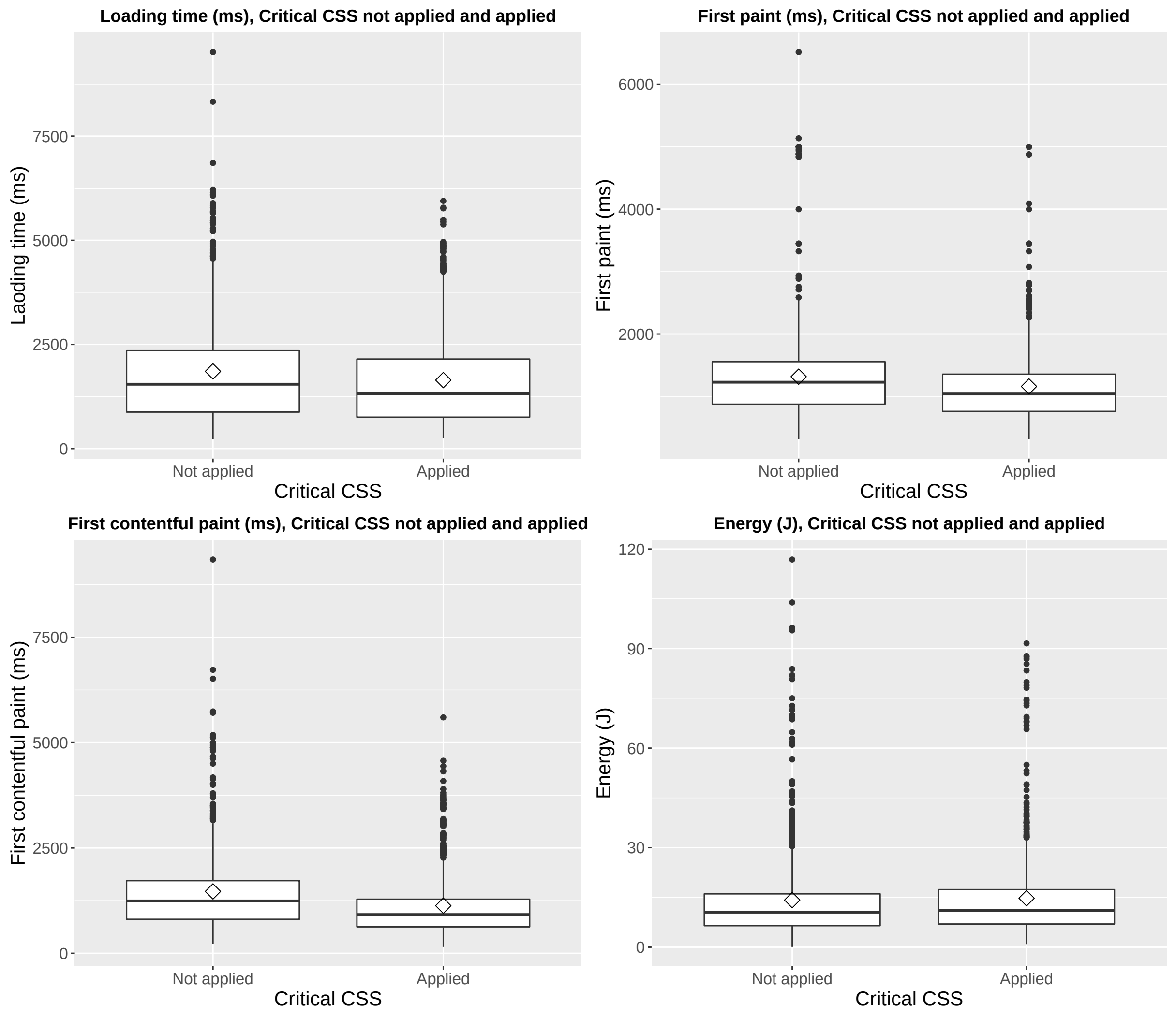


Figure 4: Comparison of without and with the Critical CSS Technique applied across all dependent variables

5.3 Hypothesis testing

Since the collected data are not following a normal distribution, we test our hypotheses using the Wilcoxon Signed Rank non-parametric test.

Table 5: P-values for all hypotheses and Cliff's Delta effect size estimates for each rejected null hypothesis.

