Accelerating Mobile Web Performance through JavaScript Classification on the Browser

Sashank Silwal
Computer Science, NYUAD
sashank.silwal@nyu.edu

Advised by: Yasir Zaki

ABSTRACT

JavaScript is widely recognized as the primary programming language for the web and is utilized by 97.9% of all websites as a client-side programming language [13]. It plays an indispensable role in facilitating dynamic interactivity on websites, including running animations, making network requests, and more. However, the use of JavaScript on modern mobile web pages has created a significant performance bottleneck for low-end mobile phone users, particularly in developing regions. The average page size has increased from 1720 Kilobytes in 2016 to 3077 Kilobytes in 2021 [2], with JavaScript contributing increasingly to the overall page size (from 24% in 2016 to 27% in 2021) [2]. Compared to other web resources of the same size, JavaScript is more computationally expensive to process due to the need for parsing, interpretation, and execution. This cost contributes significantly to the growing complexity of modern mobile pages, creating a critical performance bottleneck for low-end smartphone devices, which ultimately results in reduced user engagement. Lower-tier mobile devices with limited RAM and CPU power struggle to process JavaScript code efficiently, while slow internet speed can impede the faster downloading of the JavaScript file over the network.

To address this issue, we propose an approach that generates lightweight versions of mobile web pages by directly eliminating the use of unnecessary JavaScript by the browser. Our solution involves the use of a supervised machine learning model that operates on the browser and offers insights into each JavaScript script embedded in a web page. The model is designed to enhance the web browsing experience by predicting the class of each script, preserving essential scripts, and blocking non-essential ones. Our research goals include developing and training the machine learning model to classify JavaScript code, caching the classifications for later use, deploying the model in an open-source browser such as DuckDuckGo, and comparing the results with similar existing solutions. By using an open-source browser like DuckDuckGo, we can ensure that our solution is accessible and transparent, while also taking advantage of the browser's advanced features and capabilities. Our proposed solution aims to improve the performance and usability of mobile web pages for low-end mobile devices and areas with slow internet speeds.

KEYWORDS

Supervised Machine Learning, Classification, JavaScript, Web Page, User Experience

Reference Format:
Sashank Silwal. 2023. Accelerating Mobile Web Performance through JavaScript Classification on the Browser. In *NYUAD Capstone Project 2 Reports, Spring 2023, Abu Dhabi, UAE*. 9 pages.

1 INTRODUCTION

JavaScript (JS) is a popular programming language that is widely used on the web to enable dynamic interactivity, such as making network requests, running animations, and more. However, the use of JS on modern mobile web pages can also be a major contributor to slow page load times and a poor browsing experience. This problem is particularly significant in low and middle-income countries, where 87% of broadband connections are made through mobile devices [9]. These countries often rely on affordable, low-end smartphone devices to access the web, which can struggle with the processing power and memory required to load and execute heavy JS scripts.

The current state of the World Wide Web (WWW) shows a 45% increase in JS usage on a median mobile page, with only 7% fewer JS kilobytes transferred to mobile pages compared to desktop pages (475.1 KB for desktop and 439.9 KB for

This report is submitted to NYUAD's capstone repository in fulfillment of NYUAD's Computer Science major graduation requirements.

جامعة نيويورك أبوظبي
NYU ABU DHABI

Capstone Project 2, Spring 2023, Abu Dhabi, UAE

mobile) [3]. While the main purpose of JS on web pages is to provide interactive content, it is not always essential to the main page content or interactive functionality [6]. This raises the question of how to reduce the impact of JS on web page performance, particularly for low-end mobile devices and slow internet connections.

This paper proposes a solution that produces lightweight versions of mobile web pages by eliminating unnecessary JS directly through the browser. Our approach builds on an existing solution called SlimWeb, which is a JS classification approach that lightens mobile web pages on the fly using a browser plugin. However, we address the limitations of SlimWeb by implementing the machine learning model directly on the browser, rather than using a plugin. This approach takes advantage of the greater resources available on the browser, such as memory storage. We will be using an open-source browser called DuckDuckGo, known for its privacy features and built-in ad blocker [4], to implement and test our solution. We will reuse the machine learning model developed using 127,000 JS elements by scraping more than 20,000 popular web pages for SlimWeb, which has a median accuracy of 90% [5]. Our goal is to extract highly predictive features from the code and use them to classify the JS, improving the performance and usability of mobile web pages for low-end devices and slow internet connections.

