Breaking Down Complexity: MAML's Impact on Web Page Optimization in Developing Regions

Ayush Pandey
Computer Science, NYUAD
ayush.pandey@nyu.edu

Advised by: Professor Yasir Zaki, Professor Lakshminarayanan Subramanian

ABSTRACT

The web experience for users in developing regions continues to be subpar, despite the increased penetration of mobile internet usage. This can be attributed to a combination of increased web page complexity and inadequate optimization efforts by content providers. Modern web pages are becoming increasingly complex, often comprising more than 100 web objects, requiring over 30 TCP connections, 20 recursive DNS lookups, and excessive usage of recursive JavaScript. These factors significantly impact the initial load time, as browsers must retrieve, parse, and execute these files before rendering the page. In comparison to users in developed regions who benefit from enhanced internet speeds, individuals in developing regions are generally plagued with access to low-end smartphones and slower data connections, hence exacerbating the issue of poor page load times. Recognizing these challenges, this paper presents MAML, a novel web specification language designed to simplify web pages, thereby reducing costs and lowering barriers for content creation in developing regions. We have developed an intuitive web-based user interface named "MAML Editor," which enables web developers to transform complex web pages into significantly lighter versions with functionality almost equivalent to the original. Furthermore, this editor is equipped with an "Auto Translation Engine" that is designed to expedite the process of converting pre-existing web pages into MAML-compatible pages. We evaluated the efficacy of the MAML editor through a competition involving 20 web developers with varying levels of web-development experience, who were tasked with the recreation of 50 existing web pages. The results demonstrate that users equipped with our editor were successful in reproducing many of these pages without incurring a substantial quality reduction. Additionally, the MAML editor's simplification led to a significant reduction in bandwidth requirements, which can decrease the average data consumption for users in developing regions by ~50%. Additionally, the web browsing experience was substantially accelerated, with a median page load time speedup of up to five seconds.

KEYWORDS

MAML, Web Specification Format, Page Load Time, Auto Translation Engine, MAML Editor, DOM Complexity

Reference Format:
Ayush Pandey. 2024. Breaking Down Complexity: MAML's Impact on Web Page Optimization in Developing Regions. In *NYUAD Capstone Project 2 Reports, Spring 2024, Abu Dhabi, UAE.* 17 pages.

1 INTRODUCTION

In recent years, web development practices have evolved significantly, with a strong emphasis on creating visually appealing and interactive web pages. This evolution has led to the widespread use of complex frameworks and libraries, which, while powerful, often result in resource-heavy pages. For example, popular frameworks like React and Angular have become the go-to choices for many developers, but they can contribute to increased page load times in developing regions and increased resource consumption due to their size and the complexity of the Document Object Model (DOM).

To analyze current trends in DOM complexity, we evaluated the top 1000 pages retrieved from Tranco's [30] list using Google Lighthouse, an open-source, automated tool designed to improve web page quality across multiple dimensions, one of which is performance. Lighthouse flags pages with more than 800 DOM elements, and recommends DOM tree depth to be less than 32 for better performance. The results of our evaluation show that 62.1% of these pages feature more than 800 DOM elements, and pages reach a maximum DOM tree depth of 60. Additionally, our analysis

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

جامعة نيويورك أبوظبي
NYU ABU DHABI

Capstone Project 2, Spring 2024, Abu Dhabi, UAE

revealed that these pages have an average of 524.7 KB of unused JavaScript and 81.9 KB of unused CSS.

Every modification to the DOM, no matter how minor, can trigger a cascade of recalculations and re-renderings, collectively known as reflows and repaints. These operations are resource-intensive and can significantly degrade performance, resulting in slower page loads and a suboptimal user experience. Furthermore, the extensive use of JavaScript to manipulate the DOM in frameworks like React and Angular not only increases initial load times but also adds to ongoing computational costs. Although these frameworks enable developers to create highly interactive and responsive sites, they often compromise speed, especially on low-end devices and networks.

In response to these challenges, developers and organizations are increasingly advocating for more streamlined web development approaches. Approaches such as Polaris [33] and Shandian [36] prioritize optimizing the DOM Tree to minimize its depth and complexity, thereby enhancing page performance. However, they still heavily rely on JavaScript execution and resource management, which can continue to impact performance, particularly on devices with limited processing power. This is where the concept of a flat layout comes into play. By adopting a self-contained dictionary-like representation, MAML significantly reduces the computational overhead and resource size associated with traditional web development practices, without needing to compromise interactivity.

