Towards bridging the digital divide in developing regions

Moumena Chaqfeh[a], **Rohail Asim**[a], **Bedoor AlShebli**[a], **Muhammad Fareed Zaffar**[b], **Talal Rahwan**[a], **and Yasir Zaki**[a,1]

[a]New York University Abu Dhabi, UAE; [b]Lahore University of Management Sciences, Pakistan

This manuscript was compiled on November 25, 2022

The World Wide Web empowers people in developing regions by eradicating illiteracy, supporting women, and generating economic opportunities. However, their reliance on limited bandwidth and low-end phones leaves them with a poorer browsing experience compared to privileged users across the digital divide. To evaluate the extent of this phenomenon, we sent participants to 56 cities to measure the cost of mobile data and the average page load time. We found the cost to be orders of magnitude greater, and the average page load time to be four times slower, in some locations compared to others. Analyzing how popular webpages have changed over the past years suggests that they are increasingly designed with high processing power in mind, effectively leaving the less fortunate users behind. Addressing this digital inequality through new infrastructure takes years to complete and billions of dollars to finance. A more practical solution is to make the webpages more accessible by reducing their size and optimizing their load time. To this end, we developed a solution called Lite-Web, and evaluated it in the Gilgit-Baltistan province of Pakistan, demonstrating that it transforms the browsing experience of a Pakistani villager using a low-end phone to almost that of a Dubai resident using a flagship phone. A user study in two high schools in Pakistan confirms that the performance gains come at no expense to the pages' look and functionality. These findings suggest that deploying Lite-Web at scale would constitute a major step towards a World Wide Web without digital inequality.

world wide web | digital divide | low-end mobile phone

The World Wide Web (WWW) was envisioned as an egalitarian platform that provides universal access to the wealth of accumulated human knowledge. It has enabled the creation of projects such as Wikipedia, Khan Academy, and Massive Open Online Courses (MOOCs), all of which hold the promise of democratizing education (1). In developing regions, the WWW has contributed to women's empowerment by offering a gender-opaque medium that alleviates bias, provides access to distance learning and employment opportunities, and increases the chances of receiving support from organizations concerned with the well-being of women (2, 3). Another way in which the WWW supports the developing regions is by generating economic opportunities. For example, it has been shown that fast Internet access can decrease (un)employment inequality in Africa (4), and mobile broadband access can decrease poverty, particularly among rural households (5). Additionally, the development of e-commerce can play a significant role in narrowing the urban–rural income gap in China (6). Access to critical information empowers farmers and fishermen in emerging markets. For example, by tracking weather conditions and comparing wholesale prices, farmers and fishermen in India increased their profit by 8%, eventually leading to a 4% decrease in prices for their customers (7). The WWW is even helping eradicate illiteracy—one of the main barriers to digital inclusion. For example, in Sub-Saharan Africa, where most people do not own any books, the massive proliferation of mobile devices allows people to develop, sustain and enhance their literacy skills by providing a medium through which they can access reading materials (8). Moreover, providing access to online material in Malawian boarding schools can encourage reading and improve educational outcomes (9). Other examples include non-profit initiatives such as `remoteStudentExchange.org`, which in the few months since its launch in January 2021 has given thousands of students in low- and middle-income countries direct access to (online) courses from the world's leading universities.

'The growing adoption of mobile phones has contributed to the significant increase in Internet access over the past decade. In 2018, nearly 300 million users were newly connected to the mobile web, and in 2019, the total number of mobile web users exceeded 3.5 billion worldwide. Of those users, 74% live in low- and middle-income countries (10), where mobile phones are the primary means of Internet access. Many of those users depend solely on mobile phones. For instance, across 18 developing countries, an average of 57% of Internet access in 2018 was carried out exclusively via mobile phones (10). A key enabler of mobile Internet adoption is affordability; not only is mobile data becoming available at lower prices (11), but also mobile phones are becoming more affordable. For instance, cheaper phones are expected to be available in Pakistan (12), India (13), and Africa (14) in the near future, as a new generation of phones is expected to be made available for only $20 (15).

Significance Statement

Developing regions suffer from poor internet connection and over-reliance on low-end phones, which violates Net Neutrality—the idea that all Internet traffic should be treated equally. We sent participants to 56 countries to measure global variation in web browsing experience, revealing significant inequality in mobile data cost and page load time. We also show that popular webpages are increasingly tailored to high-end phones, thereby exacerbating the inequality. Our solution, Lite-Web, makes webpages faster to load and easier to process on low-end phones. Evaluating Lite-Web on the ground reveals that it transforms the browsing experience of Pakistani villagers with low-end phones to that of Dubai residents with high-end phones. These findings call attention from researchers and policy makers to mitigate digital inequality.

T.R., and Y.Z. conceived the study, M.C., T.R., and Y.Z. designed the experiments and wrote the manuscript; M.C. and Y.Z. designed and implemented Lite-Web; R.A., M.C., and Y.Z. collected the data; R.A., F.Z., and Y.Z. ran the user study; B.A., T.R., and Y.Z. analyzed the results and produced the figures.

[1]Corresponding author. E-mail: yasir.zaki@nyu.edu

Although access to the mobile web is expanding in developing countries (16) to the point of surpassing access to piped water and consistent electricity (17), the user experience remains poor (18). This is part of a larger phenomenon known as the *digital divide*, which separates those with high-quality access to information and communications technologies from those with poorer alternatives (16). Our primary goal is to evaluate the extent of this phenomenon worldwide, and to explore affordable and scalable solutions that can potentially bridge the divide and alleviate the digital inequality experienced by underserved communities.

Results

To better understand the variation in web access quality across the globe, we needed to send participants to different cities spanning six continents, and have each of them access the same set of webpages (to control the browsing experience) using the exact same hardware (to control the processing power) and the same web browser (Google Chrome in our case) at the same local time (12:00 pm in our case) while being connected to a cellular network (rather than Wi-Fi) to ensure that any observed differences in average page load time are not influenced by variations in these factors. To this end, we leveraged the diversity of the student population at New York University Abu Dhabi by recruiting undergraduates traveling back to their home countries during the winter break. Each participant was handed the same low-end phone model, namely Xiaomi Redmi Go, and was asked to install a tool on their laptop, called *WebPageTest* (19), which automates web requests on that phone while recording various page load time metrics using the actual (rather than emulated) connection speed. Those web requests were for the 100 webpages that were most frequently visited worldwide at the time according to Alexa (20). We ended up recruiting students who visited 72 cities across six continents. Upon their arrival at their respective destinations, participants purchased a SIM card along with an affordable pricing plan from a local service provider, and kept the receipt which specified the total cost and the number of Gigabytes provided by the plan. After that, they connected the phone to their laptop via a USB cable, and ran the tool at 12:00 pm local time to automatically request the 100 webpages via Google Chrome on the phone and extract the results. The experiment took place in December 2019 and January 2020. Students who failed to follow the experimental protocol were discarded from our analysis, yielding a total of 56 cities.

Following common practice in economics (21), we use the purchasing power parity (PPP) in each country as an exchange rate to convert the value of 1 Gigabyte in their local currency to their equivalent value in USD, thereby reflecting the difference in the standard of living between countries. Fig. 1a summarizes the results of our experiment, where circles correspond to locations, colors represent average page load time, and diameters represent adjusted costs per Gigabyte. As can be seen, there is a clear digital inequality across the globe. The average page load time in some locations is four times longer than in others (about 47 vs. 12 seconds), and the cost per Gigabyte is orders of magnitude greater than in others ($43 vs. $0.08); see Supplementary Table 1 for numeric values. Similar results were obtained when using direct conversion rates (Supplementary Fig. 1) and when using gross domestic product (GDP) in purchasing power parity (PPP) for each country (Supplementary Fig. 2). To facilitate the comparison between the different locations, we plotted the distribution of the cost of one Gigabyte per location (Fig. 1b) as well as the distribution of the page load time per location, averaged over different webpages (Fig. 1c). Indeed, these distributions highlight the inequality between the locations. Moreover, to understand how the webpages themselves differ in terms of their complexity, we plotted the distribution of the page load time (seconds) per webpage, averaged over different locations (Fig. 1d). As can be seen, the page load time differs greatly across the webpages, ranging from 3.6 to 62.6, with the mean being 20.8 (note that this is the time required to load the entire page). Since the hardware specifications can affect the page load time, all measurements were taken using the same phone model, ensuring that the specifications were unified across locations and webpages. A low-end phone—Xiaomi Redmi Go—was used to help us understand the web browsing experience of disadvantaged users; see Supplementary Table 2 for the technical specifications of this phone. Note, however, that users with high income may afford high-end phones instead, which would make the inequality even greater. To further understand the differences between cities, we started off by dividing them into quartiles based on their population size. We found the page load time (Supplementary Fig. 3) and cost per Gigabyte (Supplementary Fig. 4) to be comparable across quartiles despite the massive difference in population size, e.g., the median times were: 16.05, 17.41, 16.50, and 18.52, while the median costs are: 2.82, 2.50, 2.97, and 2.00. Furthermore, we compared capital and non-capital cities, and found the page load time to be almost identical (Supplementary Fig. 5). However, compared to non-capital cities, the cost per Gigabyte in capitals is twice as high (Supplementary Fig. 6). Interestingly, we found a positive correlation ($r = 0.46$, $p = 0.0004$, Supplementary Fig. 7) between page load time and cost per Gigabyte, indicating that those with poorer connection quality pay more, not less, than their counterparts.

Ideally, digital inequality can be eliminated by providing cheap, fast connections worldwide. Unfortunately, this would not only take years to accomplish, but would also be extremely costly, e.g., achieving universal, affordable, and good quality Internet access in Africa by 2030 would require 100 billion US dollars (22). A significantly cheaper alternative would be to make the webpages themselves "lighter", by reducing their bandwidth and processing requirements. Such a solution would be desirable even if the lighter versions were slightly different from the original pages, as long as the compromise to the user experience is minimal. However, given the myriad webpages in the WWW, it may seem infeasible to analyze them all to identify the elements that are costly (in terms of bandwidth and processing time) and non-essential to the webpage (in terms of appearance and functionality). Our key insight is to focus on JavaScript elements, which are not only computationally intensive, but are also widely-used across the WWW (23). Processing these elements is more demanding for web browsers than equivalently-sized web components (24). Moreover, the download size of these elements often represents a considerable percentage of the total download size per page (25, 26). Surprisingly, despite its ubiquity, the cost of JavaScript processing on page load time is not fully understood to date. Motivated by this observation, we went

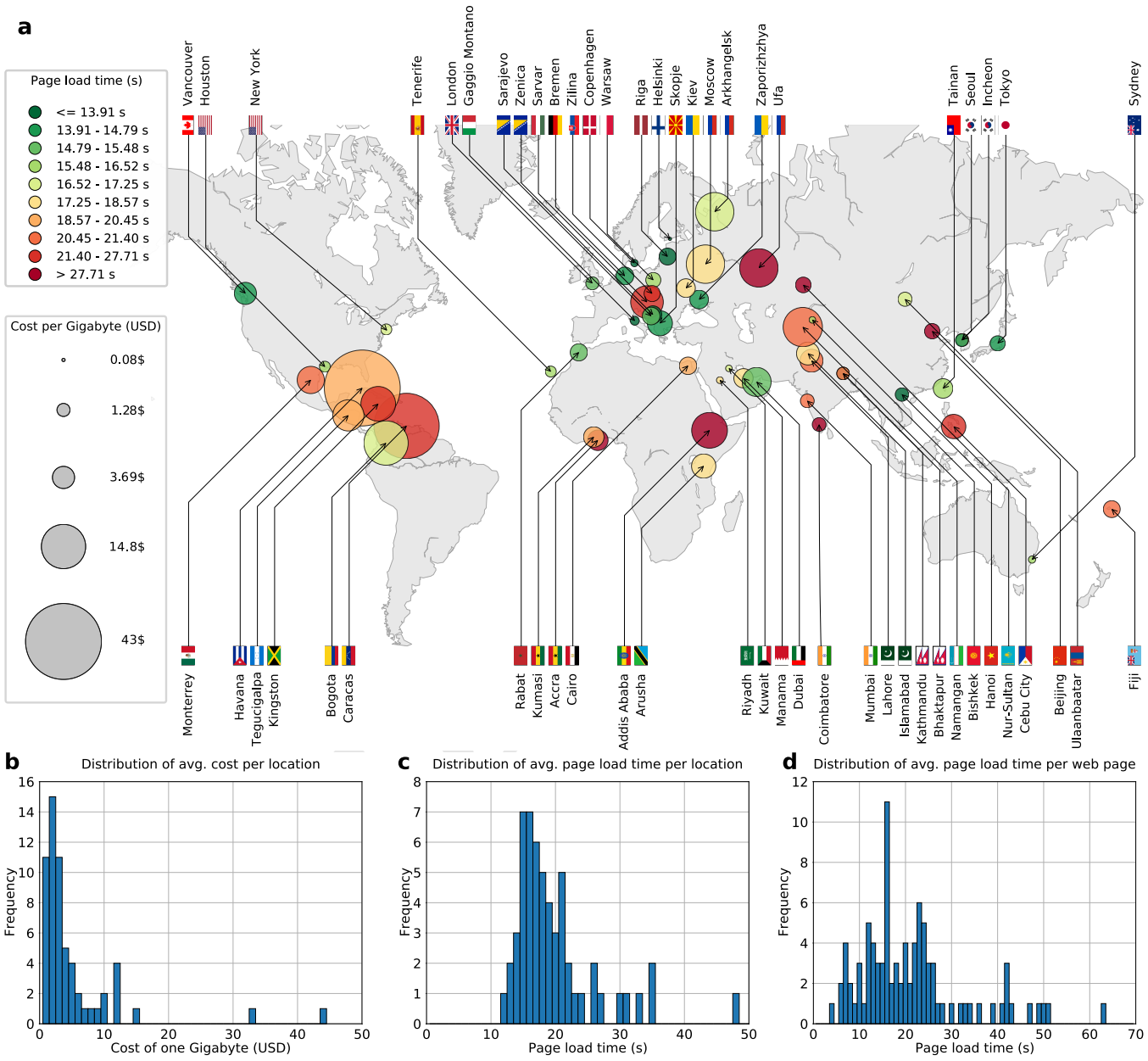


Fig. 1. Average page load time and data cost across different locations. **a**, Each location is represented by a circle whose diameter reflects the costs per Gigabyte (measured based on the purchasing power parity), and whose color represents the average page load time (measured in seconds) of the 100 most frequently visited pages worldwide. Page load times were measured by accessing the pages via the same low-end mobile phone model—Xiaomi Redmi Go—using a cellular network at that location. **b**, Distribution of the cost of 1 Gigabyte (USD) per location. **c**, Distribution of the page load time (seconds) per location, averaged over different webpages. **d**, Distribution of the page load time (seconds) per webpage, averaged over different locations.

six years back in time to understand how the processing of JavaScript affected the web browsing experience on high-end vs. low-end phones over the years. To this end, we considered the 100 webpages most frequently visited in 2019. For each page, we retrieved a version per year over the period 2015-2020 from the Internet Archive Wayback Machine (27). The pages whose versions demonstrated technical issues were filtered out, ending up with a total of 55 webpages. We cloned the retrieved versions on our own web server and ran all experiments locally on that same server. This ensures that, when comparing the performance of webpages across different phones and years, we eliminate any differences related to network connectivity, access, and servers. For each year in 2015-2020, two mobile phones released in that year were used—a low-end phone and a high-end phone—to access the webpages retrieved in that year. We set up WebPageTest (19) to record the JavaScript processing time while accessing the pages from the different phones. Fig. 2a shows the average time (in seconds) taken over the 55 webpages per year, using high-end phones (blue curve) and low-end phones (red curve); the phone models are named in the figure itself, and their technical specifications are provided in Supplementary Tables 3 and 4. As can be seen, the time spent processing JavaScript has decreased slightly on high-end phones, yet increased significantly on low-end phones over the years (from just over 2 seconds to nearly 8 seconds). Note that the increase is not due to a reduction in the processing power of the low-end phones used in our experiment; see Supplementary Table 3. This suggests that the observed increase is attributed to the webpages becoming more computationally intensive over the years. It also suggests that popular webpages are designed with high processing power in mind, neglecting the less fortunate users who can only afford low-end phones, thereby exacerbating the digital inequality. Finally, Fig. 2b shows the percentage of page load time spent on JavaScript processing. As can be seen, in the past three years, the percentage was 20% for high-end phones and nearly 50% for low-end phones.

The above results imply that users on the less fortunate side of the digital are increasingly unable to access globally popular websites. These users may instead access locally popular websites, under the assumption that such websites are designed with low-end phones in mind to cater to the local needs. To verify whether this is indeed the case, we compared the 100 websites most frequently visited globally to the 100 most popular Pakistani websites in terms of both the page load time as well as the JavaScript processing time. The evaluation was done using QMobile i6i 2020—a low-end phone manufactured by one of Pakistan's largest smartphone marketing companies. As shown in Supplementary Fig. 8, even if users in Pakistan visit local, rather than global, websites, this would not reduce the time spent processing JavaScript, which amounts to about 42% of page load time.