Value	P-value Wilcoxon signed rank test		Cliff's Delta	
Browser	Chrome	Firefox	Chrome	Firefox
Loading time	**0.00726**	0.0838	-0.120	-
First paint	**7.59e-16**	0.714	-0.357	-
First contentful paint	**2.65e-06**	**3.97e-06**	-0.325	-0.233
Energy usage	0.139	0.959	-	-

RQ1: *What is the impact of the Critical CSS technique on the run-time performance of Android mobile web apps?*

The run-time performance consists of load time, first paint, and first contentful paint. The hypothesis test for Google Chrome shows that for the loading time the null hypothesis can be rejected and significant evidence to confirm the alternative hypotheses has been found, meaning that applying the Critical CSS technique leads to a lower load time for Google Chrome. The effect size is negligible, indicating that the difference in load times between sites that have the technique applied versus not applied is small. The p-values for both browsers testing the first paint are very different, this can most likely be explained by possible data corruption (explained in Section 7). However, to maintain valid conclusions, assumptions on what the results would be without corrupted data can not be made and it is only based on statistics. Following this reasoning, the null hypothesis cannot be rejected for the Firefox browser. Nevertheless, the null hypothesis for the first paint on the Chrome browser is rejected and the effect size is medium. Therefore, there is significant

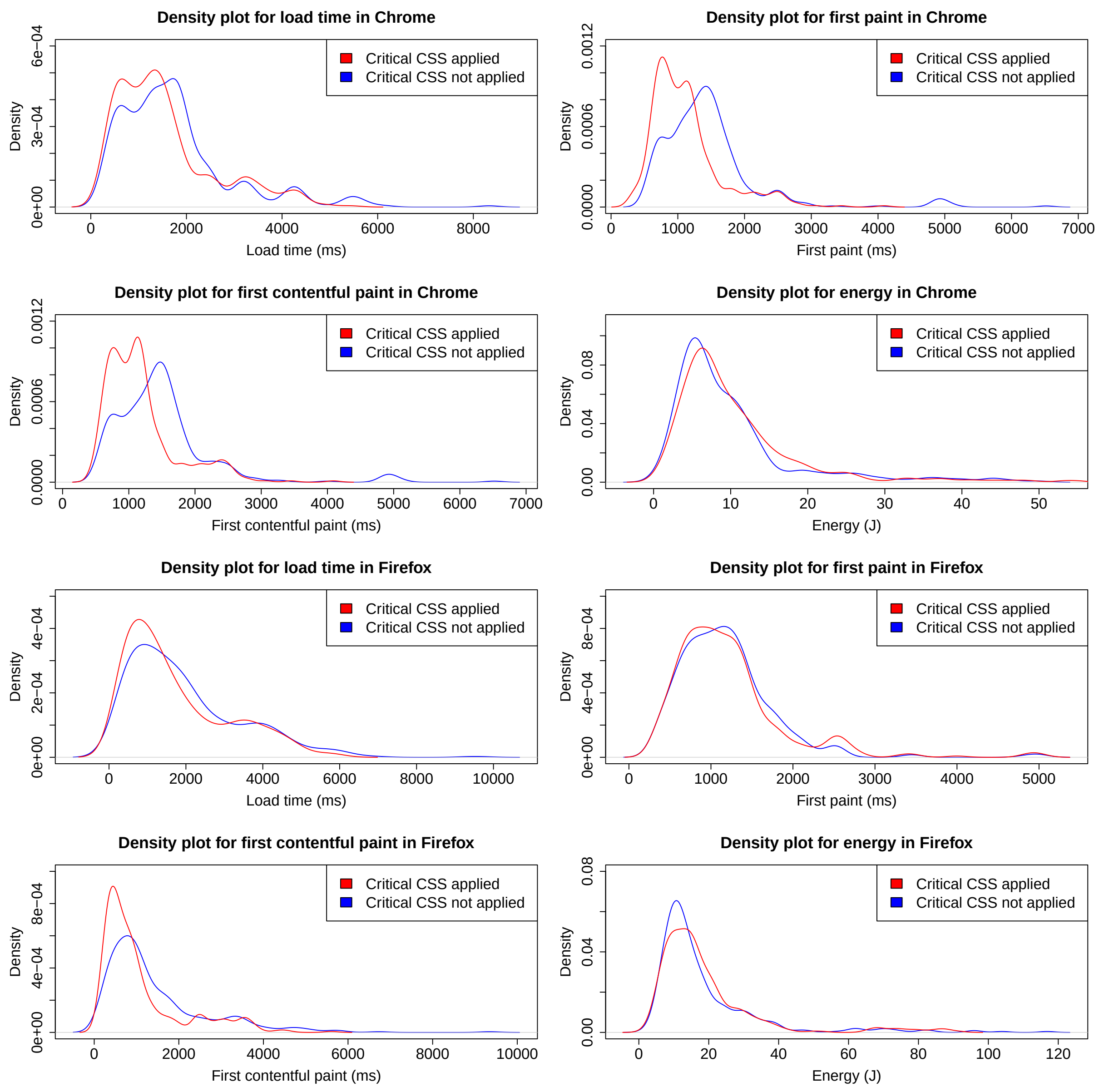


Figure 5: Density plots for Google Chrome and Mozilla Firefox respectively

evidence for the alternative hypothesis and it can be claimed that applying the Critical CSS technique leads to lower first paint times. The last run-time performance metric is first contentful paint, for which on both browsers the null hypothesis is rejected and the effect size is small. This shows that applying the Critical CSS technique leads to lower first contentful paint times on both browsers. Overall, it can be concluded that applying critical CSS to web apps served to Android mobile devices has a small but positive effect on their run-time performance.

RQ2: *What is the impact of the Critical CSS technique on the energy consumption of Android mobile web apps?*

The results show that for both browsers the null hypothesis about energy usage can not be rejected. Therefore it is concluded that applying the Critical CSS technique has no significant impact on the energy consumption of Android mobile web apps.

In our analysis we have treated the browser as a potential blocking factor. Throughout the analysis, we have seen that most results are consistent across both browsers, except for the unexpected first paint measurements on Firefox. Further investigation into the exact cause of that data is needed, but apart from that, there is no reason to assume that the type of browser should be used as a blocking factor in future research.

6 DISCUSSION

For RQ1, we conclude that applying critical CSS to web apps served to Android mobile devices has a small but positive effect on their run-time performance. From the perspective of *front-end developers*, applying critical CSS to web apps will enhance the run-time performance of their mobile web apps on Android. Therefore, depending on the characteristics of web apps, developers can apply the critical CSS technique to ensure a fast loading time of the web app.