2 BACKGROUND AND RELATED WORK

In this section, we describe related work that demonstrates the key factors that trigger poor performance for bandwidth-constrained users and also outlines different related efforts that have addressed this challenge.

2.1 Poor Web Performance in Developing Regions

Poor web performance for users in developing regions can be triggered due to two factors: network-induced and complexity-induced delays.

Network-Induced Delays: A measurement study by Zaki et. al. [39] showed that developing regions like Ghana suffer from high page load times (multi-seconds), due to HTTP redirects, DNS lookups, and TLS/SSL connection setups. They found that the actual time spent downloading content represents only a small fraction of the end-to-end page download time. Koradia et. al. [28] have shown that cellular data connectivity in India suffers from significantly high latencies of up to 1200 ms. It has been shown in another work [17] that in bandwidth constrained environments, TCP flows experience severe unfairness, high packet loss rates, and flow silences due to repetitive timeouts, thereby resulting in poor web performance. Other works have pointed out traffic engineering and lack of infrastructure as reasons for high delays [14, 22, 24, 25]. Feamster et. al. [20] have also demonstrated that despite having web caches, there is high latency in South Africa due to the fact that the inter-connectivity within the country is still a major issue. Studies have also looked at inefficient DNS configurations, a lack of local content caching servers, and a lack of cross-border cable systems [23, 27, 39].

Complexity-Induced Delays: In addition to the network-related delay, the complexity of web pages is another issue causing high delays while rendering web pages. Modern web pages have evolved significantly over the past couple of years into a very complex ecosystem with a large number of objects that span several servers across the globe. Butkiewicz et. al. [8] have shown that more than 60% of web pages request data from at least 5 different non-origin sources, and that these contribute to more than 35% of the overall page size. Furthermore, in order to load a page, a modern browser must fetch and render several objects, which include HTML files, JavaScript files, CSS files, image files, and other objects. These objects form a complex object dependency graph. That is, as these objects are downloaded and evaluated, they trigger the download of other objects. For example, a JavaScript object can include other JavaScript files that have to be recursively fetched when the page is being displayed. This unpredictability in the object dependency graph of a web page incurs delays in the rendering page [9, 33, 35]. This situation is exacerbated by the fact that web pages are getting more and more complex every day.

2.2 Existing Solutions And Related Work

To overcome the challenges of poor web performance in developing countries, several systems and techniques have been proposed in recent years to optimize web browsing over poor network connections, including network-level optimizations, caching techniques, and content distribution mechanisms [15, 16, 18, 19, 26, 34]. Recent works [9, 33, 36] have focused on the complexity of web pages and suggested different approaches to address them. A system named Klotski [9] proposed re-prioritizing web content relevant to a user's preferences by enabling fast selection and load-time estimation for the subset of resources to be prioritized. Another recent system, Shandian [36], proposed restructuring the web page load process and exercising control over what portions of the web page get communicated and in what order. They chose a split-browser architecture to deploy their solution. Polaris [33] uses fine-grained object dependency graphs to load the objects in the optimal way for the browser to enable fast loading of the page.

Many solutions have been recently proposed to optimize the usage of JavaScript in modern web pages. For example, JSCleaner [13] leverages a rule-based solution to identify and block non-essential JavaScript elements. Such elements are non-critical to the user experience, given that their code does not contain any functions that handle page content or functionality. Similarly, SlimWeb [12] proposed a machine learning technique to classify JavaScript elements into several categories based on categories identified by experts in the web community [7]. SlimWeb suggests blocking JavaScript categories related to *Advertising*, *Analytic*, and *Social* in order to speed up the web browsing experience.

Another solution, Muzeel [29], focuses on optimizing JavaScript code by identifying and eliminating *deadcode*. That is, JavaScript code that is present in the JavaScript general-purpose libraries but not used by the web page. Muzeel utilizes a novel interaction bot to emulate user interactions, which in turn helps in identifying JavaScript functions that can safely be removed without affecting the user experience.