So far, we demonstrated that a significant percentage of page load time is spent on JavaScript processing, and this percentage is greater for users of low-end phones, regardless of whether they browse local or global websites. With this in mind, we propose a solution called Lite-Web, which focuses on producing lighter versions of webpages by optimizing the usage of JavaScript. Lite-Web is a hybrid approach, combining three of our state-of-the-art solutions, namely: *SlimWeb* (28) and *JSCleaner* (29), both of which block non-essential Javascript elements, and *Muzeel* (30), which optimizes essential JavaScript elements. Let us now provide a basic description of these three solutions; for more technical details, refer to the Methods section. Starting with *SlimWeb*, this solution is based on the idea that JavaScript elements can be classified based on their code, rather than their serving domains, as is the case with alternative commercial solutions. Relying on JavaScript code is particularly challenging since the code tends to span thousands of lines, and may include obfuscated code (which is deliberately made difficult to understand to prevent reverse engineering), machine generated code (which is often not human readable), or "uglified" code (which is generated via techniques that reduce code size at the expense of readability). By leveraging machine learning techniques, *SlimWeb* is not only capable of overcoming the above challenges, but also classifying previously unseen elements, including unknown libraries, unidentified serving domains, and obfuscated code, all of which are commonly found in today's Web. Such classification would not be possible using standard profiling techniques. As for the classes used in *SlimWeb*, they are based on the main JavaScript categories identified by experts in the web community (31). Out of these classes, *SlimWeb* blocks the following three: (1) *Advertising*, which facilitates advertisement; (2) *Analytic*, which collects data about the users; and (3) *Social*, which enables social interactions such as likes and shares. Having described *SlimWeb*, let us now move on to *JSCleaner*—the second component of our hybrid approach. Specifically, this rule-based solution is used to identify and block non-essential JavaScript elements that do not fall under any of the three classes used by *SlimWeb*. These elements are classified by *JSCleaner* as non-critical to the user experience if their code does not contain any functions that handle the page content or functionality. Finally, let us describe the third component of our hybrid approach, namely *Muzeel*. Unlike the previous two solutions, which block non-essential JavaScript elements, *Muzeel* optimizes the code of essential elements. This is done by identifying and eliminating *dead code*—parts of the JavaScript code that are never used by the webpage. One of the reasons behind the existence of such code is the use of general-purpose libraries that provide far more functionalities—and hence far more code—than what is actually required by the page. The use of such libraries is a common practice among web developers to speed up the development process, with libraries such as jQuery appearing in 83% of mobile pages worldwide (32). The identification of dead code is challenging for several technical reasons stemming from the dynamic nature of the JavaScript programming language; see the works by Chugh et al. (33) and Obbink et al. (34) for more details. *Muzeel* utilizes a novel interaction-bot that emulates how a user may interact with the page. Such an approach enables the identification of JavaScript functions that can safely be removed without affecting the user experience and the overall page content. For a technical description of the three components of Lite-Web—namely *SlimWeb*, *JSCleaner*, and *Muzeel*—see Supplementary Notes 2, 3, and 4, respectively. For an overview of related works, see the Discussion section.

To evaluate the impact of Lite-Web, we needed to run field experiments that are true to the web browsing experience in developing regions. As a first step, it was crucial to identify a location where the inhabitants' well-being is severely affected by poor Internet connectivity. Moreover, both the websites and

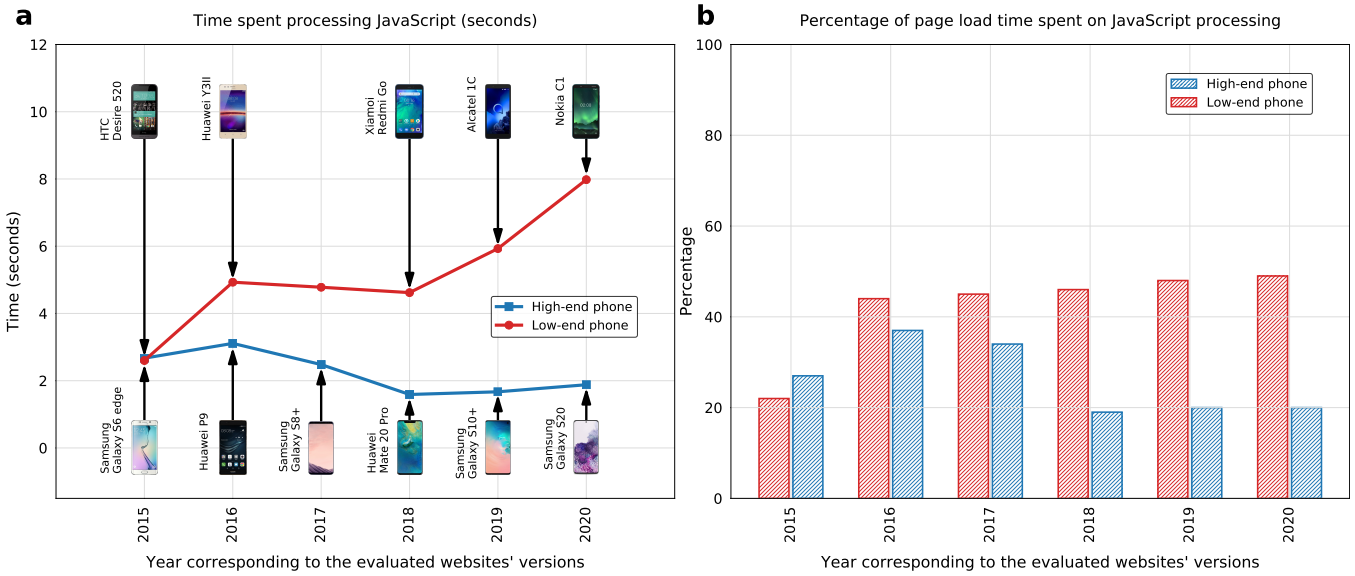


Fig. 2. JavaScript processing time, measured on high-end vs. low-end mobile devices over the past six years. For each of the 100 webpages most frequently visited in 2019, we retrieved a version per year from 2015 to 2020. The pages whose versions demonstrated technical issues were filtered out, ending up with a total of 55 webpages. For every year in 2015-2020, two mobile phones released in that year were used—a high-end phone and a low-end phone—to access the webpages retrieved in that year; the phone models are specified in the figure. **a**, Average JavaScript processing time (in seconds), measured using a high-end phone (blue curve) and a low-end phone (red curve). The data point for the low-end phone of 2017 was interpolated, since no such phone was available to purchase at the time of the study. **b**, Percentage of page load time spent on JavaScript processing, using a high-end phone (blue bar) and a low-end phone (red bar).

the mobile phones used in the experiment needed to be popular in the identified location. Finally, the participants involved in the evaluation were required to be digital natives, who regularly browse the Internet and are familiar with the local network conditions. Against these desiderata, we chose the Gilgit-Baltistan province in Pakistan, where poor Internet quality causes severe disruption to students, preventing them from keeping up with their peers. This was demonstrated by the students' protests in July 2020 demanding digital rights (35), leading to the hashtag #Internet4GilgitBaltistan becoming the second-highest ranked on Twitter in Pakistan (36). As for the mobile phone on which the experiments are conducted, we chose the same low-end phone used earlier, namely QMobile i6i 2020, since it is manufactured by a popular Pakistani company. Finally, we used the Tranco-list (37) to retrieve the 100 Pakistani webpages that were most frequently visited in 2021. Now, we are ready to evaluate the impact of Lite-Web both quantitatively (using automated measurements) and qualitatively (through a user study).

Starting with the quantitative evaluation, we sent two teams to four different locations within the Gilgit-Baltistan to measure the impact of Lite-Web based on four evaluation metrics: page load time, Speed Index, page size, and JavaScript processing time. More specifically, the four locations are Taus, Hundur, Sherqilla, and Puniyal, all marked on the map in Supplementary Fig. 9. The measurements were conducted using the WebPageTest framework (19), where the QMobile i6i mobile phone was controlled through a laptop to automatically launch both the original and the Lite-Web versions of each of the 100 Pakistani webpages. This experiment was repeated three times to account for any subtle variations that may arise when the same webpage is visited multiple times. As a result, we ended up with a total of 1,200 visits (4 locations $\times$ 100 webpages $\times$ 3 visits).

Fig. 3a depicts the impact of Lite-Web on page load time—a measure representing the elapsed time from initiation (when the user types in the Web address) to completion (when the page is fully loaded). As shown in the figure, the reduction in page load time across the four locations is 68% (in Taus), 43% (Hundur), 72% (Sherqilla), and 64% (Puniyal), with the average time reduced from 61 to 23 seconds. To determine whether this improvement is sufficient to bridge the digital divide, we compared Lite-Web's outcome to what the people of Gilgit-Baltistan would experience if they were browsing the same 100 Pakistani pages in a developed region (Dubai) on a high-end phone (Samsung Galaxy S20+) using a superior cellular network connection (4G+). As can be seen in Fig. 3a, the additional waiting time that users in Gilgit-Baltistan would suffer compared to their privileged counterparts is reduced from 48 seconds (average difference between yellow bars and pink bar) to just 10 seconds (average difference between blue bars and pink bar), amounting to an overall reduction of about 80%. Fig. 3b corresponds to the second performance metric, namely Speed Index, which measures the time taken for the contents of a page to be visibly populated and displayed to the user. Again, the use of Lite-Web results in a significant improvement across all four locations, reducing the gap between developed and developing regions by about 70%. Fig. 3c depicts the impact of Lite-Web on the time spent processing JavaScript. As can be seen, the time drops by an average of 54% across locations, and the gap between Gilgit-Baltistan and Dubai drops by about 80%. Fig. 3d shows how the size of different webpages is reduced by Lite-Web. Specifically, the page size averaged across webpages and locations is reduced by about 50% (from 0.54 to 0.28 megabytes). Notice that the average page size in Gilgit-Baltistan (without Lite-Web's improvements) is slightly smaller than Dubai's. This is because high-end phones request bigger size images compared to low-

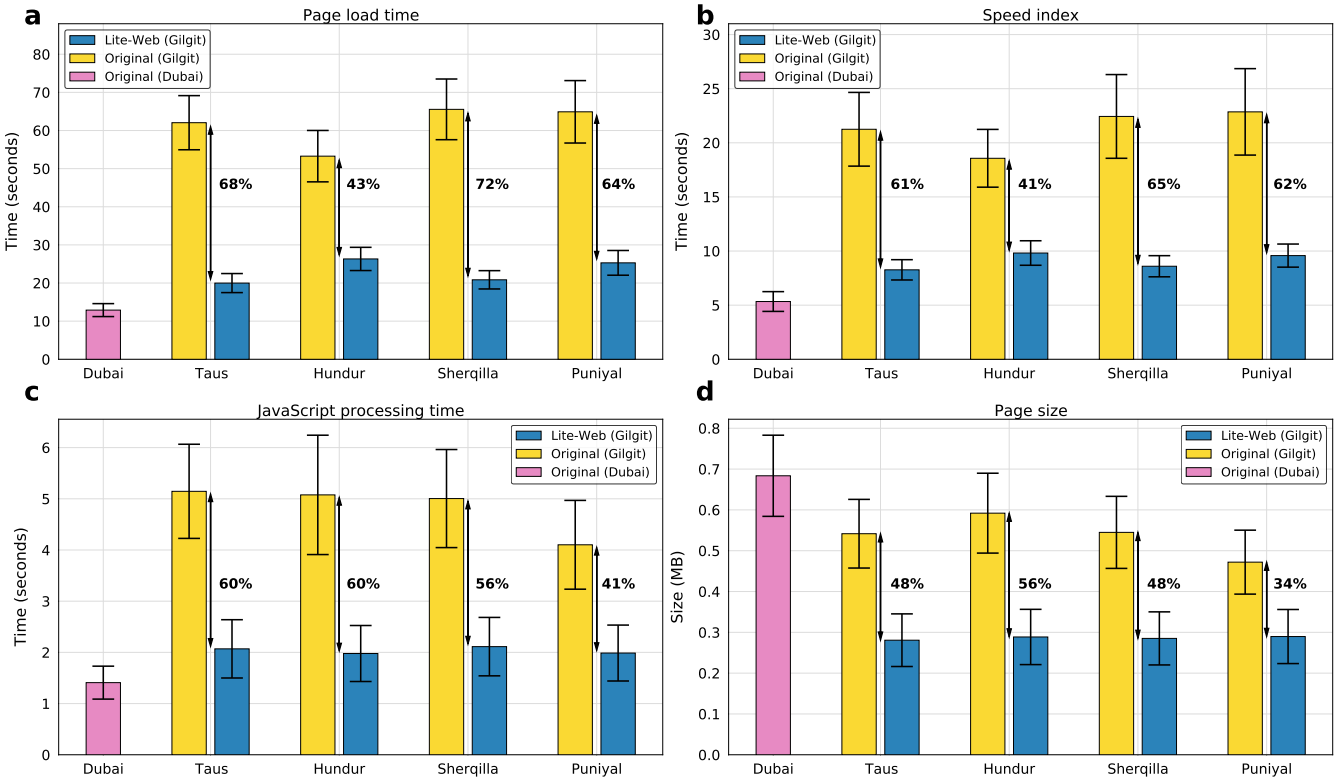


Fig. 3. Quantitative evaluation of Lite-Web. Using the 100 most frequently visited Pakistani webpages in 2021 to evaluate Lite-Web in four locations situated in the Gilgit-Baltistan province—namely Taus, Hundur, Sherqilla, and Puniyal. The evaluation is done by comparing the Lite-Web version (blue bar) to the original version (yellow bar) on the same low-end phone (QMobile i6i 2020) under the same cellular network conditions (SCOM 4G). Additionally, both the original and the Lite-Web versions are compared to a baseline (pink bar) whereby the same 100 webpages are running on a high-end phone (Samsung Galaxy S20+ 2020) under a cellular network in Dubai (Etisalat 4G+). Error bars represent the 95% confidence intervals. **a**, Evaluating page load time. **b**, Evaluating Speed Index. **c**, Evaluating JavaScript processing time. **d**, Evaluating page size.

end alternatives. However, after using Lite-Web, the webpages become smaller than those downloaded in Dubai by about 60%. Finally, we evaluated the impact of each of Lite-Web's constituent parts, namely SlimWeb, Muzeel, and JSCleaner. As shown in Supplementary Figures 10 to 13, SlimWeb is the most impactful in terms of the time-based metrics (page load time, Speed Index, and JavaScript processing time), while SlimWeb and Muzeel have a comparable impact in terms of page size reduction.

We compared Lite-Web to two state-of-the-art industry solutions that are widely deployed, namely Opera Mini (38) and Brave (39). In particular, Opera Mini sends users' webpage requests to their proxy server, where the pages are first requested and then compressed before being sent back to the user in order to reduce the transfer size and speed up the browsing experience. It is estimated that Opera Mini has about 170 million users (40). However, Opera Mini is prone to breaking interactive sites that rely heavily on JavaScript. Brave, on the other hand, is a privacy-focused browser, which automatically blocks online advertisements and website trackers in its default settings. As of December 2021, Brave has more than 50 million monthly active users, and 15.5 million daily active users (41). Similar to the evaluation done earlier, we wanted to compare Lite-Web to these two state-of-the-art industry solutions based on four evaluation metrics: page load time, Speed Index, page size, and JavaScript processing time. The measurements were conducted in the city of Lahore in Pakistan using the WebPageTest framework (19), which controlled the QMobile i6i mobile phone to automatically launch the Lite-Web, Opera Mini, and Brave versions for each of the 100 most popular Pakistani webpages. This experiment was repeated three times to account for any subtle variations that may arise when the same webpage is visited multiple times. Note that the results for Opera Mini are only depicted for the page load time and the Speed Index, since the webpagetest framework was unable to collect the remaining two evaluation metrics. The results this evaluation are depicted in Supplementary Fig. 14. As can be seen, Lite-Web achieves improvements ranging between 24% to 57% depending on the benchmark and the evaluation metric.

To assess whether the above improvements come at the expense of the page look or functionality, we recruited 200 local students from two high schools in the Gilgit-Baltistan province. Those students were randomly assigned to control and treatment groups of equal sizes. After that, the 100 Pakistani webpages were assigned to the students as follows: the webpages were divided into 25 disjoint, exhaustive, and equally-sized lists. Then, each list was assigned to 4 randomly chosen students from the control group (who interacted with the original versions of the webpages), as well as 4 randomly chosen students from the treatment group (who interacted with the Lite-web versions). All participants interacted with their assigned versions for 15 minutes using the same low-end phone model (QMobile i6i) equipped with a cellular data connection. Importantly, none of the participants knew the purpose of the study nor the group to which they belonged. This was done to

minimize the risk of subject bias, whereby participants tend to behave according to what they believe the experimenter wants to see. The study was conducted by a CITI-trained (42) person following Institutional Review Board (IRB) approval (HRPP-2021-32) from New York University Abu Dhabi. Furthermore, a letter of approval was obtained from the school principal to conduct the study on the school premises. Social distancing measures were observed, and participants were asked to wear masks throughout the study; see Supplementary Fig. 15. As a token of our appreciation, we donated twelve QMobile QTab v7 Pro tablets to the schools' libraries to be used for educational purposes.

The results of the user study are summarized in Fig. 4. Specifically, the left panel of Fig. 4a summarizes the users' evaluation of the webpages' appearance. This shows no significant difference between the control and treatment groups, In other words, we found no evidence indicating that the performance gains attributed to Lite-Web come at the expense of appearance. Similar results were observed when accounting for the gender (Supplementary Fig. 16) and age (Supplementary Fig. 17) of participants. The right panel of Fig. 4a focuses on the users (in both the control and treatment) who noticed something missing in terms of appearance; those users were asked to assess the impact of the missing components on the browsing experience. As can be seen, the treatment looks very similar to the control, with the only difference being two additional participants (out of 100) who indicated a slight impact of the missing components, and four additional participants who indicated no impact. Fig. 4b is similar to Fig. 4a except that it evaluates the impact of Lite-Web on the webpages' functionality rather than appearance. Again, the left panel shows no significant difference between the control and treatment. In other words, we found no evidence that Lite-Web's performance gains come at cost to functionality. Accounting for users' gender (Supplementary Fig. 18) and age (Supplementary Fig. 19) reveals similar trends. The right panel of Fig. 4b focuses on the few participants who noticed something missing in terms of functionality. Five additional users (out of 100) in the treatment group indicated a slight to moderate impact, and three additional users in the control group indicated a high impact. Finally, after participating in the study, all 200 students were asked to indicate the degree to which they agree with the following statement: *I occasionally avoid visiting certain websites because my Internet is too slow to load them.* Fig. 4c depicts the distribution of the responses, showing that the majority (70%) agree (somewhat or strongly) with the statement. These findings suggest that students in the Gilgit-Baltistan province are excluded from certain webpages because of being on the less fortunate side of the digital divide. More broadly, these results suggest that people in developing regions are in need of solutions such as Lite-Web to empower them to reach otherwise practically unreachable parts of the World Wide Web. As a sensitivity analysis, we repeated the same experiment but with a few modifications. First, we divided the 100 websites based on deciles, and randomly picked a website from each part, resulting in just 10 websites. Second, we recruited students from Lahore University of Management Sciences. Third, we recruited 800 participants and asked each of them to evaluate all 10 webpages, resulting in 800 evaluations per webpage. The evaluation yielded broadly similar results; see Supplementary Fig. 20.