To achieve this, Falcon Browser includes a service that periodically crawls popular web pages and classifies their embedded JS using a supervised learning model into critical and non-critical elements. Upon starting the browser, it requests the classification results with the service. The browser is then responsible for preserving the critical JS elements while blocking the non-critical ones, resulting in lighter mobile pages. Falcon Browser is designed primarily for low-end smartphones that are common among mobile users in developing regions. This approach can improve page load times, reduce bandwidth and energy consumption on mobile devices, and extend battery life.

In summary, the contributions of this paper include:

- An ML-driven JS classifier that categorizes JS elements with 90% accuracy.
- An easy to install solution that is integrated into an open-source browser that benefits from the classification by providing lighter versions of web pages for mobile users.
- 30% reduction in page load times compared to the original pages

2 MOTIVATION

The development of SlimWeb and Falcon Browser is premised upon the recognition that not all JavaScript (JS) may be integral to the core functionality of a webpage. In this regard, the selection of github.com serves as an example for showcasing the optimization of webpage performance through the removal of non-critical JS elements, whilst maintaining the primary content and interactive features of the page. Falcon Browser's analytical capabilities demonstrate that the github.com webpage encompasses 1 advertisement, 7 analytics, 6 content delivery network (CDN), 1 content, 4 hosting, 2 miscellaneous, 1 social, 6 tag manager, and 7 utility elements. A performance analysis was conducted on two versions of the github.com page: the original version and a JS-optimized version. The latter was loaded by Falcon Browser, which blocked analytics, advertising, and social elements whilst preserving other JS elements.

Table 1: Performance Comparison of Original and Optimized Versions of github.com

Metric	Original	Optimized
Number of JS requests	35	26
Size of JS resources (KB)	393.1	324
Speed Index (s)	5.2	4.8

Our analysis revealed that removing non-critical JS elements reduced the number of requests by approximately 26% (from 35 requests in the original page to 26 requests in the optimized page). Despite these reductions, the main content and functional components of the page remained intact. We confirmed this through visual evaluation. However, it is worth noting that the removal of an ad element from the top of the page did occur. Overall, our example highlights the potential benefits of identifying and removing non-critical JS from web pages to improve the user experience for mobile users without sacrificing quality.

For further details, please refer to Table 1.

3 RELATED WORK

An increasing interest in the research community to achieve a solution to simplify the ever-increasing complexity of web pages is seen with different approaches. One solution has been to decrease the web page complexity caused by the presence of JS. For example, Opera Mini is a web browser that uses the server to process JS and communicate the results to the client browser. This process requires constant communication between the browser and the server [5], which can lead to delays in the connection.

JSCleaner [6] is another approach to web and JS simplification that examines the content and classifies it into different buckets to transform webpages into simpler or "lighter" versions. In contrast to our approach of eliminating unnecessary JS, JSCleaner simplifies the use of JS in existing pages by selectively removing or replacing a portion of these scripts, rather than eliminating them.

SlimWeb is a solution that aims to speed up page loads and improve the mobile browsing experience [5]. It uses a JS classification service and a browser plugin to block non-critical JS elements based on labels received from the service. The JS classification service crawls popular web pages to identify the JS elements and uses a supervised learning classifier to categorize them.

Wprof [14] is an in-browser tool that acts as a profiler to provide an understanding of the key hindering effects behind Page Load Time (PLT). PLT is the average amount of time it takes for a page to show up on the screen. Wprof captures the dependency graph for a given web page and identifies delay bottlenecks. The study also found that synchronous JS evaluation plays a significant role in PLT because it blocks parsing.

In the industry, companies like Google and Facebook have attempted to tackle the problem of growing web complexity through Google AMP [8] and Instant Article [7]. AMP redefines how pages should be written by providing web developers with a framework to create their web pages. Instant Article is an HTML document that loads very quickly on Facebook. A major difference between our approach and these approaches is that we aim to simplify what already exists on today's web, rather than creating new web pages.

Due to the increasing complexity of JS and the burden it puts on low-end phones to download, load, and process the scripts, there has been significant interest in the research community in finding ways to address this issue. The above-mentioned works have a similar goal of decreasing the complexity of web pages to improve the user experience, but the methods applied are different from the one discussed in this paper.