For *browser vendors*, it is important to notice that applying the Critical CSS technique leads to lower first contentful paint times on both browsers. Thus the browsers may be benefited from the web apps where critical CSS has been applied to avoid the render blocking issue on web apps' initial load.

For RQ2, we further investigate the impact of the Critical CSS technique on the energy consumption of mobile web apps on Android devices. Our experimental results show that applying the Critical CSS technique tends to have no significant impact on the energy consumption of mobile web apps on Android. From the results for the energy consumption, it is not definitive whether the Critical CSS technique imposes a positive or negative effect.

From these results, it is concluded that web app developers can consider applying the Critical CSS technique to web apps in order to enhance run-time performance for Android devices. The statistical results show that this run-time performance gain can be realised with little to no significant costs with respect to energy consumption. It should be noted that Critical CSS is most likely not a one-size-fits-all solution and therefore web app developers should at all times investigate whether applying the technique makes sense for their application. A potential indication is the amount of content under the fold of the web app and the size of the corresponding CSS files. Rendering of web apps is blocked until the CSS files are requested, received, downloaded, and processed by the browser. When the CSS files are large or under poor network conditions, rendering a web application would take time. With critical CSS techniques, developers can speed up the page rendering time of their web apps and deliver a fast and smooth experience to the user who is under poor network conditions or on a mobile network where high latency is an issue. But critical CSS also has some downsides, e.g., it prevents caching CSS in the browser. This means that the users need to re-download the inlined critical CSS on every request. Therefore, with critical CSS, loading the first page might be faster in a newly visited web app, but it might slow down all subsequent pages or return visits of the previous pages.

7 THREATS TO VALIDITY

The following section discusses various threats that might affect the validity of this research. For this discussion, we will follow the classifications described by Cook and Campbell [9].

Internal Validity. The threat of notifications affecting the outcome has been minimized by disabling the notifications for most apps that send such notifications, such as the Play Store, Gmail, and other main applications. During the execution of the experiment the threats of history, maturation, and selection were mitigated by randomly selecting the subjects, applying the treatment to each selected subject randomly ten times, and clearing the cache between runs. This approach should also ensure that certain environmental factors such as battery charge, time of day, and caching, have the same effect on each run. However, it might be the case that the battery charge has had a different effect on the later runs. During each run, charging the battery was disabled and in between runs it was enabled again with a time between each run of one minute. Towards the end of the experiment, it was found that this time was not sufficiently long to ensure that the battery could completely charge, resulting in some of the later measurements to be collected on a non full battery.