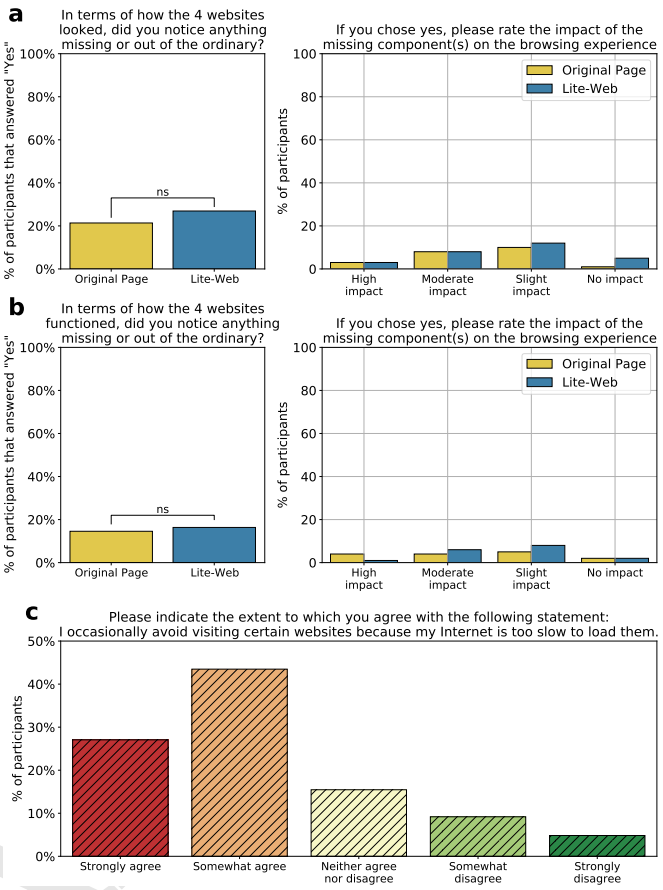


Fig. 4. High school students' evaluation of Lite-Web's impact on the appearance and functionality of websites. Each participant interacted with 4 of the 100 Pakistani websites most frequently visited in 2021; the control and treatment groups interacted with the original and Lite-Web versions of these websites, respectively. **a,** Left panel: Percentage of participants who answered "Yes" to the question: "*In terms of how the 4 websites looked, did you notice anything missing or out of the ordinary?*" (ns = not significant; $p = 0.42$); those who answered "Yes" were subsequently asked: "*If you chose yes, please rate the impact of the missing component(s) on the browsing experience*"; the distribution of their responses is depicted in the right panel. **b,** Similar to (**a**) but for questions asking about how the websites functioned, rather than how the websites looked (ns = not significant; $p = 0.85$). **c,** Responses of all participants (control and treatment) to the question: "*Please indicate the extent to which you agree with the following statement: I occasionally avoid visiting certain website because my Internet is too slow to load them.*"

Discussion. Our goal was to understand the extent of the digital divide phenomenon worldwide, and propose a scalable and affordable solution that can potentially alleviate it. We measured the mobile data cost and page load time in 56 cities, and found evidence of digital inequality across the globe. In particular, we found the cost of one Gigabyte in some locations to be orders of magnitude greater than in others, and the average page load time to be four times as long. Crucially, in each location, the results were averaged over the same 100 webpages, and the measurements were taken using the same low-end phone model, to unify the experimental setup across locations. An interesting avenue for future work would be to scale up this experiment, covering more areas within countries and over time, to chart the digital divide.

In an attempt to identify a solution that can bridge the digital divide, we focused on JavaScript elements, which are more computationally intensive than any other equally-sized

web component. Specifically, we studied how the above 100 webpages have changed from 2015 to 2020, and found that the time spent processing JavaScript has remained largely the same on high-end phones, but has increased significantly on low-end phones over the years. This suggests that webpages are designed with high processing power in mind while neglecting the less fortunate users who can only afford low-end phones, thereby exacerbating the digital inequality. More importantly, we found that a significant percentage of page load time is spent on JavaScript processing, and this percentage is greater for users of low-end phones. Motivated by this key observation, we proposed a solution called Lite-Web, consisting of three novel algorithms designed specifically to optimize the usage of JavaScript elements in today's web. We evaluated Lite-Web across four locations in a province of Pakistan known for its poor Internet connectivity, namely Gilgit-Baltistan. The evaluation focused on the 100 most popular Pakistani pages, and was done using a locally manufactured low-end phone. This demonstrated Lite-Web's ability to substantially reduce the size and loading time of webpages, thereby effectively transforming the local browsing experience to that of Dubai's residents who can afford flagship phones with fast Internet connections. Importantly, this transformation comes at no expense to the look and functionality of webpages, as evidenced by the user study conducted in two high schools in the region. However, given that Lite-Web blocks ads, it can reduce the revenue of the content providers, and may disadvantage companies in developing regions as they can no longer advertise their services to the users. Having said that, it should be noted that Lite-Web is not the only solution that blocks ads and analytics. In fact, one of the main features of the "Brave Browser" (39)—a very successful modern browser, with more than 50 million monthly active users and 15.5 million daily active users—is to block ads and trackers. Moreover, ads constitute only one of the categories blocked by SlimWeb, which in turn constitutes only one of three components of Lite-Web. If need be, the ad-blocking feature of SlimWeb can be disabled, in which case the solution would still provide significant speedups to the page load time (28).

Our study comes with a number of limitations. First, when reporting the page load time and mobile data cost across cities (Figure 1), our data represents a single point-in-time snapshot of performance and price. Mobile network performance evolves rapidly, both due to network upgrades as well as increased usage of infrastructure, but these factors are not considered in our analysis. Similarly, we do not consider the role of policy and competitive factors that drive the data cost. Moreover, although participants were instructed to purchase a plan they considered to be affordable, this plan is not representative of the entire spectrum of plans available in their respective city. Having said that, our experiment facilitates a comparison across cities since the price was deemed affordable by an undergraduate student who came from that city (in addition to the experimental protocol which controlled for processing power, web browser, pages visited, connection medium, and time of day). As such, the analysis in Figure 1 provides evidence of digital inequality across cities, but should not be interpreted beyond that.

Over the past decade, expanding Internet access has become a target for international advocacy efforts from the United Nations, and many solutions have been proposed to provide affordable, high-quality connection to everyone. However, such efforts rely on critical infrastructure that would require years to build and hundreds of billions of US dollars to fund (43). A significantly cheaper alternative is to make the webpages lighter for developing regions. Surprisingly, this alternative has only just started gaining attention. For example, Facebook has introduced a solution called Facebook Lite (44) for Android users with limited connectivity and low-end phones. However, this solution is designed solely for Facebook. Another initiative is Google's Accelerated Mobile Pages (AMP) (45), which provides a framework that can assist web developers in creating lighter versions of their webpages. Unfortunately, AMP does not consider existing webpages, but rather requires the creation of new ones from scratch. This makes it hard to deploy on a massive scale, especially given the billions of webpages already present in the WWW. From the developer's perspective, one way to reduce the size of JavaScript files before they are embedded into the page is to use uglifiers (46, 47). These rely on removing non-essential characters such as white spaces and newlines from JavaScript files to improve transmission efficiency. However, unlike our Lite-Web solution, uglifiers do not reduce JavaScript processing time—a major contributor to the digital inequality, as our experiments have shown. From the user's perspective, several JavaScript blocking tools (48–52) can be used to reduce the amount of JavaScript transferred to their browsers. However, these tools are restricted to a predetermined block-list, and are not equipped with any sort of intelligence that can automatically classify previously-unseen JavaScript elements to determine whether they should be blocked. A very recent solution called Percival (53) has shown promising results in blocking ads using deep learning. It intercepts images obtained during page execution to flag potential ads. However, this solution is computationally intensive, resulting in a non-negligible performance overhead on desktop PCs. As such, it cannot be applied on low-end mobile phones with limited computational power. In a recent work (54), the authors proposed WebMedic—a method to remove less-useful (rather than entirely unused) functions from the page. They found that 20% of the memory can be saved for the majority of webpages while preserving 80% of the functionality. However, further research is needed to maximize the speedup while minimizing the impact on the page functionality. Other ways to improve the browsing experience are offered by platform-based solutions. For instance, Apple News (55) is a news aggregator app developed exclusively for Apple mobile devices, whereas Instant Articles (44) is a tool that allows publishers to create fast and interactive content on Facebook. However, such solutions are narrow in scope, and are not generalized to all devices and/or all webpages. We saw how the gap between high-end and low-end phones has increased over the past five years in terms of JavaScript processing. If this trend continues without any interventions, it would lead to a segregation of disadvantaged and advantaged users, whereby the former are practically unable to access the webpages that cater to the latter. Such segregation would violate the *Net neutrality* principle (56), which requires treating all Internet traffic equally, without discriminating or charging differently based on user, content, website, location, type of equipment, or access medium. Our findings call for attention from researchers and policymakers alike, to mitigate disparity and adhere to the net neutrality principle across the globe. More

broadly, Internet connectivity has arguably become a basic human right in the twenty-first century, and the emerging literature on reducing web complexity (34, 39, 44, 45, 54, 57–63) constitutes a promising step towards realizing the United Nation's vision "to ensure that digital technologies are built on a foundation of respect for human rights and provide a meaningful opportunity for all people and nations" (16).

Data Availability. Our data were collected from several experiments that we ran: a) in the wild page load times and cost collected from 56 cities around the world, b) in lab experiments on JavaScript processing times over past six years, and c) in the wild quantitative evaluation of Lite-Web from two schools in Pakistan.

The whole data will be shared upon publication under the following repository https://github.com/comnetsAD/digital-divide.

Materials and Methods

Our proposed Lite-Web solution combines three novel algorithms that we developed to reduce the processing cost of JavaScript in today's webpages, namely SlimWeb (64), JSCleaner (29), and Muzeel (30). For a given webpage, Lite-Web first runs SlimWeb's machine learning classifier to identify and block JavaScript elements that are non-essential to the user experience; see Supplementary Note 2 for more details. A user study (64) showed that, in order to achieve faster browsing, people are willing to sacrifice parts of the page that are responsible for: (1) advertising, (2) analytics, and (3) social interactions. Based on this finding, all JavaScript elements belonging to the above three categories are blocked by Lite-Web. Additionally, Lite-Web runs a modified version of the rule-based classification used by JSCleaner to identify and block JavaScript elements that are non-critical to the page content or interactive functionality. More details on how Lite-Web modifies JSCleaner's rules can be found in Supplementary Note 3. So far, Lite-Web preserves JavaScript elements that are identified as essential by SlimWeb and the modified JSCleaner rules. By analyzing these preserved elements, we found many of them to be large JavaScript libraries that are incorporated wholly into the page, even though only a few functions of these libraries are utilized (30). This key observation suggests that optimizing webpages can go beyond eliminating non-essential JavaScript elements, by optimizing the essential ones. This optimization is done through the elimination of functions that are included in the essential elements yet not used by the webpages. This is precisely what Muzeel is designed to do. The elimination of unused functions provides data cost savings (since the files containing such functions are often quite large), as well as performance improvements (60) (since the number of functions that require processing is now reduced). Further details on how Muzeel operates can be found in Supplementary Note 4. When evaluating Lite-Web in Gilgit-Baltistan, we deployed Lite-Web in a cloud server hosted in Pakistan. This server maintained a database of JavaScript elements extracted from the 100 most popular Pakistani webpages, labeled by SlimWeb and JSCleaner as either essential or non-essential. The server caches a modified version of the essential ones, which is stripped out of any unused functions by Muzeel. The phones' browsers were configured to utilize our Lite-Web server as a web proxy. As such, JavaScript requests are either deemed non-essential by the proxy and subsequently blocked, or deemed essential, in which case the Muzeel'ed versions of these elements are sent back from the server cache. All other web elements' requests, apart from JavaScript, are served live from the Internet.

ACKNOWLEDGMENTS. We thank the Aga Khan Higher Secondary School management for granting us permission to run a user study on their premise, and thank the students who participated in this study, the results of which are summarized in Fig. 4. We also thank Aezaz Ali, Russell Coke, Inara Kaneez, Sajid Karim, and Tauqeer Saleem, who volunteered to help us run our evaluation of Lite-Web in the Gilgit-Baltistan province, thereby generating the results depicted in Fig. 3. Finally, we thank the participants, mostly students from New York University Abu Dhabi, who participated in generating the results of Fig. 1, namely Aaysuh Deo, Adam Atallah, Aigul Saiapova, Aisha Hodzic, Alem Shaimardanov, Ali Shazal, Ananya Valli Krishnakumar, Andriy Lunin, Anthony Chua, Barkin Simsek, Bernice delos Reyes, Daniel Hawie, Denat E Negatu, Desmond Ofori Atta, Enid Mollel, Fadhl Eryani, Fiona J Lin, Fuseini Sunnuma, Gabriel Anders Kedmi Moller, Gabriel Antonio Garcia Leyva, Gautham Dinesh Kumar Lali, Hilina Bayew, Hussain AlEssa, Ivana Drabova, Jacinta Hu, Jahnae Miller, Joonha Yu, Jordan Simpson, Kenya Vazquez, Komiljon Turdaliev, Lauris Paegle, Manuel Padilla, Marcin Waniek. Mariam Elgamal, Maryam Khalili, Máté Hekfusz, Michael Liu, Miro Manino, Muhammad Shehryar Hamid, Nadja Fejzic, Natasha Treunen, Nathnael Hailu Tsegaye, Odera Ebeze, Omar Ould Ali, Oscar Gomez, Pamela Martinez, Prajjwal Bhattarai, Prajna Soni, Priyanshu Mishra, Raeed Riaz, Sangjin Lee, Sam Ixcaragua, Sampanna Bhattarai, Samridha shrestha, Sara Pan Algarra, Shivani Mishra, Sodgerel Mandakhnaran, Syed Taimur Hasan, Tai Lu, Tami Gjorgjieva, Teona Ristova, Thomas Poetsch, Tsion Gurmu, Vera Petrova, Victoria Gabriela Marcano, Vladyslav Cherevkov, Yehowahi Sekan, and Yves Teng.

1. Hansen JD, Reich J (2015) Democratizing education? examining access and usage patterns in massive open online courses. *Science* 350(6265):1245–1248.
2. (2009) Information and communication technologies for women's socioeconomic empowerment (http://documents1.worldbank.org/curated/fr/812551468148179172/pdf/518310PUB0REPL101Official0Use0Only1.pdf). Accessed: 2021-02-4.
3. (2016) World development report 2016: Digital dividends (https://www.worldbank.org/en/publication/wdr2016). Accessed: 2021-12-20.
4. Hjort J, Poulsen J (2019) The arrival of fast internet and employment in africa. *American Economic Review* 109(3):1032–79.
5. Bahia K, et al. (2020) The welfare effects of mobile broadband internet: Evidence from nigeria. *World Bank Policy Research Working Paper* (9230).
6. Li L, Zeng Y, Ye Z, Guo H (2021) E-commerce development and urban-rural income gap: Evidence from zhejiang province, china. *Papers in Regional Science* 100(2):475–494.
7. Jensen R (2007) The digital provide: Information (technology), market performance, and welfare in the south indian fisheries sector. *The Quarterly Journal of Economics* 122(3):879–924.
8. West M, Ei CH (2014) *Reading in the mobile era: A study of mobile reading in developing countries.* (UNESCO).
9. Derksen L, Leclerc CM, Souza PC, , et al. (2019) *Searching for answers: The impact of student access to wikipedia.* (University of Warwick, Centre for Competitive Advantage in the Global . . .).
10. (2019) The state of mobile internet connectivity 2019 (https://www.gsma.com/mobilefordevelopment/wp-content/uploads/2019/07/GSMA-State-of-Mobile-Internet-Connectivity-Report-2019.pdf). Accessed: 2020-10-04.
11. (2020) Making mobile internet technology more affordable (https://www.newpath.com/). Accessed: 2020-10-11.
12. (2020) Introducing the world's most affordable smart feature phone – the digit 4G (https://www.kaiostech.com/introducing-the-worlds-most-affordable-smart-feature-phone-the-digit-4g/). Accessed: 2020-10-11.
13. (2020) Jio likely to launch affordable android phones in India by dec 2020: Report (https://www.bgr.in/news/jio-launch-india-android-phone-low-cost-2020-price-more-913657/). Accessed: 2020-10-11.
14. (2020) Six new partners to deliver affordable smart feature phones running on kaios in Africa (https://www.kaiostech.com/press/six-new-partners-to-deliver-affordable-smart-feature-phones-running-on-kaios-in-africa/). Accessed: 2020-10-11.
15. (2019) A $20 phone for Africa is mwc's unlikeliest hero (https://thenextweb.com/plugged/2019/02/26/a-20-phone-for-africa-is-mwcs-unlikeliest-hero/). Accessed: 2020-10-11.
16. (2019) The age of digital interdependence (https://digitalcooperation.org/wp-content/uploads/2019/06/DigitalCooperation-report-web-FINAL-1.pdf). Accessed: 2020-10-04.
17. (2016) More Africans have access to cell phone service than piped water (https://edition.cnn.com/2016/01/19/africa/africa-afrobarometer-infrastructure-report/index.html). Accessed: 2020-10-11.
18. Steve S, James B, Tess Z, Caitlin L, Claire S (2017) Connecting the next four billion: Strengthening the global response for universal internet access (https://www.usaid.gov/sites/default/files/documents/15396/Connecting_the_Next_Four_Billion.pdf). Accessed: 2020-10-11.
19. (2019) Webpagetest - website performance and optimization test (https://www.webpagetest.org/). Accessed: 2019-09-10.
20. (1996-2020) Keyword research, competitor analysis, and website ranking | alexa (https://www.alexa.com/). Accessed: 2019-11-04.
21. Krugman PR, Obstfeld M (2009) *International economics: Theory and policy.* (Pearson Education).