To handle the cost of JS, web developers utilize uglifiers [1] to reduce the size of JS files before using them in their pages. These uglifiers remove all unnecessary characters (including white spaces, new lines, and comments) from these files without changing functionality. However, no speedups are expected at the processing level since the browser has to interpret the entire JS.

In contrast, JS blocking extensions [8] can aid users in reducing the amount of JS transferred to their browsers, potentially leading to processing speedups. These blockers rely on updating the blocking lists to prohibit a specific category of JS, such as ads or trackers, with a main drawback being their inability to predict if an "unknown" JS falls into a target category. This is because the rules in their blocking lists are generated by human annotators [11], which hinders effective maintenance. For example, the most popular list consists of around 60,000 rules [12] and is not generalized to handle unseen examples.

Our approach aims to simplify existing web pages by eliminating unnecessary JS to improve the user experience. Unlike Opera Mini, we do not require constant communication between the browser and server. Unlike JSCleaner, we do not selectively remove or replace a portion of JS, but rather eliminate it entirely. Unlike SlimWeb, we do not use a classification service and browser plugin to block non-critical JS elements. Unlike Wprof, we do not aim to identify delay bottlenecks, but rather eliminate JS altogether. Unlike Google AMP and Instant Article, we do not involve creating new web pages, but rather simplifying existing ones. Unlike uglifiers, we do not merely reduce the size of JS files, but eliminate them altogether to improve processing speed. And unlike JS blocking extensions, we do not rely on human-generated rules or lists to block specific categories of JS, but rather use machine learning to identify and eliminate unnecessary JS.

4 METHODOLOGY

4.1 JavaScript Classification

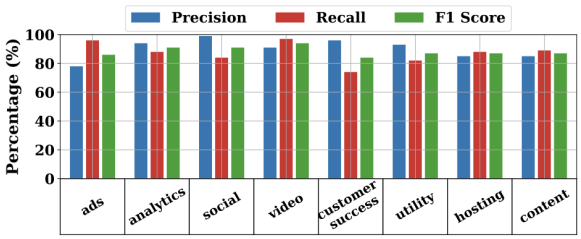


Figure 1: Summary of the 4-layer Neural Network from the SlimWeb Paper

In this study, we utilized the classification model developed by SlimWeb [5] to categorize JavaScript (JS) elements in web pages. The SlimWeb model is a multi-class classification approach that employs a neural network and has achieved a precision-recall of 90% with a dataset of 127,000 JS elements, as depicted in Figure 1. This neural network was trained on a dataset of labeled JS elements, where each element belonged to one of 12 known categories. The total number of neurons in the input layer was set to be equal to the total number of features, while the optimal number of neurons in the first and second hidden layers were set to 350 and 50, respectively, through hyper-parameter search. Performance evaluation was done using recall, precision, and F1-score as metrics.

Falcon Browser, utilizes the SlimWeb classifier to categorize JS elements in a web page and then label them as either critical or non-critical. Non-critical elements include ads, socials, marketing, and analytics, based on both user surveys and quantitative evaluations presented in the SlimWeb paper [5].

4.2 Implementation of the ML model on Browser

Algorithm 1 Classifying the JavaScript

```
Input: string script
Output: category
category, predictionProb ← MLClass(script)
if predictionProb > 0.8 and category ∈ [ads, marketing, socials, analytics] then
    return category
else
    return unassigned_category
end if
```

After the completion of training the model, we converted it to the ONNX format, which is specifically designed to ensure compatibility with different frameworks and hardware platforms. The ONNX format facilitates deployment of the model to various environments, including the DuckDuckGo browser, which is written in Java. In order to integrate the model into the browser, we employed the ONNX Runtime Java API. The ONNX Runtime is an inference engine optimized for ONNX models, designed to be cross-platform and cross-framework, and supports a wide range of programming languages, including Java. The ONNX Runtime Java API allows us to load the model into the browser and perform real-time classification of JS elements on web pages.