Initially we selected 50 web apps from the Tranco list, however in this study we could consider only 40 of them as subjects of our experiment. Specifically, after executing the complete experiment it was found that for nine web apps (*i.e., apple.news, bestbuy.com, cam.ac.uk, etsy.com, eff.org, lazada.sg, opendns.com, theverge.com, unicef.org*) incomplete or no data was collected. To ensure that sufficient data was collected, a new run was executed to collect data for these missing subjects. For five of these nine missing subjects, a complete dataset was collected. Two subjects, *lazada.sg* and *theverge.com*, could not be served correctly. The exact reason for the error could not be retrieved from system logs. Most likely a certain type of encoding error seemed to occur during the experiment. The other two subjects, *opendns.com* and *bestbuy.com*, returned an incomplete set of measurements and were excluded from the results to ensure that statistically valid conclusions could be drawn. Finally, six more web apps (*i.e., allaboutcookies.org, asos.com, jhu.edu, lijit.com, python.org, and zerodha.com*) had to be removed due to one or more zero measurements.

A final threat to the internal validity is the usage of the software profiler Trepn for energy consumption. By using this plugin we assume that it correctly measures the energy consumption, but using a hardware device such as the Monsoon power monitor might result in more precise and reliable measurement. However, we do not indicate that this is the case as Trepn has been tested to perform close to hardware-based power monitors, such as the Monsoon, in terms of accuracy but this is still a potential oversight [10].

External Validity. To minimize the threats to the external validity the subjects for this study have been sampled randomly from the Tranco list. By doing so the selected sample is representative of the population of most used web apps. Here it should be noted that applying the critical CSS technique is not necessarily something that makes sense if one thinks about web apps with a certain design that loads all content at once, such as Google. Therefore, the results may not apply to the whole population of web apps, but the conclusions are applicable for the majority of existing web apps. Furthermore, this research focused on web apps that had no prior critical CSS applied. It could be the case that the web apps that have this technique applied, have done this with a specific design in mind that enhances the advantages of this technique. Investigating if there is a difference in run-time performance and energy consumption for those web apps is beyond the scope of this research, however, it may be interesting to investigate in the future.

Construct validity. The setup, which was explained in previous sections, attempted to minimize the effect of external factors on the outcome of the experiment and, if it could not be mitigated, it was tried to ensure that the external factor has an equal influence on each subject. To ensure that network related issues, such as DNS-lookup, package loss, or any of the many errors that can occur, are mitigated a local Flask-server was used to which the HTTP requests were made. In Section 4.2 we explained how this was mitigated.

Another factor that may have influenced the experiment is the usage of the local web apps. Here an interesting trade-off has to be made, namely between the network issues as mentioned above and simulating the real world. To measure the run-time performance of each web app three variables were measured, the load time, first paint, and first contentful paint. However, after analyzing the results, it was found that in Chrome, sometimes, both the first paint and first contentful paint contained the same value, while in Firefox these values are different. Furthermore, in some cases, the PerfumeJS plugin returned no first paint value for Firefox and in other cases, Trepn returned no energy consumption for Google Chrome. To ensure validity, this data had to be removed. Due to these differences, it was investigated with a test setup if these errors are browser specific, or the error may be related to serving the web apps locally where an HTML page was served with a delayed content aggregated in JavaScript. In Chrome, this delay was detected, but Firefox did not return a first paint value. The root of this error may lay in the fact that the web apps were downloaded and certain functions or triggers did not execute. Future research might give more insights into these observed differences. Fortunately, a first contentful paint value was found each time a web app was served and, together with the load time, we believe that sufficient data was gathered to measure the run-time performance.

Conclusion Validity. As can be seen in Section 5.3 most run-time performance metrics show that the null hypotheses can be rejected. However, the discussion above challenges these values, as certain metrics, like the first paint and first contentful paint times, might have been affected by the different browsers and experiment setup. Further exploration into the exact workings of the metrics calculations should give a clearer view of how these results can be interpreted.