22. (2019) Connecting Africa through broadband a strategy for doubling connectivity by 2021 and reaching universal access by 2030 (https://www.broadbandcommission.org/Documents/working-groups/DigitalMoonshotforAfrica_Report.pdf). Accessed: 2020-10-21.
23. Elliott E (2019) How popular is javascript in 2019? (https://medium.com/javascript-scene/how-popular-is-javascript-in-2019-823712f7c4b1). Accessed: 2020-11-08.
24. Osmani A (2018) The cost of javascript (https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4). Accessed: 2019-05-05.
25. (2020) Report: State of javascript (https://httparchive.org/reports/state-of-javascript#bytesJs). Accessed: 2020-11-08.
26. (2020) Report: State of the web (https://httparchive.org/reports/state-of-the-web#bytesTotal). Accessed: 2020-11-08.
27. (2014) Internet archive wayback machine (https://web.archive.org/). Accessed: 2020-07-21.
28. Chaqfeh M, et al. (2021) To block or not to block: Accelerating mobile web pages on-the-fly through javascript classification. *CoRR* abs/2106.13764.
29. Chaqfeh M, Zaki Y, Hu J, Subramanian L (2020) Jscleaner: De-cluttering mobile webpages through javascript cleanup in *Proceedings of The Web Conference 2020*. pp. 763–773.
30. Kupoluyi T, et al. (2021) Muzeel: A dynamic javascript analyzer for dead code elimination in today's web. *arXiv preprint arXiv:2106.08948*.
31. Archive H (2019) Third parties | 2019 | the web almanac by http archive (https://almanac.httparchive.org/en/2019/third-parties). Accessed: 2020-01-2.
32. Hearne S (2020) Third parties | 2020 | the web almanac by http archive (https://almanac.httparchive.org/en/2020/third-parties). Accessed: 2021-09-26.
33. Chugh R, Meister JA, Jhala R, Lerner S (2009) Staged information flow for javascript. *SIGPLAN Not.* 44(6):50–62.
34. Obbink NG, Malavolta I, Scoccia GL, Lago P (2018) An extensible approach for taming the challenges of javascript dead code elimination in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. (IEEE), pp. 291–401.
35. (2020) Students in Gilgit Baltistan protest as they suffer due to poor Internet quality (https://newsvibesofindia.com/students-gilgit-baltistan-protest-suffer-due-to-poor-internet-quality-28006/). Accessed: 2021-09-28.
36. (2020) #Internet4GilgitBaltistan Trends on Twitter, activists demand digital rights for Gilgit-Baltistan (https://gbee.pk/2020/07/internet4gilgitbaltistan-trends-on-twitter-activists-demand-digital-rights-for-gilgit-baltistan/). Accessed: 2021-09-28.
37. Le Pochat V, Van Goethem T, Tajalizadehkhoob S, Korczyński M, Joosen W (2019) Tranco: A research-oriented top sites ranking hardened against manipulation in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019.
38. (2006) Opera mini (https://www.opera.com/browsers/opera-mini). Accessed: 2022-04-20.
39. (2020) Brave: the privacy preserving browser (https://brave.com/). Accessed: 2022-04-20.
40. (2021) Opera launches hype, an in-browser chat service for opera mini users, in south africa, zambia and ghana (https://investor.opera.com/news-releases/news-release-details/opera-launches-hype-browser-chat-service-opera-mini-users-south). Accessed: 2022-04-20.
41. (2022) Brave passes 50 million monthly active users, growing 2x for the fifth year in a row (https://brave.com/2021-recap/). Accessed: 2022-11-11.
42. (2019) Citi program - collaborative institutional training initiative (www.citiprogram.org). Accessed: 2019-10-10.
43. (2020) Affordability report 2020 (https://1e8q3q16vyc81g8l3h3md6q5f5e-wpengine.netdna-ssl.com/wp-content/uploads/2020/12/Affordability-Report-2020.pdf). Accessed: 2021-02-14.
44. Facebook (2015) Instant articles | facebook. Accessed: 2020-03-21.
45. Google (2019) AMP is a web component framework to easily create user-first web experiences - amp.dev (https://amp.dev). Accessed: 2019-05-05.
46. Bazon M (2012) Uglifyjs (http://lisperator.net/uglifyjs/). Accessed: 2020-05-01.
47. CircleCell (2011) Jscompress - the javascript compression tool (https://jscompress.com/). Accessed: 2020-05-01.
48. Data S (2016) Sybu javascript blocker – google chrome extension (https://sybu.co.za/wp/projects/js-blocker/). Accessed: 2020-05-02.
49. Roman T (2018) Js blocker (https://jsblocker.toggleable.com/). Accessed: 2020-05-02.
50. Bauman M, Bonander R (2017) Advertisement blocker circumvention system. US Patent App. 15/166,217.
51. AdBlock (2009) Surf the web without annoying pop ups and ads (https://getadblock.com/). Accessed: 2020-05-02.
52. International C (2009) Ghostery makes the web cleaner, faster and safer (https://www.ghostery.com/). Accessed: 2020-05-2.
53. Abi Din Z, Tigas P, King ST, Livshits B (2020) Percival: Making in-browser perceptual ad blocking practical with deep learning in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. pp. 387–400.
54. Naseer U, Benson TA, Netravali R (2021) Webmedic: Disentangling the memory-functionality tension for the next billion mobile web users in *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*. pp. 71–77.
55. Apple (2015) Apple news - apple. Accessed: 2020-03-21.
56. Wu T (2003) Network neutrality, broadband discrimination. *J. on Telecomm. & High Tech. L.* 2:141.
57. (year?) Opera mini for android (https://www.opera.com/mobile/mini). Accessed: 2021-01-11.
58. Ghasemisharif M, Snyder P, Aucinas A, Livshits B (2019) Speedreader: Reader mode made fast and private in *The World Wide Web Conference*. pp. 526–537.
59. Kelton C, Varvello M, Aucinas A, Livshits B (2021) Browselite: A private data saving solution for the web. *arXiv preprint arXiv:2102.07864*.
60. Wang XS, Balasubramanian A, Krishnamurthy A, Wetherall D (2013) Demystifying page load performance with wprof in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. (USENIX, Lombard, IL), pp. 473–485.
61. Wang XS, Krishnamurthy A, Wetherall D (2016) Speeding up web page loads with shandian in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. (USENIX Association, Santa Clara, CA), pp. 109–122.
62. Netravali R, Mickens J (2018) Prophecy: Accelerating mobile page loads using final-state write logs in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. (USENIX Association, Renton, WA), pp. 249–266.
63. Netravali R, Goyal A, Mickens J, Balakrishnan H (2016) Polaris: Faster page loads using fine-grained dependency tracking in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. (USENIX Association, Santa Clara, CA).
64. Chaqfeh M, et al. (2021) To block or not to block: Accelerating mobile web pages on-the-fly through javascript classification. *arXiv preprint arXiv:2106.13764*.

Supporting Information for

Towards bridging the digital divide in developing regions

Moumena Chaqfeh, Rohail Asim, Bedoor AlShebli, Muhammad Fareed Zaffar, Talal Rahwan, Yasir Zaki

Corresponding Yasir Zaki.
E-mail: yasir.zaki@nyu.edu

This PDF file includes:

Supporting Information Text

Supplementary Note 1: Feature selection. The objective of this supplementary note is to summarize the feature selection for both SlimWeb and JSCleaner; for more details see (1) and (2). To represent each JavaScript element embedded in a webpage as a vector of features, we utilized the set of 1262 features proposed in (2). These features represent all Application Programming Interfaces (APIs) that facilitate interacting with the Document Object Model (DOM) of HTML, which is the main programming interface that defines the logical structure of HTML documents and the way they can be accessed and manipulated (3, 4), including properties, methods, and event handlers. By knowing all the APIs that a certain JavaScript element uses, we can infer the purpose of that element, since those APIs cover all the ways in which JavaScript can interact with the webpage (including reading, writing, and event-handling).

To design a complete set of features, the full list of the aforementioned APIs along with their properties, methods, and events are considered. Each feature was manually labeled according to the DOM (3) and the DOM HTML APIs specifications (4), with the objective of categorizing each of them to aid the rule-based classification in JSCleaner. It is worth noting that only the features, without their labels, are required for the machine learning classification in SlimWeb. Four possible labels were identified:

1. **Read feature:** A feature that can only be used to extract information from webpages.
2. **Read/Write feature:** A feature that can be used to read from or write to webpages.
3. **Write feature:** A feature that can only be used to modify a given page, by creating or adding new web elements to the page, or by changing existing elements in the page. For example, the "appendChild" method of the Node interface writes to the page by adding a child node to an existing parent node.
4. **Event feature:** A feature that handles a user interactivity event (such as a click) or a web event (such as "DOMContentLoaded" which fires when the initial document is completely loaded and parsed).

Duplicate features (properties or methods that are found in more than one API, such as id, name, type, width, length, and height) were removed from the final set of features due to their common utilization by the JavaScript language core (5). This resulted in a final set of 1,262 features, each with a unique label, which can be reproduced by extracting all the APIs specified in (3) and (4), along with their properties, methods, and events. The final set is used for the rule-based classification of JSCleaner. On the other hand, when using SlimWeb, we created a reduced set of 508 features for a faster on-the-fly classification for SlimWeb; this was done using recursive feature elimination with cross-validation.

Supplementary Note 2: SlimWeb. The objective of this supplementary note is to summarize the design details of SlimWeb; see (1) for further details. SlimWeb is built around a machine learning algorithm that classifies each JavaScript element embedded in a given webpage into one of eight categories. These categories are based on the best web practices defined by experts in the field (6). Specifically, the eight categories are as follows:

1. **Advertising:** JavaScript elements related to advertising and marketing.
2. **Analytics:** JavaScript elements that measure or track users by recording their actions.
3. **Social:** JavaScript elements that enable social features (such a sharing) in webpages.
4. **Video:** JavaScript elements that enable video players and manage streaming functionality.
5. **Utilities:** JavaScript elements related to developer utilities, such as API clients, site monitoring utilities, and fraud detectors.
6. **Hosting:** JavaScript elements brought by web hosting platforms (such as WordPress, Wix and Squarespace), which include publicly hosted open source libraries (such as jQuery) served over different public CDNs (Content Delivery Networks) and private CDN usage.
7. **Customer success:** JavaScript libraries brought from customer support/marketing providers that offer chat and contact solutions.
8. **Content:** JavaScript libraries brought by content providers or publishing-specific affiliate tracking, including tag management elements that tend to load other JavaScript elements and initiate specific tasks.

To train different supervised learning models, a labeled JavaScript dataset was created by crawling 20,000 popular pages, and extracting the JavaScript elements of those pages. The domain names behind these elements were cross-checked against a publicly available HTTP Archive repository (7) that lists existing JavaScript libraries along with their domain names, and their respective category from the above list. This process yielded 127,000 matches, each corresponding to a labeled (i.e., categorized) JavaScript element. This constitute SlimWeb's training set, in which each JavaScript element is represented by a vector of features, taken from Supplementary Note , as well as a label taken from the above eight categories.

Using the training set, we experimented with six supervised learning models. These consisted of three distance-based classifiers: K-Nearest Neighbors (KNN), Support Vector Machine (SVM) and Linear Support Vector Classifier (LSVC)); two tree-based classifiers: Random Forest Classifier (RFC) and XG Boost; and a simple neural network model. Next, we specify the parameterization used for each model.

Starting with distance-based classifiers, for KNN we chose the output class to be the maximum class represented in the K-nearest neighbors. We chose the value $K = 293$ since it gave the best classification performance on the training set (the evaluation metrics are described later on in this section). For SVM and LSVC, we used the standard training methodology over the entire feature space.

Out of the available tree-based classifiers, RFC was selected due to its ability to reduce over-fitting and improve accuracy. We used the Exhaustive Grid Search approach to adjust the hyper-parameters. The number of trees was set to 250 as this yielded the best accuracy. As for XG Boost, it was selected due to its superior performance in several machine learning challenges (8). We set the maximum depth to 32, since larger values did not increase the accuracy. The learning rate was set to 0.3.

Finally, we experimented with a simple neural network, and used two hidden layers since the accuracy did not improve with a greater number of such layers. The *ReLU* (Rectified Linear Unit) (9) activation function was used in the hidden layers, while the output layers utilized the *softmax* activation function. When evaluating the above models, we used the following standard performance metrics:

- Recall, which is computed as follows:

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad [1]$$

- Precision, which is computed as follows:

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad [2]$$

- F1-Score, which measures the classification accuracy by combining both the precision and recall, and is computed as follows:

$$F1\text{-}score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad [3]$$

Supplementary Table S5 presents a summary of the evaluation results for the different supervised learning models. Each value in the table represents an overall average of the corresponding metric, which is computed across all the eight different categories. The detailed evaluation of the models for each category is shown in Supplementary Fig. S21. Based on these results, we opted to use the simple neural network, as it outperformed all other alternatives.

Based on the above evaluation, Lite-Web uses a simple neural network to classify JavaScript elements into one of the eight categories outlined earlier. Then, as mentioned in the main manuscript, Lite-Web blocks the elements that are classified as *Advertising*, *Analytics*, or *Social*.

Supplementary Note 3: JSCleaner. JSCleaner employs a rule-based classification using the labeled features described in Supplementary Note to classify JavaScript elements into essential and non-essential. In Lite-Web, we modified the original version of JSCleaner which considers a third class that is referred to as replaceable JavaScript. Instead of replacing these elements with HTML as in the original JSCleaner, we consider them as essential. Using the 1,262 features described earlier, JavaScript elements are labeled by the modified version of JSCleaner according to the following rules:

1. **Essential classification rule:** If a JavaScript element contains a set of event features or writing features then it is classified as *essential*.

2. **Non-essential classification rule:** If a JavaScript element neither contains event features nor writing features then it is classified as *nonessential*.

Supplementary Note 4: Muzeel. The objective of this supplementary note is to summarize the design details of Muzeel; see (10) for further details. Muzeel automatically identifies the order of events needed to cover all possible states a webpage can reach at run-time. Based on this, it dynamically analyzes JavaScript elements to identify and eliminate their unused functions, i.e., their dead-code. This is done by considering both the page-load events and the user interactivity events of a given page. Such events are handled by "event handlers" which call specific JavaScript functions whenever events are fired. While existing approaches (11) identify only the JavaScript functions that are required for the page load events, Muzeel additionally identifies the JavaScript functions required by user events through user-interactivity emulation. Below is a detailed description of the three main phases of Muzeel, which are: pre-processing, dead-code identification, and dead-code elimination.

Pre-processing: During this phase, Muzeel modifies the functions from all JavaScript elements used in the page. This modification ensures that, whenever a function is called by an event, it outputs a unique ID to the browser's console. This way, if a function's ID was not logged to the browser's console, it indicates that this function is never used by the page and thus can be safely removed.

Dead-code identification: Muzeel loads a webpage in an automated browser to emulate the user interactivity events once the page has loaded. This way, JavaScript functions that handle page load events or user interactivity events can be logged to the browser console. A user interactivity event is associated with a *page element*, defined as an object that appears in an HTML tag within the Document Object Model (DOM) of the page (such as image, button, or navigation element). Muzeel identifies the page elements and extracts the set of all user events associated with each of them to drive the user interactivity emulation. To this end, it uses `XPaths` to uniquely identify page elements across reloads. To emulate user interactivity on a webpage, Muzeel triggers all user events using a browser automation tool. Here, the dependencies between events should be taken into consideration. For instance, some events are only successful if triggered after others. Similarly, triggering certain events may prevent the interactivity with other events. Muzeel considers such dependencies when emulating user interactivity. Further details on elements' identification and how Muzeel emulates the user interactivity can be found in (10).

Dead-code elimination: Using the browser's console logs—which contains the IDs of the JavaScript functions that were called by the page—Muzeel annotates the *used* functions. This way, functions that are never called can be removed from their respective JavaScript files. Muzeel implicitly considers nested functions that are called upon the execution of other functions due to user interactivity. Whenever a function handling a user interactivity event calls a set of nested functions, the IDs of all those functions are logged to the browser console. Additionally, the elimination of a given function leads to the elimination of all the nested ones. By following these steps, Muzeel obtains a complete trace of the used functions and an accurate identification of dead-code in a given webpage.