However, this approach is not without its limitations. One major drawback is the need for the client browser to download the unknown JS file in order to classify and label it, which can be time-consuming and negatively impact performance. To address this issue, we implemented a caching mechanism that stores the classification results for later use. This means that a JS element only needs to be classified once upon first request to the website, and the label will be known in subsequent requests to the same JS, reducing the need for repeated classification of the same scripts.

Additionally, to further reduce the number of JS elements that require classification, we established a server-based solution that updates the classifications on a regular basis and shares the results with the client-side browser if the page is encountered for the first time. This enables the browser to cache the classification results and avoid the need to download or classify the scripts itself, thereby enhancing performance.

Our ultimate objective is to develop the most effective and efficient solution for integrating the classification engine into the browser, with the aim of improving the performance and usability of mobile web pages on low-end devices and in areas with slow internet speeds.

5 FALCON BROWSER DESIGN

The Falcon Browser aims to enhance the browsing experience on mobile devices by accelerating page loads, with two primary components: a JavaScript classification service and the in-browser JavaScript classification mechanism. The former employs a supervised learning classifier that categorizes JS elements into one of nine pre-defined categories, including advertising, analytic, social, marketing, video, customer success, utility, hosting, and content, which are commonly used and defined by experts in the web community. The first four categories are non-critical, while the remaining categories are critical to the page content or interactivity.

As demonstrated in the Figure 2 JS classification service crawls popular web pages to identify JS elements, which are then labelled and stored in a database. Periodic updates are shared with users' browsers, which are responsible for classifying and blocking non-critical JS elements. To avoid the intensive task of classifying every JavaScript on a mobile device, Falcon Browser integrates a classifier service that crawls the top 350 web pages on Tranco's list and stores their classifications in a database. At the beginning of each run, Falcon Browser downloads this classification and saves it in a cache. When there is an HTTPS request to a JavaScript file, the cache is checked for an existing classification. If there is a cache miss, the browser employs an internal pre-trained machine learning model, which predicts the category and compares its accuracy to a threshold (set at 0.8 for the test case), as demonstrated in Algorithm 1. If the category is ads, analytics, marketing, or socials, and the accuracy is higher than the threshold, the script is blocked from executing. By using this approach, Falcon Browser reduces the processing load on mobile devices and enhances the browsing experience for users.

6 EVALUATION

This section presents an evaluation of Falcon Browser based on objective metrics, with a focus on webpage loading speed, data savings, and web compatibility. The evaluation employed a Xiaomi Redmi Go, a low-end mobile device with a quad-core 1.4 GHz Cortex-A53 CPU and 1 GB RAM. To facilitate the evaluation, the mobile device was connected via USB to a Linux machine that utilized the WebPageTest browser automation tool. This tool enabled automated webpage loads, telemetry collection, and performance metrics and network requests analysis.

The evaluation focused on two classic web performance metrics, namely SpeedIndex and PageLoadTime. The SpeedIndex is the average time at which visible parts of the page are displayed, while PageLoadTime measures how long it takes for a page to fully load, including all its resources such as images, scripts, and stylesheets. Specifically, the evaluation