Even in cases that data is not normally distributed, it is possible to perform the parametric paired t-test. However, in the context of not normally distributed data, the Wilcoxon Signed Rank test has larger statistical power to test our hypotheses.

8 CONCLUSIONS

In conclusion, this paper researched the effect of the Critical CSS technique on Android mobile web apps. We tested 40 web apps with and without the Critical CSS technique applied on Google Chrome and Mozilla Firefox for the first paint, first contentful paint, energy consumption, and loading time. The results of these tests showed that applying the Critical CSS technique had a medium sized positive impact on the first paint for Google Chrome and no measurable impact on first paint for Mozilla Firefox. It also had a small sized positive impact on the first contentful paint for Google Chrome and Mozilla Firefox. Finally, the loading times differences for Google Chrome were negligible and no measurable difference for Mozilla Firefox. The energy consumption for both Google Chrome and Mozilla Firefox showed no significant difference between applying the Critical CSS technique or not. Therefore, depending on the characteristics of the web application, it is advisable for developers to apply the Critical CSS technique to enhance run-time performance on Android devices.

Possible future work include replicating the experiment with web apps for which the Critical CSS technique is expected to have a significant impact instead of focusing on the Tranco top 1000. These web apps should have a large portion of their content below-the-fold, as this content generates CSS that is not in the critical path. Another approach would be to focus on web apps that already have applied the critical CSS technique and remove the critical path from the web app. In this way, it can be tested if applying the technique has made significant improvements. Additionally, it would be useful to replicate this research using devices that run on iOS as it holds a market share of 26.75%[22] in the market of mobile operating systems. If for iOS devices the same conclusions can be drawn as for Android devices, the vast majority of mobile devices would benefit from applying the Critical CSS technique. A different approach to reduce internet traffic and thus likely reducing energy usage for CSS is the experimental prefers-reduced-data feature[23]. However, this feature is currently not supported by any browser. Nevertheless, a small percentage (3.9%) of CSS developers[24] stated that they have used it so this becomes interesting to experiment with once the technology matures. Furthermore, in the future, it will be interesting to investigate how applying the Critical CSS might impact the energy consumption of Android devices with different screen sizes.

REFERENCES

[1] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh, "Who killed my battery? analyzing mobile browser energy consumption," in *Proceedings of the 21st international conference on World Wide Web*, 2012, pp. 41–50.

[2] D. Tian and Y. Ma, "Understanding quality of experiences on different mobile browsers," in *Proceedings of the 11th Asia-Pacific Symposium on Internetware*, 2019, pp. 1–10.

[3] G. Jovanovski and V. Zaytsev, "Critical css rules—decreasing time to first render by inlining css rules for over-the-fold elements," in *Postproceedings of 2016 Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE)*, 2016, pp. 353–356.

[4] D. H. Bui, Y. Liu, H. Kim, I. Shin, and F. Zhao, "Rethinking energy-performance trade-off in mobile web page loading," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, Sep. 2015. [Online]. Available: https://doi.org/10.1145/2789168.2790103

[5] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. Internet Society, 2019, pp. 1–15.

[6] O. de Munk and I. Malavolta, "Measurement-based experiments on the mobile web: A systematic mapping study," in *Proceedings of the International Conference on Evaluation and Assessment on Software Engineering (EASE)*. ACM, 2021, pp. 191–200. [Online]. Available: http://www.ivanomalavolta.com/files/papers/EASE_2021.pdf

[7] I. Malavolta, E. M. Grua, C.-Y. Lam, R. De Vries, F. Tan, E. Zielinski, M. Peters, and L. Kaandorp, "A framework for the automatic execution of measurement-based experiments on android devices," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering Workshops*, 2020, pp. 61–66.

[8] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma, "Modeling, profiling, and debugging the energy consumption of mobile devices," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–40, 2015.

[9] D. T. Campbell and T. D. Cook, "Quasi-experimentation," *Chicago, IL: Rand McNally*, 1979.

[10] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma, "Modeling, profiling, and debugging the energy consumption of mobile devices," *ACM Computing Surveys*, vol. 48, no. 3, pp. 1–40, Feb. 2016. [Online]. Available: https://doi.org/10.1145/2840723

[22]https://gs.statcounter.com/os-market-share/mobile/worldwide
[23]https://developer.mozilla.org/en-US/docs/Web/CSS/@media/prefers-reduced-data
[24]https://2021.stateofcss.com/en-US/