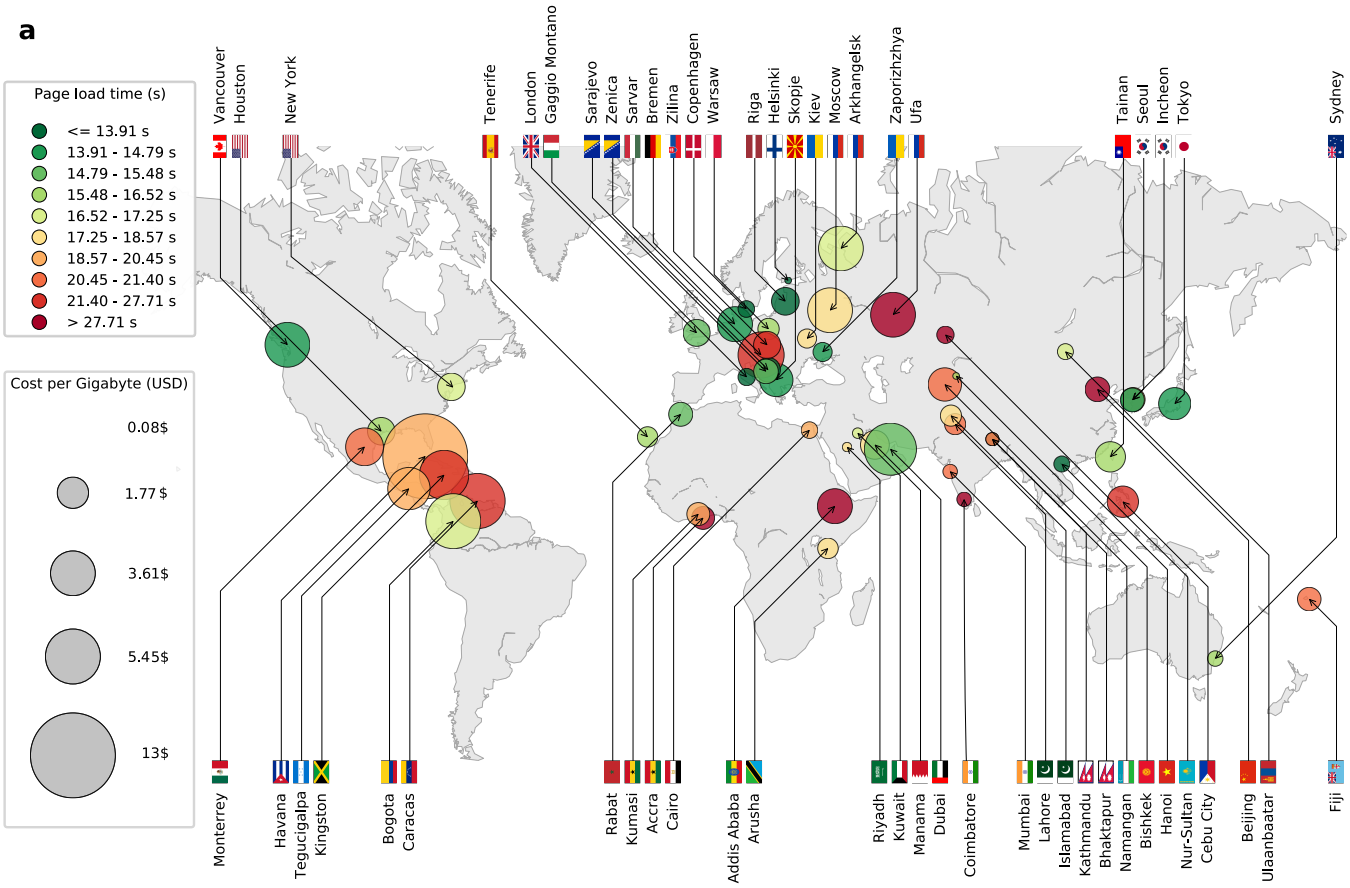


Fig. S1. Average page load time and data cost across different locations. Similar to Figure 1a in the main article, except that the cost per Gigabyte is computed based on direct conversion rate instead of using the purchasing power parity.

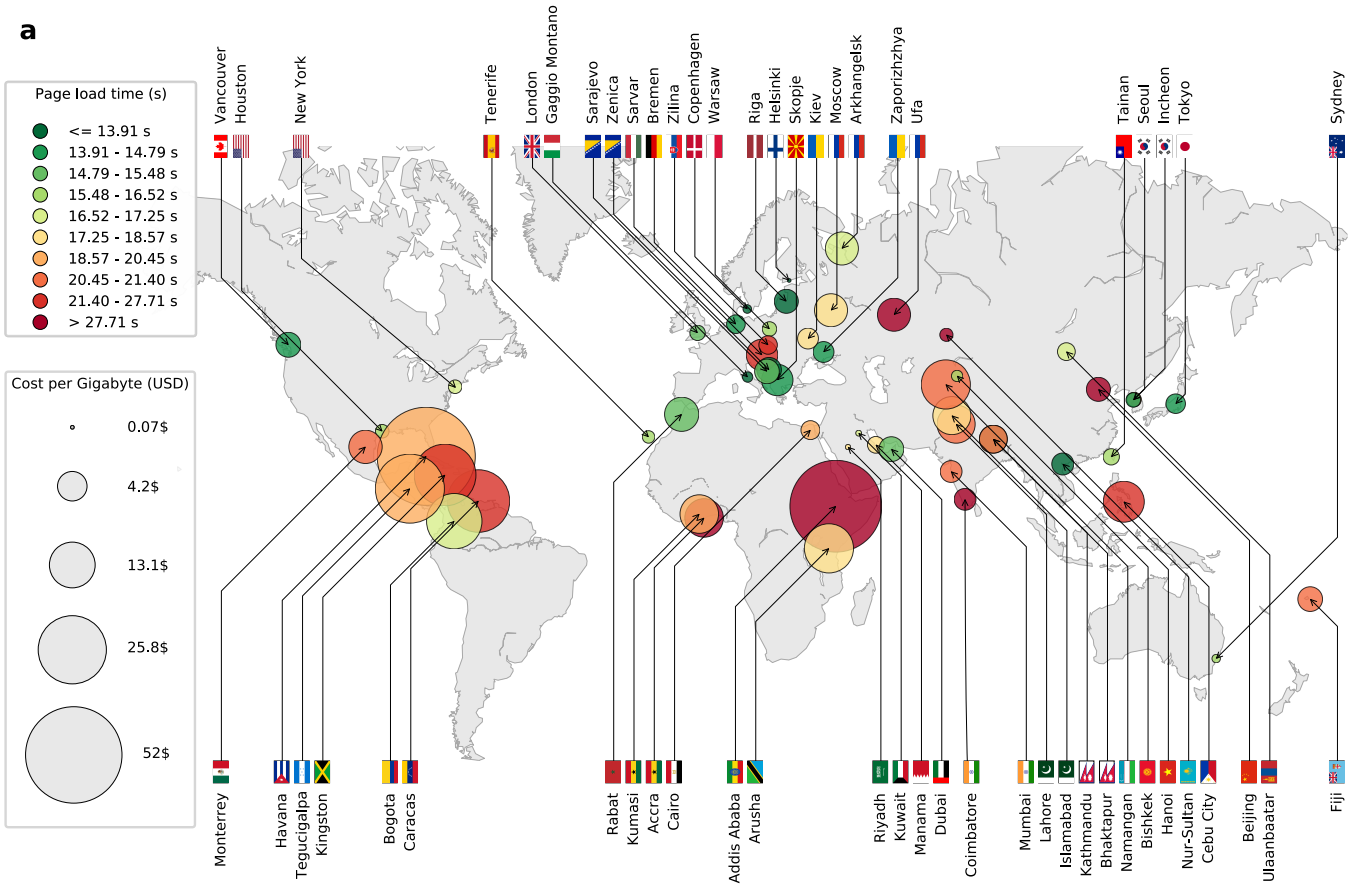


Fig. S2. Average page load time and data cost across different locations. Similar to Figure 1a in the main article, but instead of using the purchasing power parity (PPP), the cost per Gigabyte is computed based on the gross domestic product (GDP) in purchasing power parity (PPP) for each country.

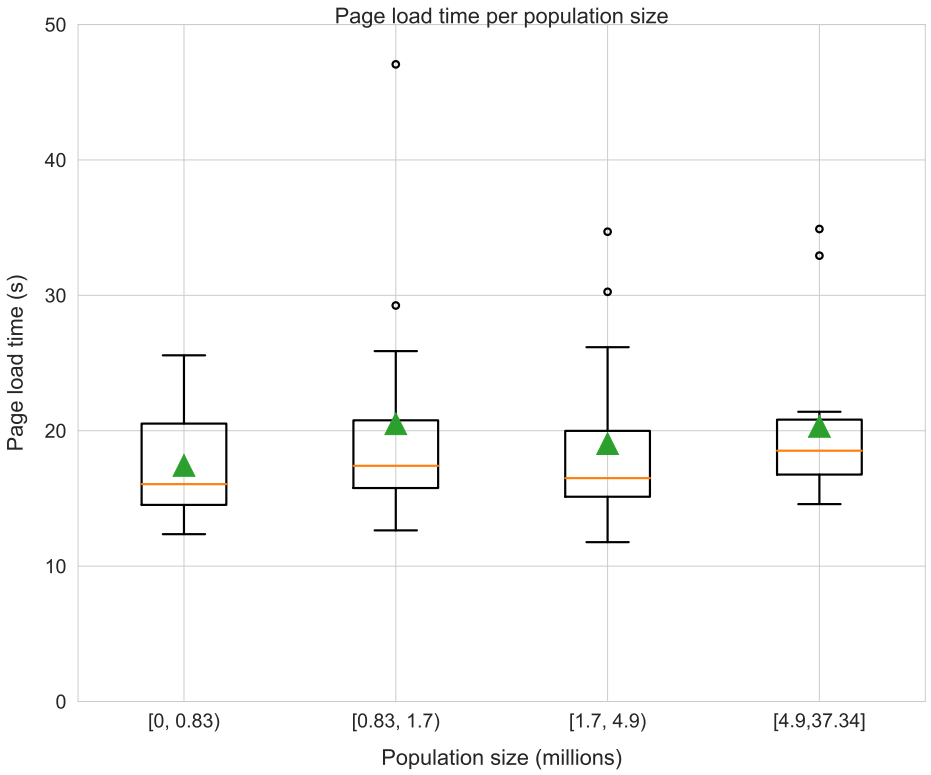


Fig. S3. Page load time for different population sizes. The 56 cities analyzed in Figure 1 of the main article were divided into quartiles based on population size. The box plot summarizes the page load time in each quartile, with triangles representing the mean.

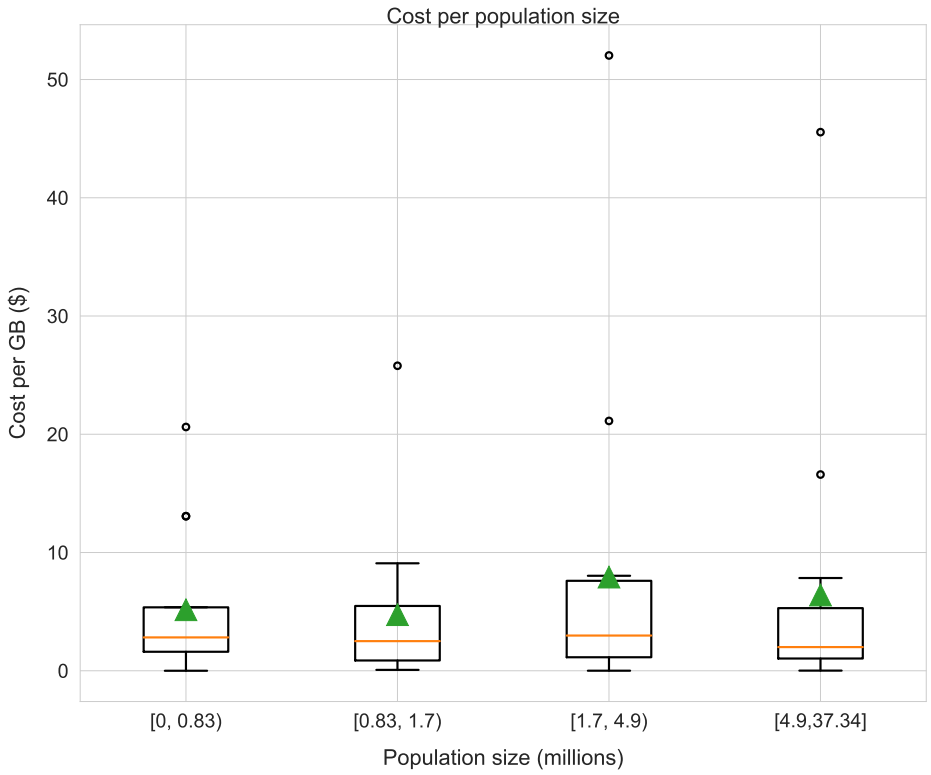


Fig. S4. Cost per Gigabyte for different population sizes. The 56 cities analyzed in Figure 1 of the main article were divided into quartiles based on population size. The box plot summarizes the cost per Gigabyte in each quartile, with triangles representing the mean.

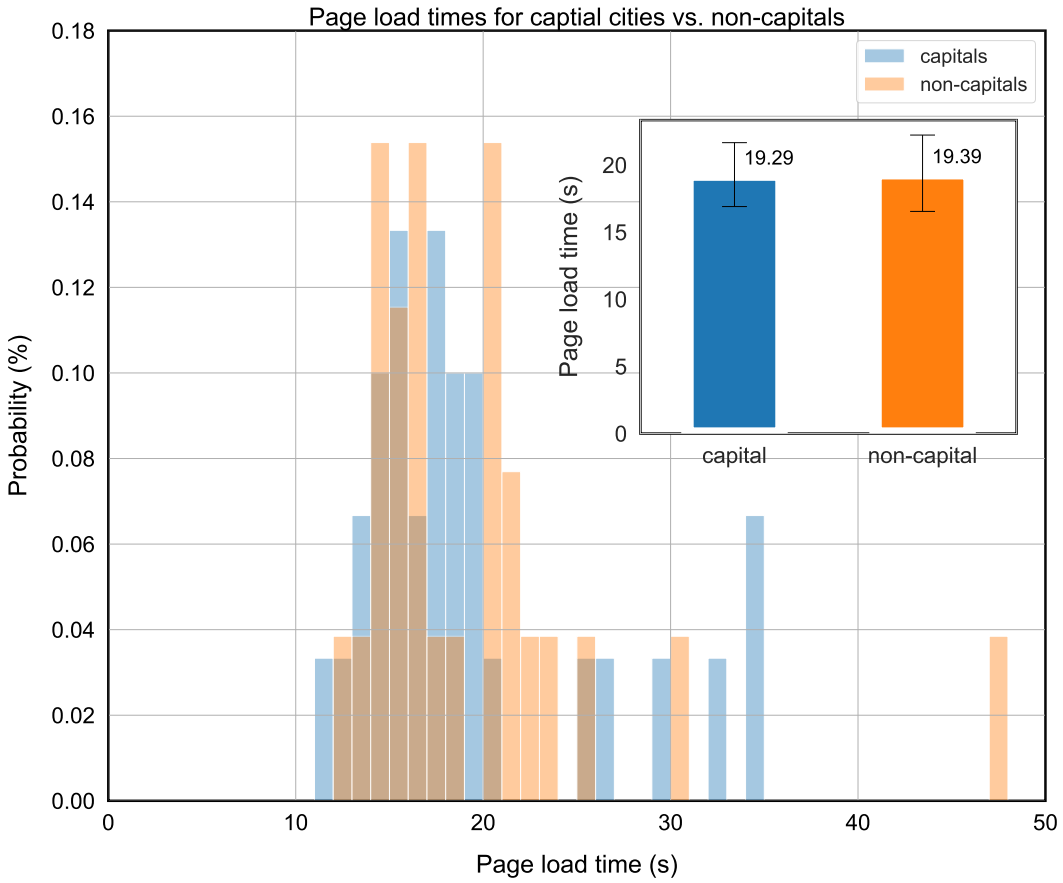


Fig. S5. Page load time in capital vs. non-captial cities. The 56 cities analyzed in Figure 1 of the main article were divided into capital and non-capital cities. The distribution of page load time is depicted in blue for capital cities and orange for non-captial cities, with the inset depicting the means along with the 95% confidence intervals. The page load time is almost identical across the two groups.

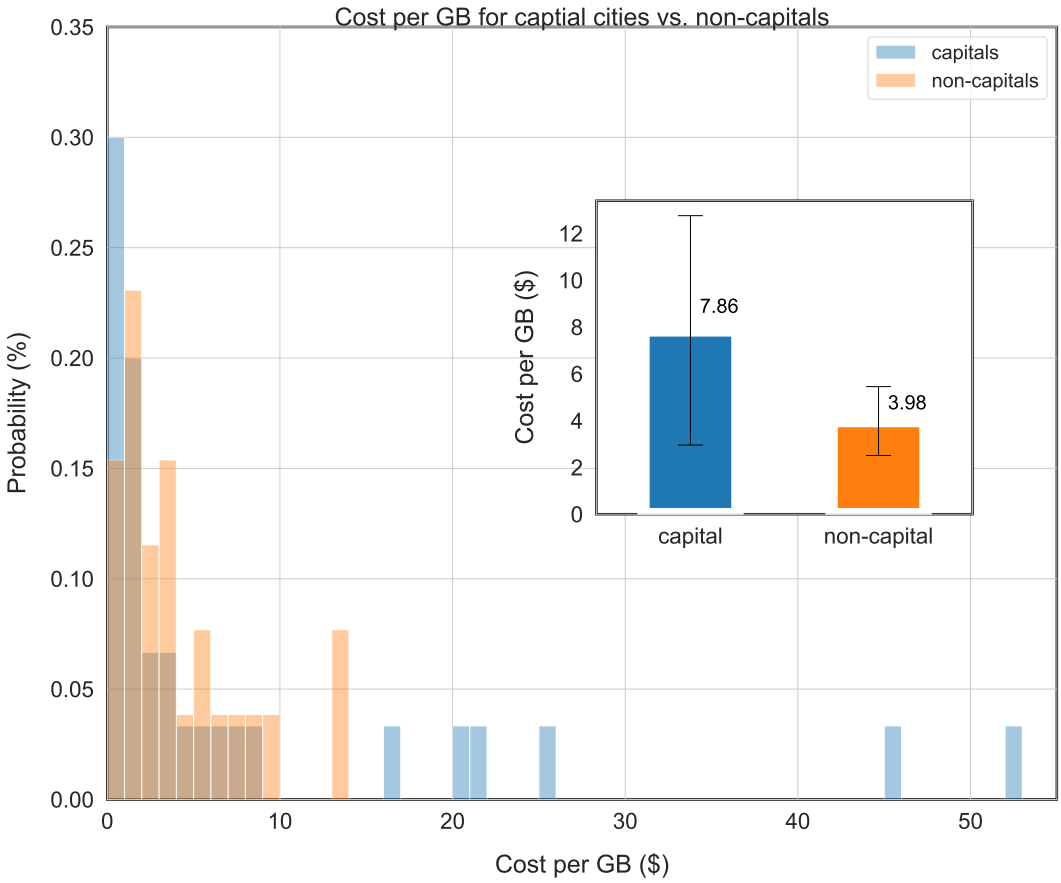


Fig. S6. Cost per Gigabyte in capital vs. non-captial cities. The 56 cities analyzed in Figure 1 of the main article were divided into capital and non-capital cities. The cost per Gigabyte is depicted in blue for capital cities and orange for non-captial cities, with the inset depicting the means along with 95% confidence intervals. The cost in capital cities is nearly twice the cost in non-captial cities.