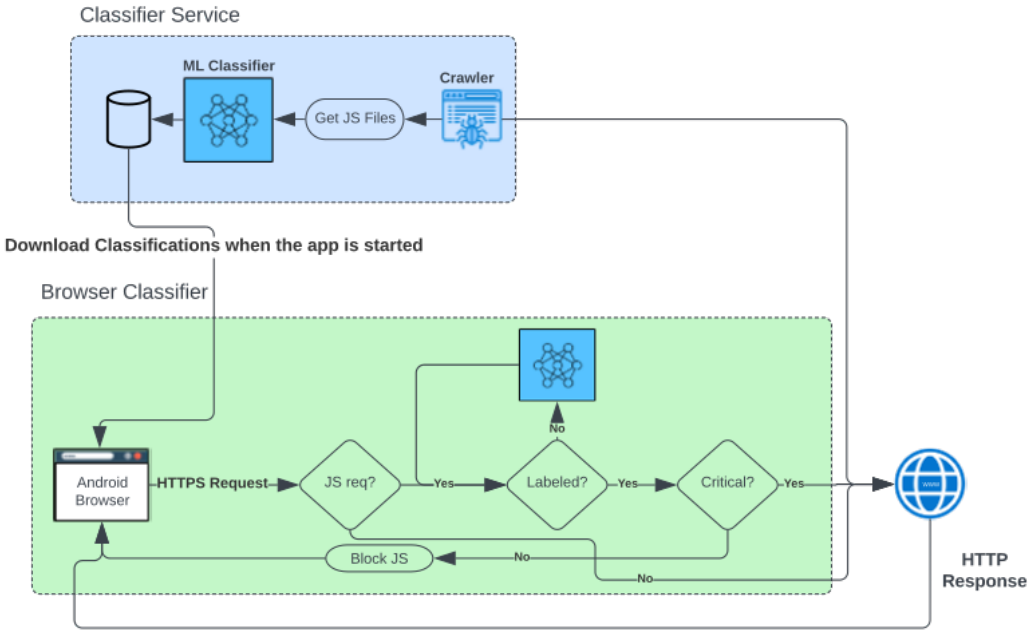


Figure 2: Browser Architecture

employed Falcon Browser to load the Top 350 websites from Tranco's list, and each webpage was loaded three times to obtain an average of the metrics [10].

To evaluate the efficacy of Falcon Browser, the study adopted the threshold values of 0 and 0.8 proposed by the SlimWeb paper. The threshold of 0.8 was found to provide optimal results by preserving the visual integrity of most web pages while still removing non-critical JavaScript code. By testing these two thresholds, the study aimed to investigate whether Falcon Browser's JavaScript classification algorithm could accurately identify and remove non-critical scripts without compromising the visual content and functionality of web pages.

The Tranco list was chosen for the evaluation as it is generated by combining existing rankings to generate a more reliable ranking. The list currently includes the lists from four providers: Alexa, Cisco Umbrella, Majestic, and Farsight (only for the default list). Through this evaluation, the study aimed to assess Falcon Browser's ability to deliver optimized web performance and user experience on low-end mobile devices.

6.1 Quantitative Results

The figures presented in this study shed light on the effectiveness of Falcon Browser in blocking non-critical JavaScript requests and reducing the size of the requested scripts. The

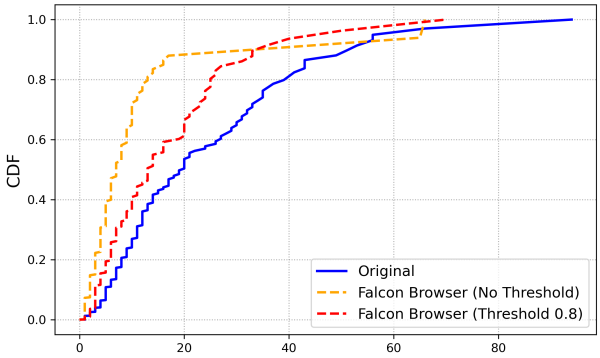


Figure 3: Distribution of Number of JS requests

Cumulative Distribution Functions (CDFs) in Figure 3 demonstrate that, on average, Falcon Browser blocks 25% of the total JavaScript requests per web page when a threshold of 0.8 is applied. When no threshold is imposed, Falcon Browser is able to block up to 55% of non-critical JavaScript requests, leading to a reduction of the original requests by 45%. This significant reduction in the number of requests results in lower data downloaded and higher bandwidth saved, contributing to improved web browsing efficiency.

Furthermore, the box plot shown in Figure 4 further supports the effectiveness of Falcon Browser in reducing the size of JavaScript requests. The median size of the original

Table 2: Comparison of PLT and Speed Index median values on 3G network

	PLT			Speed Index		
Network	%	Original	Falcon Browser	%	Original	Falcon Browser
3G	21.4	41.6	32.7	19.6	38.7	31.1

JavaScript requests was found to be 1180 KB, which was reduced by 23.7% to 900 KB when a threshold of 0.8 was applied. When the threshold was removed, the size of the requested scripts was reduced even further by 46.7% to 630 KB. These findings highlight the potential impact of Falcon Browser in improving web browsing experience and reducing the data consumption for users.

It is worth noting that the reduction in the size of JavaScript requests may not only contribute to improved web browsing efficiency but also have significant implications for website developers. With the increasing demand for faster and more efficient web browsing, developers may need to optimize their websites to reduce the number of non-critical JavaScript requests and improve the overall performance of their websites. This study provides valuable insights into the effectiveness of Falcon Browser in reducing the size of JavaScript requests and suggests that similar approaches could be adopted by website developers to optimize their websites and improve the browsing experience for their users.