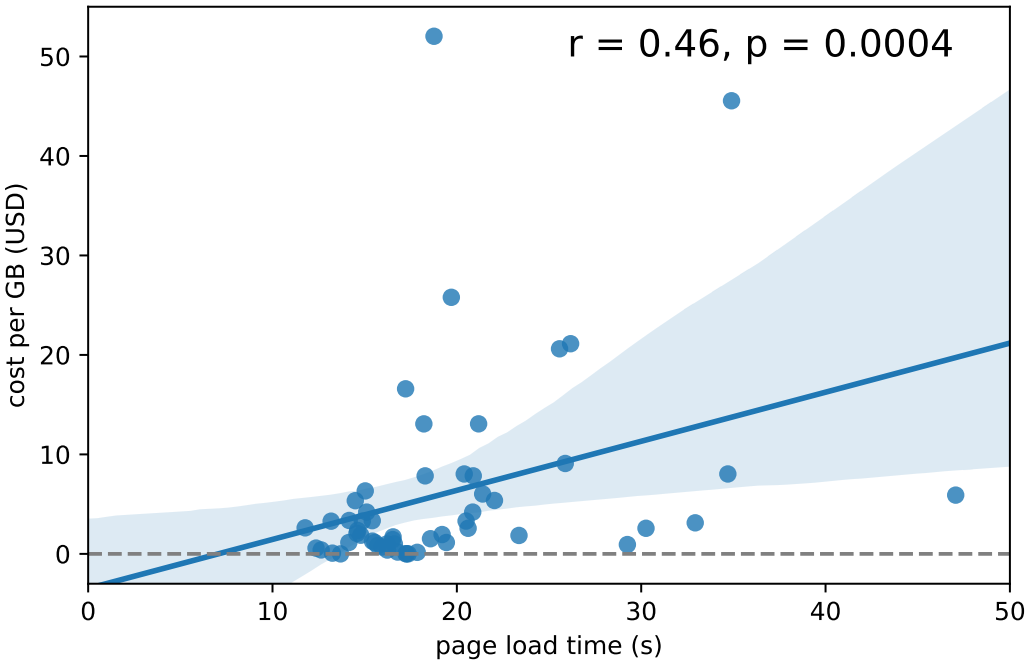


Fig. S7. Correlation between page load time and cost per Gigabyte. Each data point represents one of the 56 cities analyzed in Figure 1 of the main article.

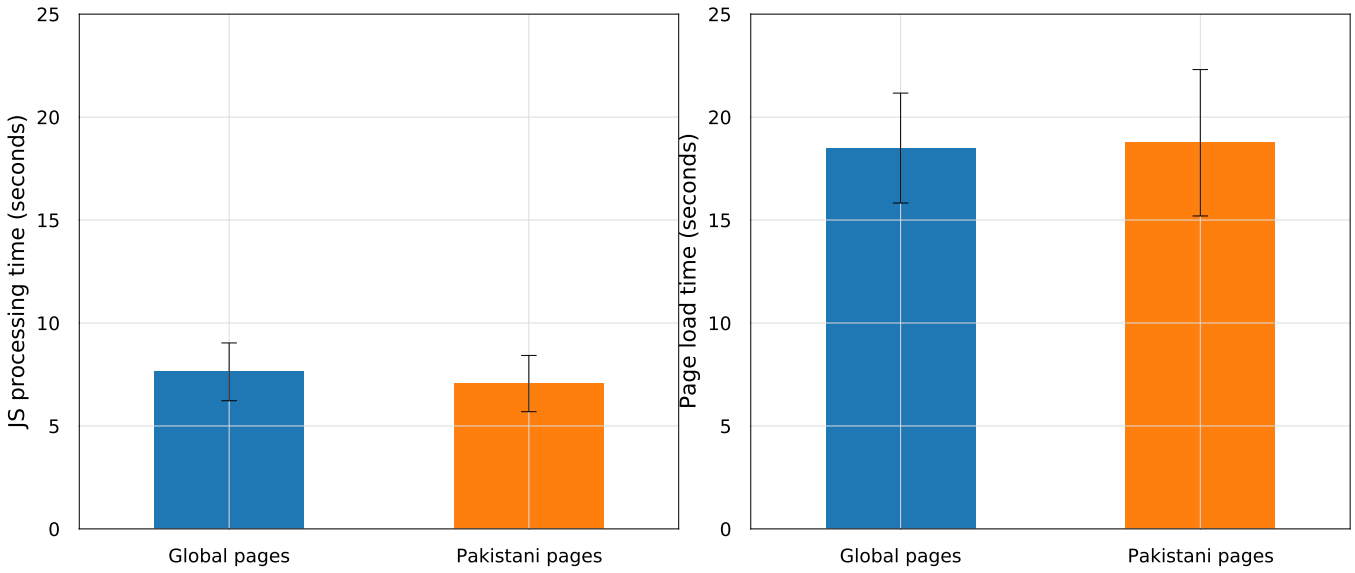


Fig. S8. Comparing popular Pakistani webpages to webpages that are popular globally. The figure compares the 100 webpages most frequently visited in 2021 to their Pakistani counterparts, i.e., the 100 Pakistani webpages most frequently visited in 2021. The evaluation is done using a low-end mobile phone, namely QMobile i6i. The left and right subfigures compare the two types of webpages in terms of JavaScript processing time and page load time, respectively. In both subfigures, the difference between the two types is extremely small.

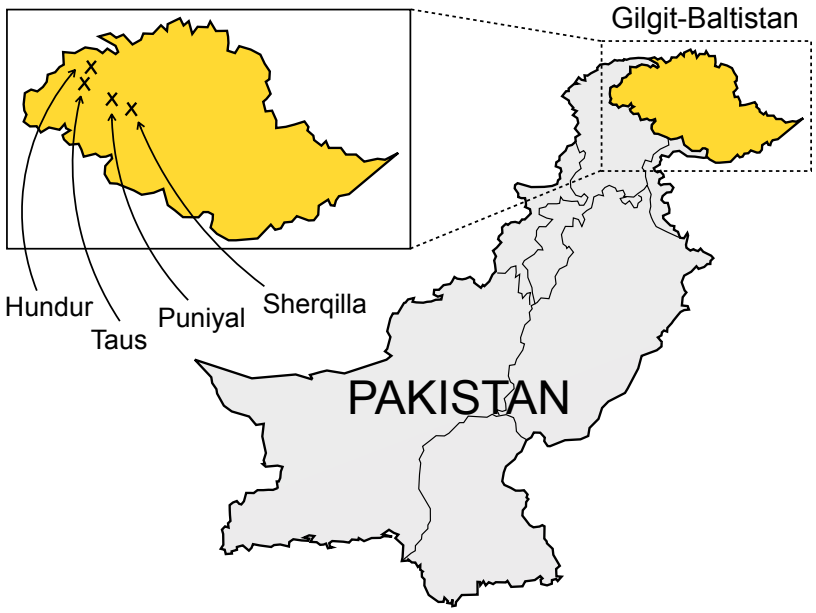


Fig. S9. Field experiment locations. The map of Pakistan, highlighting the Gilgit-Baltistan province, and the locations in which the field experiment took place.

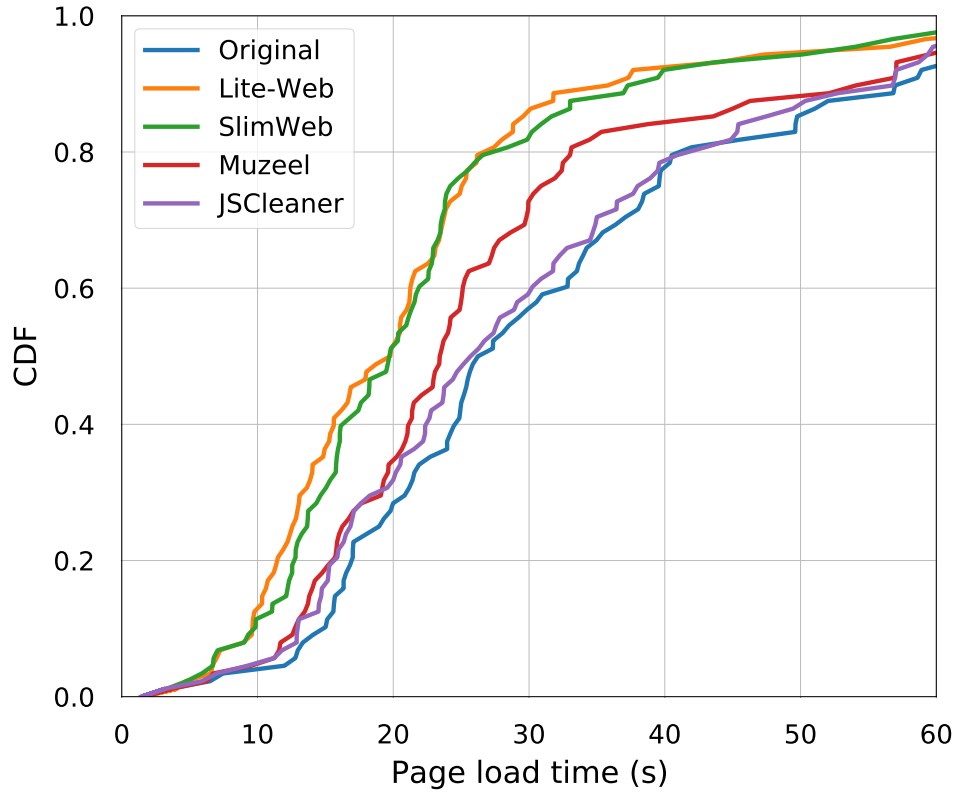


Fig. S10. The impact of the constituent parts of Lite-Web on page load time. For each of the 100 Pakistani webpages most frequently visited in 2021, we compared the page load time of the original version to the page load time in the LiteWeb version, as well as the versions produced by each of the three parts of Lite-Web, namely SlimWeb, Muzeel, and JSCleaner. The figure depicts the cumulative density functions (CDFs).

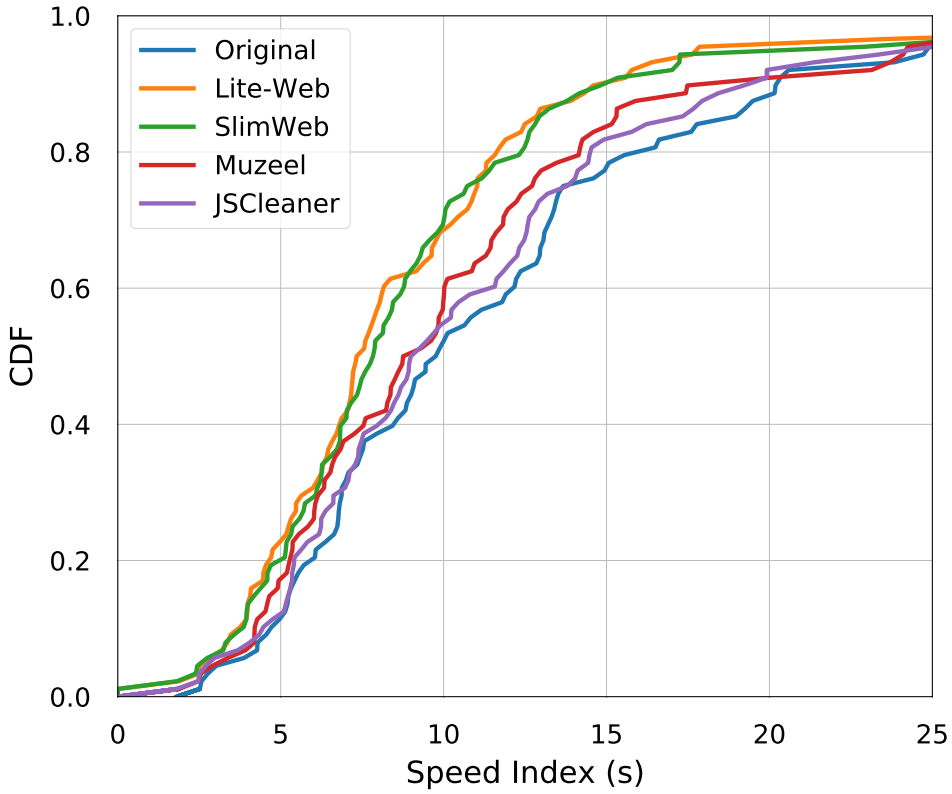


Fig. S11. The impact of the constituent parts of Lite-Web on Speed Index. For each of the 100 Pakistani webpages most frequently visited in 2021, we compared the Speed Index of the original version to the Speed Index in the LiteWeb version, as well as the versions produced by each of the three parts of Lite-Web, namely SlimWeb, Muzeel, and JSCleaner. The figure depicts the cumulative density functions (CDFs).

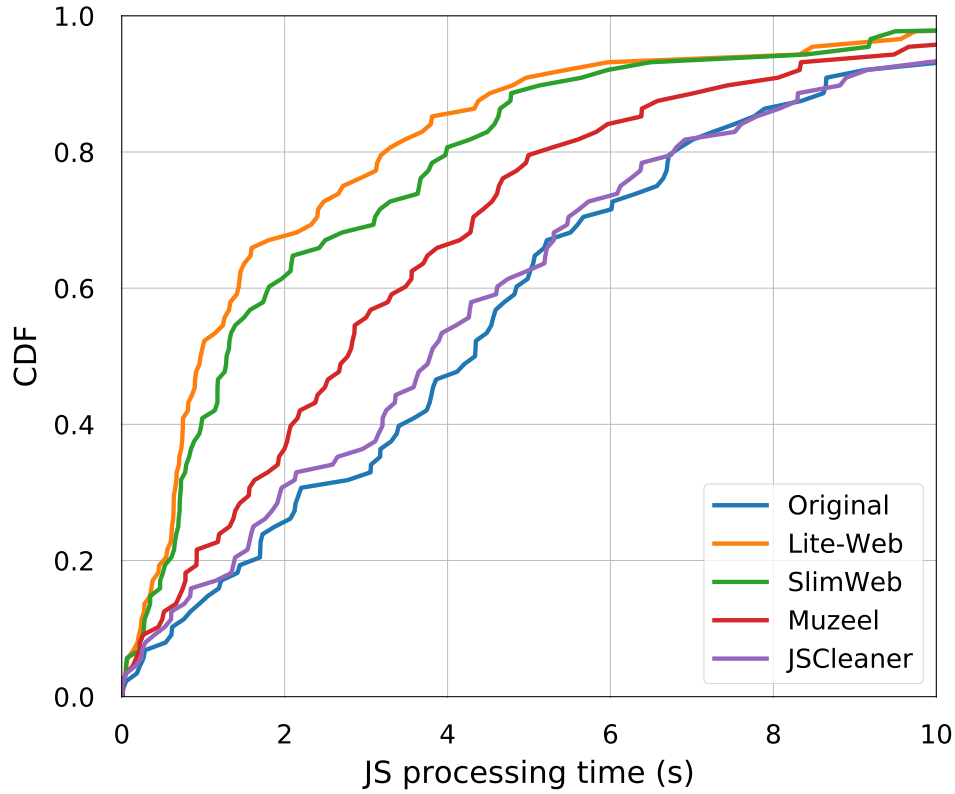


Fig. S12. The impact of the constituent parts of Lite-Web on JavaScript processing time. For each of the 100 Pakistani webpages most frequently visited in 2021, we compared the JavaScript processing time of the original version to the JavaScript processing time in the LiteWeb version, as well as the versions produced by each of the three parts of Lite-Web, namely SlimWeb, Muzeel, and JSCleaner. The figure depicts the cumulative density functions (CDFs).