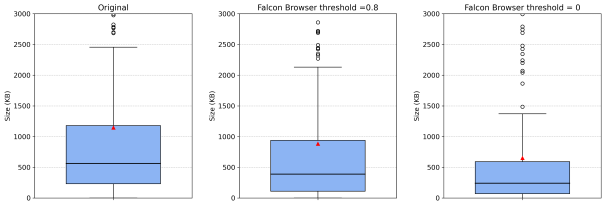


Figure 4: Comparison of Web Page Size Distribution (only due to JS) Requested with and without Falcon Browser, with and without Threshold

The results of our experiments clearly demonstrate the effectiveness of Falcon Browser in optimizing web pages and improving user experience, as shown in Figures 6a and 6b. By computing the delta between the optimized and original pages, we can see that in the majority of cases, our browser achieves savings in terms of SpeedIndex and PageLoadTime.

It is worth noting that the SpeedIndex metric is particularly important in assessing user experience, as it captures the time it takes for the page to become visually complete. A shift to the right in the CDF indicates that more pages are experiencing faster loading times, which is a clear indication of the effectiveness of Falcon Browser. The fact that 70-80

Similarly, the improvement in PageLoadTime is a crucial factor in determining user satisfaction, as it indicates the time at which the browser signals that all content has been loaded. As shown in Figure 6a, Falcon Browser achieves significant improvements in this metric, with a clear shift to the left in the CDF. This suggests that our optimizations are effective in reducing the overall loading time of web pages.

Overall, these results provide strong evidence in support of Falcon Browser as an effective tool for optimizing web pages and improving user experience. By reducing the amount of data downloaded and optimizing the loading of critical content, our browser achieves significant savings in terms of both bandwidth and time, making it a valuable tool for users with slow or unreliable network connections.

7 DISCUSSION

7.1 Limitations

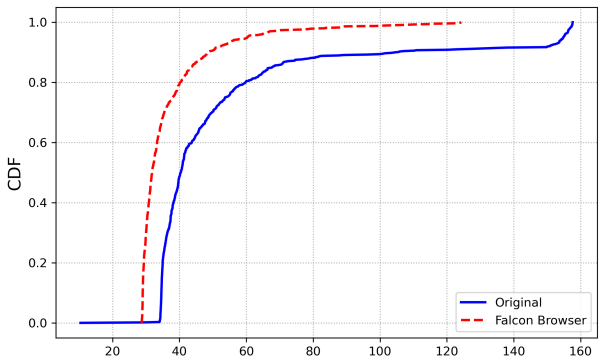
Despite the promising results shown by the Falcon Browser in terms of reducing data consumption and improving page load time, there are still some limitations and opportunities for future research and improvements.

One of the limitations of the Falcon Browser is that it is built on top of DuckDuckGo browser which is based on Webkit and WebPage tests that require the chrome toolkit api. As a result, the metrics used for testing the browser's performance were limited. In the future, there needs to be an alternative approach to better run the evaluations that can accurately capture all the relevant metrics.

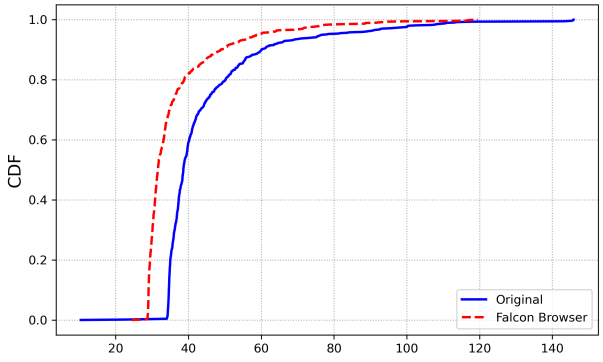
Moreover, although the Falcon Browser is built on the Slimweb paper, there is still room for further work to develop a better model for accurate predictions. This model should take into account the specific characteristics of different web pages and networks, as well as potential changes in user behavior over time.

In addition, there is an opportunity for further research to explore the impact of the Falcon Browser on other aspects of web browsing, such as user experience, security, and privacy. For instance, future studies could investigate the effect of Falcon Browser on page rendering speed, user engagement, and user satisfaction. Furthermore, there is an opportunity to explore the potential of the browser to reduce the risk of malicious attacks, such as phishing and malware, and to protect user privacy by blocking trackers and other unwanted scripts.