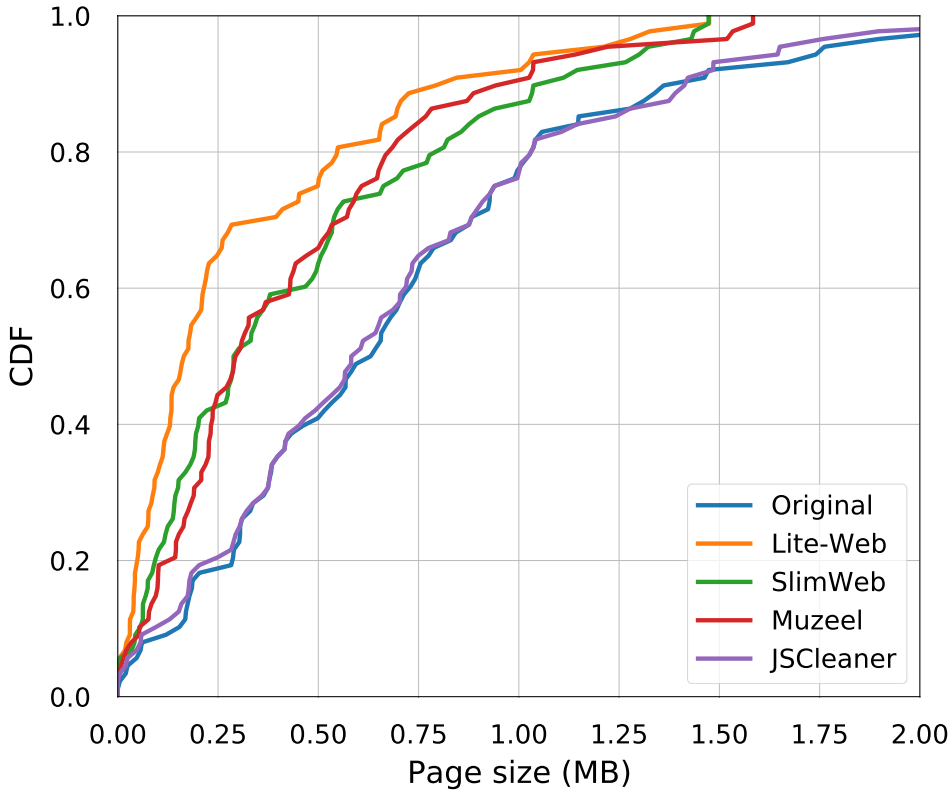


Fig. S13. The impact of the constituent parts of Lite-Web on page size. For each of the 100 Pakistani webpages most frequently visited in 2021, we compared the page size of the original version to the page size in the LiteWeb version, as well as the versions produced by each of the three parts of Lite-Web, namely SlimWeb, Muzeel, and JSCleaner. The figure depicts the cumulative density functions (CDFs).

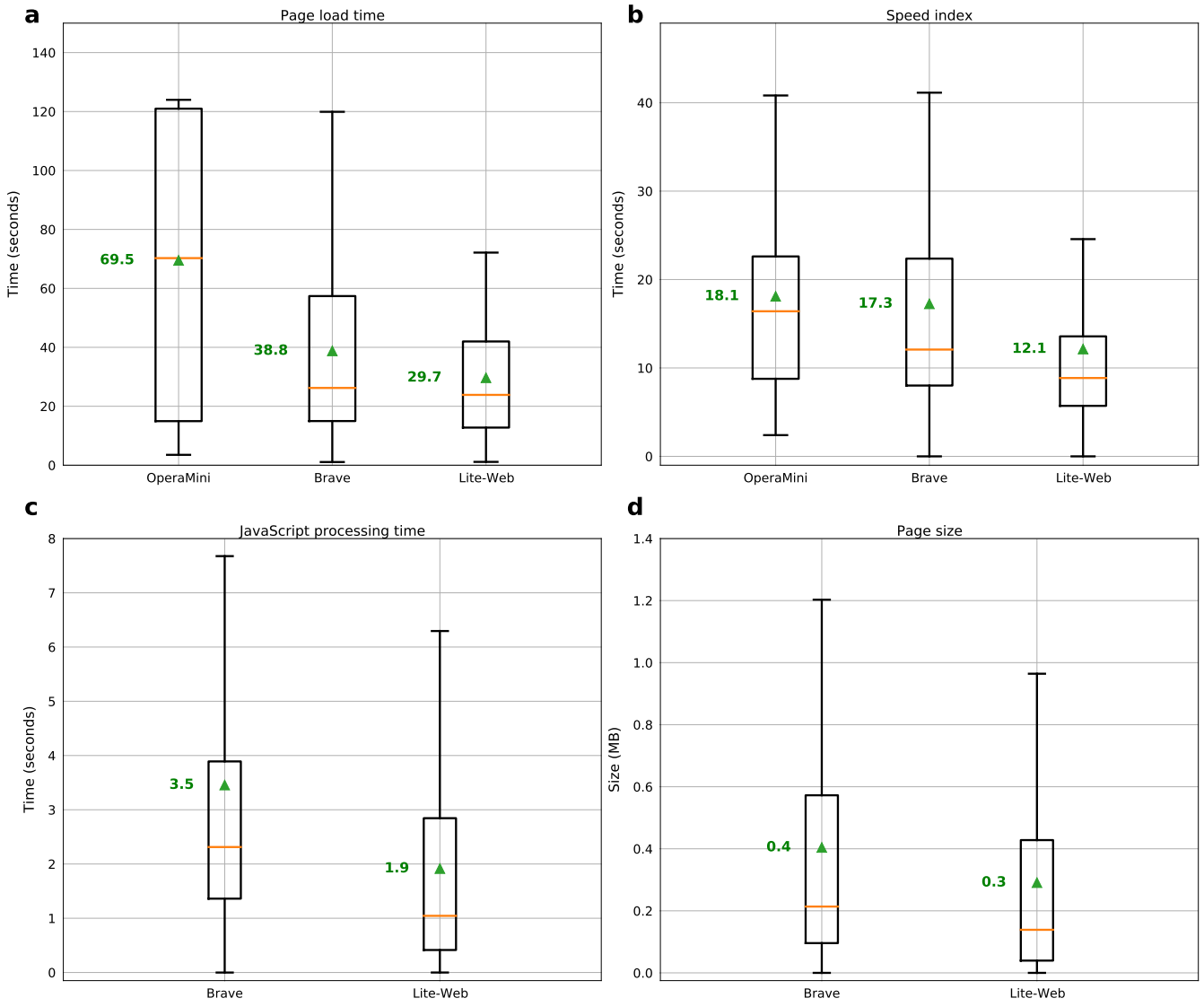


Fig. S14. Comparing Lite-Web to existing alternatives. Using the 100 most frequently visited Pakistani webpages in 2021 to compare Lite-Web to existing alternatives, namely Brave and OperaMini. The evaluation is done on the same low-end phone (QMobile i6i 2020) under the same cellular network conditions (SCOM 4G) in the city of Lahore, Pakistan. **a**, Evaluating page load time. **b**, Evaluating Speed Index. **c**, Evaluating JavaScript processing time. **d**, Evaluating page size.

Fig. S15. The user study setup. All photos were taken during the user study that took place in the Gilgit-Baltistan province of Pakistan. The upper two rows show photos taken at one of the two schools where the experiment took place, showing how social distancing was observed while wearing masks. The bottom row shows photos of local merchants using their mobile phones.

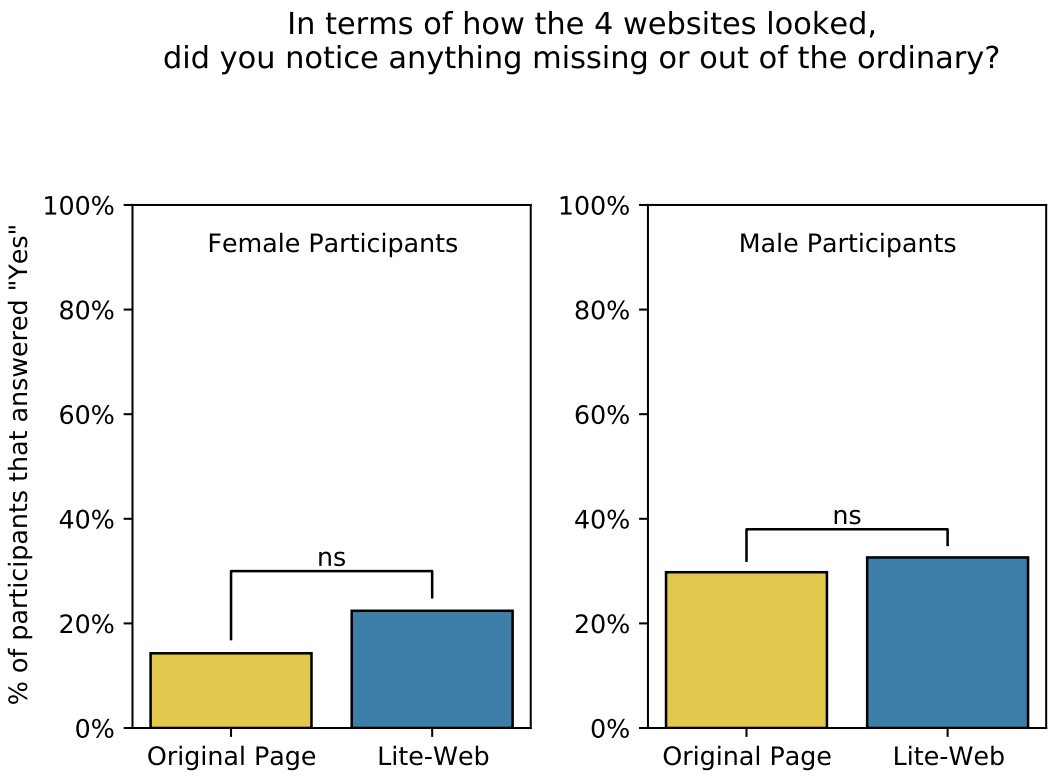


Fig. S16. Evaluating Lite-Web's impact on appearance while accounting for participants' gender. Each participant interacted with 4 of the 100 Pakistani websites most frequently visited in 2021; the control and treatment groups interacted with the original and Lite-Web versions of these websites, respectively. The figure summarizes the responses to the question: "*In terms of how the 4 websites looked, did you notice anything missing or out of the ordinary?*" after dividing participants into Female and Male. (ns $=$ not significant; $p = 0.34$ for Females and $p = 0.83$ for Males).

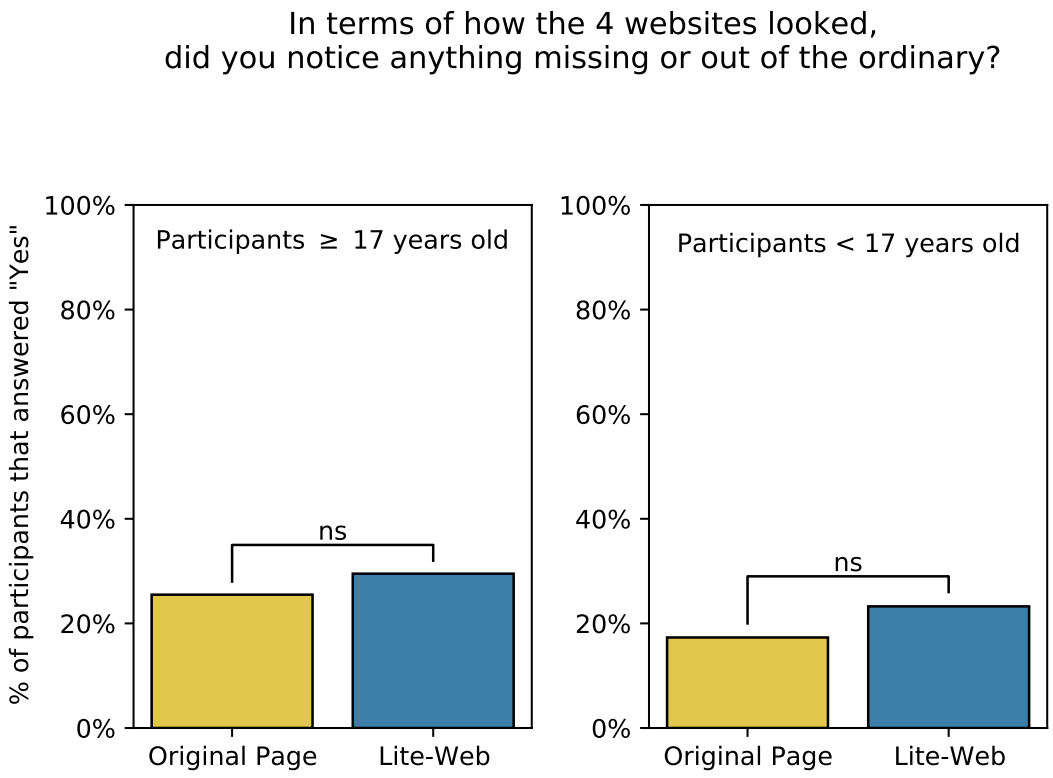


Fig. S17. Evaluating Lite-Web's impact on appearance while accounting for participant's age. Each participant interacted with 4 of the 100 Pakistani websites most frequently visited in 2021; the control and treatment groups interacted with the original and Lite-Web versions of these websites, respectively. The figure summarizes the responses to the question: "*In terms of how the 4 websites looked, did you notice anything missing or out of the ordinary?*" after dividing participants into those whose age is ≥ 17 and those whose age is < 17, where 17 is the median age. (ns = not significant; $p = 0.68$ for ≥ 17 and $p = 0.61$ for < 17).

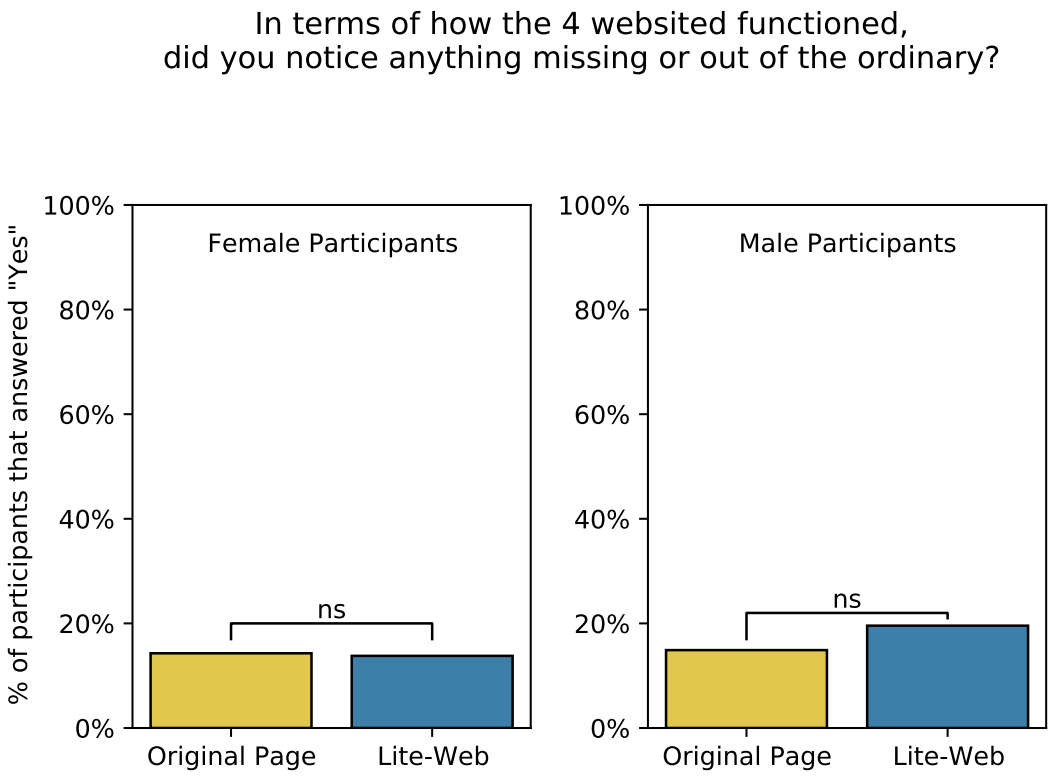


Fig. S18. Evaluating Lite-Web's impact on functionality while accounting for participants' gender. Each participant interacted with 4 of the 100 Pakistani websites most frequently visited in 2021; the control and treatment groups interacted with the original and Lite-Web versions of these websites, respectively. The figure summarizes the responses to the question: "*In terms of how the 4 websites functioned, did you notice anything missing or out of the ordinary?*" after dividing participants into Female and Male. (ns = not significant; $p = 1.0$ for Females and $p = 0.59$ for Males)

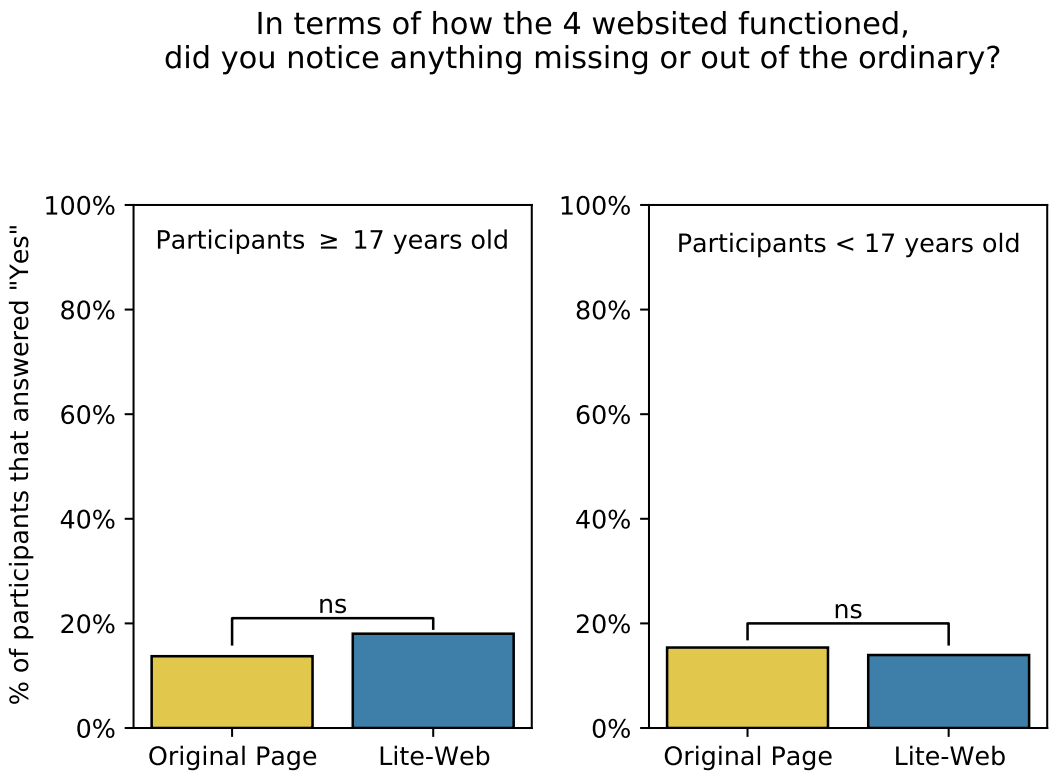


Fig. S19. Evaluating Lite-Web's impact on functionality while accounting for participants' age. Each participant interacted with 4 of the 100 Pakistani websites most frequently visited in 2021; the control and treatment groups interacted with the original and Lite-Web versions of these websites, respectively. The figure summarizes the responses to the question: "*In terms of how the 4 websites functioned, did you notice anything missing or out of the ordinary?*" after dividing participants into those whose age is ≥ 17 and those whose age is < 17, where 17 is the median age. (ns = not significant; $p = 0.61$ for ≥ 17 and $p = 1.0$ for < 17).