Finally, there is a need for broader adoption of the Falcon Browser and similar lightweight browsing tools. As more

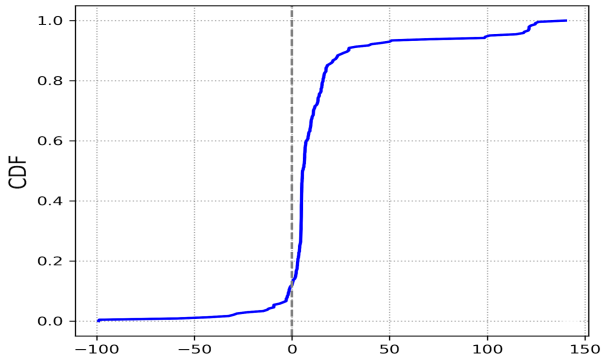


(a) Page Load Time (s)

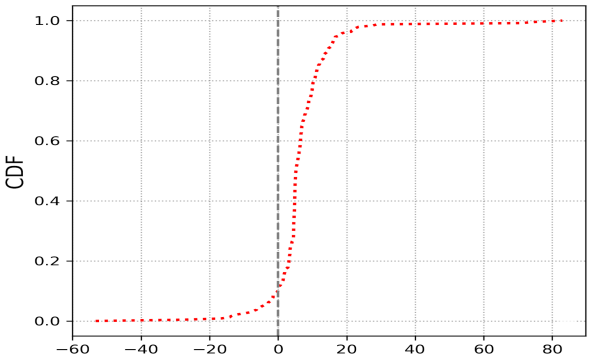


(b) Speed Index (s)

Figure 5: Falcon Browser configuration (0.8) : Quantitative results



(a) Delta Page Load Time (s)



(b) Delta Speed Index (s)

Figure 6

users adopt these tools, there is a potential to create a virtuous cycle where website developers optimize their content to be more efficient, leading to faster and more reliable web browsing for everyone.

7.2 Bridging the Digital Divide

The potential impact of Falcon Browser on users in developing countries cannot be overstated. As more people in these regions gain access to the internet, the need for efficient and affordable browsing tools becomes increasingly important. With its ability to simplify web pages and block unnecessary scripts, Falcon Browser has the potential to provide a faster and more reliable browsing experience for users with limited resources.

Furthermore, the privacy-centric design of Falcon Browser, built on top of DuckDuckGo, can help protect users in developing countries who may be more vulnerable to online privacy violations. In many of these countries, governments and other entities often monitor online activity and limit access to certain websites or services. By using a browser that prioritizes privacy, users can browse the web with more confidence and security.

In addition to its potential benefits for users in developing countries, Falcon Browser can also contribute to a more sustainable web. With its ability to reduce data consumption and improve page load times, Falcon Browser can help reduce the carbon footprint associated with web browsing. This is particularly relevant given the increasing concern about the environmental impact of technology and the need for more sustainable practices in all areas of life.

7.3 Further Improvements

Although Falcon Browser presents a promising solution to improving the web browsing experience for users in developing countries, there is still much work to be done in order to refine and enhance its technology. One of the key first steps to take is to conduct user testing in order to better understand the user experience and their overall impression of the browser. This can help identify areas of improvement and highlight specific features that users find most useful or

frustrating. Additionally, a server should be set up to receive feedback from users in case of website breakage. This can help ensure that users have a seamless and reliable browsing experience.

Furthermore, there is a need for additional research to investigate the performance of Falcon Browser on different types of networks and devices. Although the current study focused on 3G networks, it is important to evaluate the browser's performance on LTE, 4G, and other types of networks. Similarly, the study used a limited number of devices, and there is a need to test the browser on a wider range of devices to assess its performance on different hardware configurations. By testing the browser on a wider range of devices, we can better understand its performance and identify any potential compatibility issues that may need to be addressed.

Another area for future research is to evaluate the impact of Falcon Browser on web content creators and website developers. With the potential for Falcon Browser to reduce data consumption and improve page load time, there may be an opportunity for developers to optimize their websites for more efficient performance. However, this may require additional development resources and testing to ensure that websites remain functional and user-friendly for all users. Therefore, it is crucial to investigate the impact of Falcon Browser on website development and ensure that its use does not negatively affect the quality of the web content.