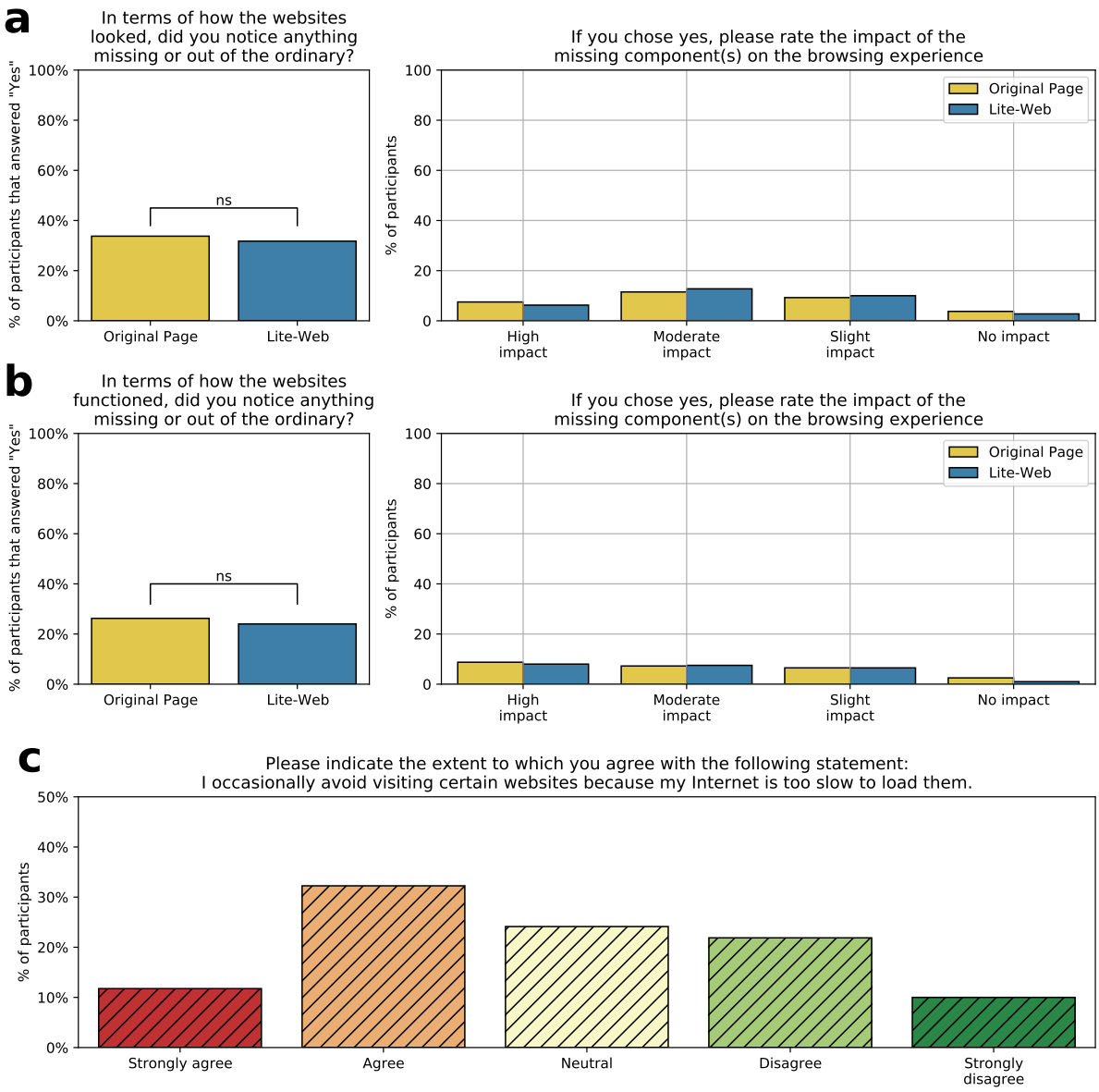
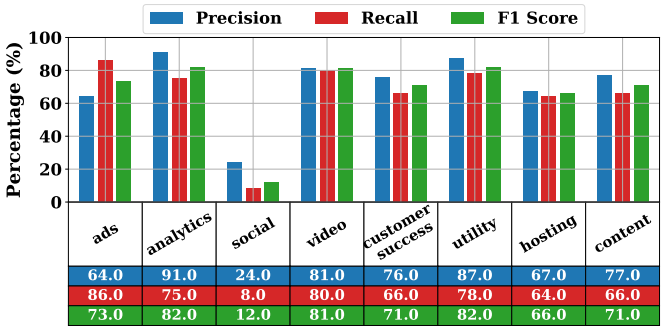
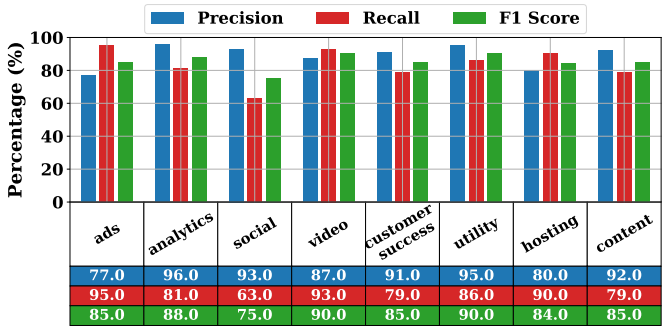


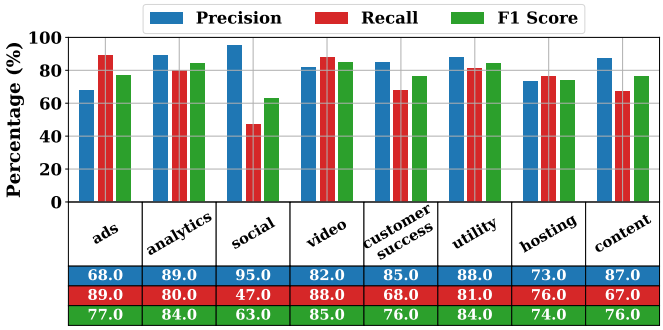
Fig. S20. University students' evaluation of Lite-Web's impact on the appearance and functionality of websites. Ten Pakistani websites were selected out of the 100 most frequently visited in 2021, and each of the 800 participants interacted with all ten webpages. The control and treatment groups interacted with the original and Lite-Web versions of these websites, respectively. **a,** Left panel: Percentage of participants who answered "Yes" to the question: "*In terms of how the websites looked, did you notice anything missing or out of the ordinary?*" (ns = not significant; $p = 0.598$); those who answered "Yes" were subsequently asked: "*If you chose yes, please rate the impact of the missing component(s) on the browsing experience*"; the distribution of their responses is depicted in the right panel. **b,** Similar to (**a**) but for questions asking about how the websites functioned, rather than how the websites looked (ns = not significant; $p = 0.514$). **c,** Responses of all participants (control and treatment) to the question: "*Please indicate the extent to which you agree with the following statement: I occasionally avoid visiting certain website because my Internet is too slow to load them.*"



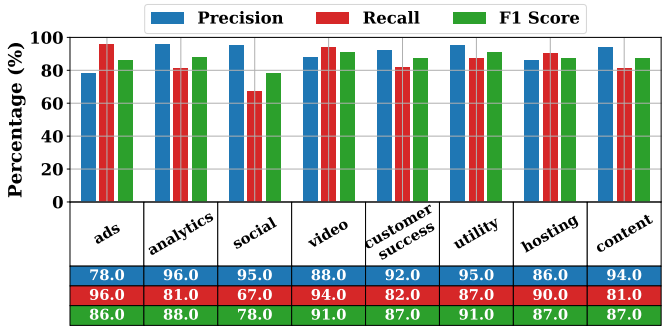
(a) K-Nearest Neighbors (KNN)



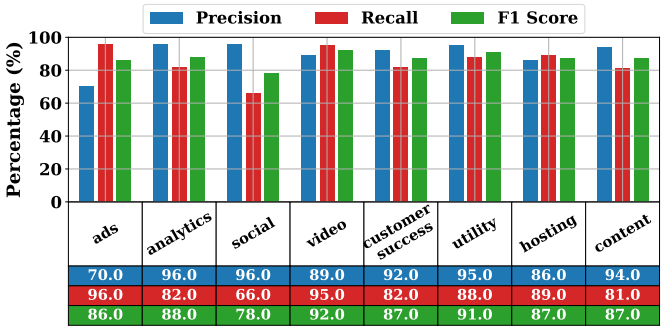
(b) Support Vector Machine



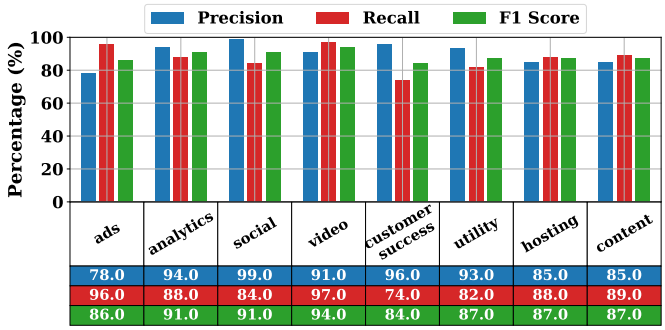
(c) Linear Support Vector Classifier (LSVC)



(d) Random Forest Classifier (RFC)



(e) XGBoost



(f) Simple Neural Network

Fig. S21. For each of the 8 categories, the figure depicts the performance of different classifiers in terms of precision, recall, and F1-score.

Table S1. The cost per Gigabyte (measured based on the purchasing power parity) and the page load time (measured in seconds) in each of the locations considered in our study.

City/Country	Page Load Time	Cost
Tokyo, Japan	14.58	1.740530688
Riga, Latvia	13.69	2.166064982
Hanoi, Vietnam	11.77	1.365457547
Sydney, Australia	16.23	0.392413342
Kuwait, Kuwait	16.79	0.326203209
Warsaw, Poland	15.55	1.71009772
Skopje, Macedonia	14.49	4.737221997
Mumbai, India	20.61	1.429965586
Ufa, Russia	47.06	11.11507653
Houston, United States	16.21	0.910384068
Fiji	20.5	2.15010142
Caracas, Venezuela	26.17	32.06470695
Beijing, China	32.93	1.782011092
Manama, Bahrain	18.57	2.794117647
Addis Ababa, Ethiopia	34.9	9.616290019
Seoul, South Korea	19.43	1.214332492
London, United Kingdom	15.42	1.282051282
Bremen, Germany	14.79	2.388818297
Ulaanbaatar, Mongolia	16.54	1.49798178
Sarvar, Hungary	22.04	8.029204014
Vancouver, Canada	14.16	3.690753691
Lahore, Pakistan	20.89	3.963790563
Gaggio Montano, Italy	12.36	0.599739244
Zenica, Bosnia and Herzegovina	13.18	2.327365729
Zilina, Slovakia	23.37	1.95164076
Cairo, Egypt	19.2	2.303403756
Islamabad, Pakistan	18.28	3.963790563
Bishkek, Kyrgyzstan	15.94	0.328808135
Dubai, UAE	15.41	6.349206349
Helsinki, Finland	13.25	0.078794902
Havana, Cuba	18.76	43.33333333
Accra, Ghana	34.7	3.158195317
Arkhangelsk, Russia	17.24	11.11507653
Kingston, Jamaica	25.57	9.056284805
Zaporizhzhya, Ukraine	14.62	2.629174993
Monterrey, Mexico	21.4	5.576513538
Copenhagen, Denmark	12.64	0.400456216
Tenerife, Spain	15.71	0.910384068
Moscow, Russia	17.27	11.11507653
Kumasi, Ghana	20.4	3.158195317
Arusha, Tanzania	18.21	4.4428402
Cebu City, Philippines	25.88	4.4428402
Namangan, Uzbekistan	21.18	11.60614212
Bogota, Colombia	17.22	14.77002473
Tegucigalpa, Honduras	19.7	7.329089399
Sarajevo, Bosnia and Herzegovina	14.87	2.327365729
Tainan, Taiwan	16.52	2.878057597
Bhaktapur, Nepal	15.11	1.203421492
Kiev, Ukraine	17.36	2.629174993
Incheon, South Korea	14.14	1.214332492
Rabat, Morocco	15.03	2.234910277
Kathmandu, Nepal	20.86	1.203421492
Nur-Sultan, Kazakhstan	29.25	1.84994771
Coimbatore, India	30.26	1.429965586
New York, United States	16.61	0.910384068
Riyadh, Saudi Arabia	17.85	0.348027842

Table S2. Xiaomi Redmi Go specifications.

Category	Specifications
Launch year	2019
CPU	Quad-core 1.4 GHz Cortex-A53
Operating System	Android 8.1 Oreo (Go edition)
Memory	8GB 1GB RAM
Price	around 70 USD

Table S3. Specifications of the low-end phones used in the experiment that evaluates JavaScript over the years 2015-2020. Note that Xiaomi Redmi Go was released in January 2019, so we used it for 2018 since we were unable to obtain another low-end phone released in 2018.

Year	Low-end Phone	CPU	RAM
2015	HTC Desire 520	Quad-core 1.1 GHz Cortex-A7	1 GB
2016	Huawei Y3II	Quad-core 1.0 GHz Cortex−A53	1 GB
2018	Xiaomi Redmi go	Quad-core 1.4 GHz Cortex-A53	1 GB
2019	Alcatel 1C	Quad-core 1.3 GHz Cortex−A53	1 GB
2020	Nokia C1	Quad-core 1.3 GHz	1 GB

Table S4. Specifications of the high-end phones used in the experiment that evaluates JavaScript over the years 2015-2020.

Year	High-end Phone	CPU	RAM (GB)
2015	Samsung Galaxy S6 edge	Octa-core 4x2.1 GHz Cortex-A57, 4x1.5 GHz Cortex-A53	3
2016	Huawei P9	Octa-core 4x2.5 GHz Cortex-A72, 4x1.8 GHz Cortex-A53	4
2017	Samsung Galaxy S8+	Octa-core 4x2.3 GHz Mongoose M2, 4x1.7 GHz Cortex-A53	4
2018	Huawei Mate 20 Pro	Octa-core 2x2.6 GHz Cortex-A76, 2x1.92 GHz Cortex-A76, 4x1.8 GHz Cortex-A55	6
2019	Samsung Galaxy S10+	Octa-core 2x2.73 GHz Mongoose M4, 2x2.31 GHz Cortex-A75, 4x1.95 GHz Cortex-A55	12
2020	Samsung Galaxy S20	Octa-core 2x2.73 GHz Mongoose M5, 2x2.50 GHz Cortex-A76, 4x2.0 GHz Cortex-A55	8

Table S5. Supervised learning evaluation summary.

Models	Recall	Precision	F1-Score
KNN	0.75	0.76	0.75
SVM	0.86	0.88	0.86
LSVC	0.79	0.81	0.79
RFC	0.88	0.89	0.88
XGBoost	0.88	0.89	0.88
4-layer Neural Network	0.89	0.9	0.89

References

1. M Chaqfeh, et al., To block or not to block: Accelerating mobile web pages on-the-fly through javascript classification. *CoRR* **abs/2106.13764** (2021).
2. M Chaqfeh, Y Zaki, J Hu, L Subramanian, Jscleaner: De-cluttering mobile webpages through javascript cleanup in *Proceedings of The Web Conference 2020*. pp. 763–773 (2020).
3. Mozilla, individual contributors, Document object model (dom) (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model) (2005-2019).
4. Mozilla, individual contributors, The html dom api (https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API) (2005-2019).
5. E International, Ecmascript® 2018 language specification (http://www.ecma-international.org/ecma-262/9.0/index.html) (2019) Accessed: 2019-05-05.
6. H Archive, Third parties | 2019 | the web almanac by http archive (https://almanac.httparchive.org/en/2019/third-parties) (2019) Accessed: 2020-01-2.
7. P Hulce, third-party-web/entities.json5 at 8afa2d8cadddec8f0db39e7d715c07e85fb0f8ec · patrickhulce/third-party-web (https://github.com/patrickhulce/third-party-web/blob/8afa2d8cadddec8f0db39e7d715c07e85fb0f8ec/data/entities.json5) (2019) Accessed: 2020-01-2.
8. T Chen, C Guestrin, Xgboost: A scalable tree boosting system in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. pp. 785–794 (2016).
9. AF Agarap, Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375* (2018).
10. T Kupoluyi, et al., Muzeel: A dynamic javascript analyzer for dead code elimination in today's web. *arXiv preprint arXiv:2106.08948* (2021).
11. NG Obbink, I Malavolta, GL Scoccia, P Lago, An extensible approach for taming the challenges of javascript dead code elimination in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. (IEEE), pp. 291–401 (2018).