Finally, it is important to address the broader issue of the digital divide and unequal distribution of access to technology and the internet. While Falcon Browser has the potential to improve the browsing experience for users in developing countries, there is a need for greater investment in improving infrastructure and reducing the cost of mobile devices and data plans. This will help to ensure that all users, regardless of their location or economic status, have equal access to technology and the internet. Without addressing these systemic issues, the potential benefits of Falcon Browser and similar technologies may be limited in their ability to bridge the digital divide and improve the lives of people in developing countries.

8 CONCLUSION

This study proposes a solution to address the challenges faced by users in low-resource settings who rely on low-end mobile devices and slow internet connections. The study introduces a machine learning-driven browser that is designed to automatically create lightweight versions of web pages by eliminating non-critical JavaScript. Drawing from previous research on SlimWeb, the proposed browser utilizes a supervised machine learning model to classify JavaScript elements on a web page and block or preserve them based on predetermined metrics. The goal of this approach is to enhance the browsing experience of users who face limitations in terms of device capabilities and network infrastructure.

This study showcases the potential of machine learning-driven solutions to improve the performance of mobile web pages and enhance the user experience for individuals with limited resources. By reducing the amount of data consumed and improving page loading times, the proposed browser can provide a more efficient browsing experience for users in low-resource settings. This approach represents a significant step towards bridging the digital divide and providing more equitable access to the internet.

Moreover, the study highlights the importance of further research in this area to explore the broader potential of machine learning-driven approaches in addressing the challenges faced by users in low-resource settings. There is a need to investigate the impact of such solutions on user engagement, satisfaction, and security, as well as the potential implications for website developers and content creators. Additionally, further research is needed to test the performance of such solutions on different types of networks and devices to ensure that they are scalable and adaptable to different settings.

REFERENCES

[1] [n.d.]. UglifyJS. https://lisperator.net/uglifyjs/

[2] 2022. The Growth of Web Page Size - KeyCDN Support. https://www.keycdn.com/support/the-growth-of-web-page-size.

[3] 2022. State of JavaScript. https://httparchive.org/reports/state-of-javascript

[4] 2023. Home. https://duckduckgo.com/app

[5] Moumena Chaqfeh, Muhammad Haseeb, Waleed Hashmi, Patrick Inshuti, Manesha Ramesh, Matteo Varvello, Fareed Zaffar, Lakshmi Subramanian, and Yasir Zaki. 2021. To Block or Not to Block: Accelerating Mobile Web Pages On-The-Fly Through JavaScript Classification. https://doi.org/10.48550/ARXIV.2106.13764

[6] Moumena Chaqfeh, Yasir Zaki, Jacinta Hu, and Lakshmi Subramanian. 2020. JSCleaner: De-Cluttering Mobile Webpages Through JavaScript Cleanup. In *Proceedings of The Web Conference 2020*. Association for Computing Machinery, New York, NY, USA, 763–773. https://doi.org/10.1145/3366423.3380157

[7] Facebook. 2015. Instant articles. https://www.facebook.com/formedia/tools/instant-articles

[8] Google. 2019. AMP on google | google developers. https://developers.google.com/amp

[9] GSMA. 2021. https://www.gsma.com/r/wp-content/uploads/2021/09/The-State-of-Mobile-Internet-Connectivity-Report-2021.pdf

[10] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2019.23386

[11] Alexander Sjösten, Peter Snyder, Antonio Pastor, Panagiotis Papadopoulos, and Benjamin Livshits. 2020. Filter list generation for underserved regions. https://doi.org/10.1145/3366423.3380239

[12] Antoine Vastel, Peter Snyder, and Benjamin Livshits. 2018. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of

Crowdsourced Ad Blocking. arXiv:1810.09160 http://arxiv.org/abs/1810.09160

[13] W3Techs. [n.d.]. Usage statistics of JavaScript as client-side programming language on websites. https://w3techs.com/technologies/details/cp-javascript

[14] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2013. Demystifying Page Load Performance with WProf. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX Association, Lombard, IL, 473–485. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/wang_xiao