2.3 Developing regions related work

In the context of developing regions, recent and noteworthy commercial attempts have been made by Google AMP [2], Facebook Instant articles [3], and Opera mobile browser [4]. AMP introduced new HTML tags and elements that are used to manage resource loading. AMP enables parallel download of objects by leveraging iframes and by declaring the size and position of objects upfront. AMP also uses caches to ensure that pages satisfy all AMP specifications and enable pre-rendering of pages. AMP caches provide the point where other optimizations such as reordering HTML tags, rewriting JavaScript URLs, compression, and other image operations are performed. Facebook Instant Articles is a platform within the mobile Facebook application created by Facebook for publishers to create fast and interactive articles on Facebook. Facebook claims instant articles make the loading of articles ten times faster than the loading of standard mobile articles. Facebook enables faster access to instant articles by preloading the data. On tapping the article, only very small data that is not previously preloaded is downloaded, and hence the article loads faster than a standard mobile article. Opera Mobile browser and its lower-end sibling, Opera Mini browser, request web pages through a proxy server called Opera Turbo. Opera Turbo compresses web pages by about 90% rendering them faster by two to three times. Opera has another extreme compression method called Mini mode. It has a higher compression ratio than Turbo mode, but with a loss of certain functionality in the web page rendered. Also, Opera Mini Browser has an in-built adblocker, which further prunes content sent to the mobile phone. As of August 2017, most of the market for Opera browsers is from developing countries with slower internet, like India, Bangladesh, Indonesia, South Africa, Nigeria, Ukraine, Tanzania, and Pakistan.

The design philosophy of MAML fundamentally differs from these works in that it aims to pre-compile web pages to simplify the HTML representation of a page. MAML explicitly eliminates recursive handling of objects in a page and simplifies the DOM (Document Object Module) representation to a simpler, flatter layout while maintaining functional equivalence with the original page. We believe that reliance on HTML, JavaScript, and CSS is the underlying problem, and none of the above-mentioned solutions tackles this fundamental issue. While Google AMP comes really close to simplifying the web page and doing several optimizations, it still relies heavily on HTML.

3 MAML DESIGN

This section describes MAML, a simplified web specification language that is designed to create flatter web pages for users in developing regions. Unlike hierarchical DOM structures present in today's web, MAML uses absolute positions to place elements on the viewport. This approach significantly reduces the depth of the DOM tree and eliminates the need for complex calculations to determine the layout, further reducing the computational load on the device. Finally, MAML does not allow any JavaScript to be used, thus eliminating a lot of today's page complexities.

MAML is designed to be lightweight and modular by eliminating unnecessary elements and attributes that contribute to the bloat of modern web pages. The MAML pages have a more efficient page structure. This modularity also allows for easier maintenance, as changes can be made to individual components without affecting the entire web page. MAML is built to be fully backward compatible with HTML and, hence, can eventually run on any of today's web browsers.

3.1 Data Structure

MAML introduces a new format for writing web pages based on the flat individual element approach, where each element would retain all the necessary information and attributes related to itself in a self-contained dictionary-like representation. For example, the following is an example that illustrates the general format of MAML:

```
{
 "type":"img","w":268,"h":31,"x":336,"y":15,"z":1,
 "src":"https://example.com/img/abc.webp",
 "alt":"Alternate Text","fit":"fill"
}
```

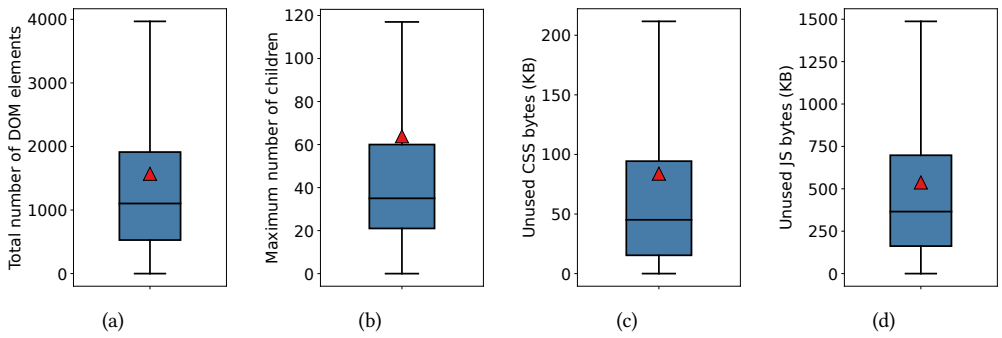


Figure 1: Web complexity for the top 1000 popular websites according to the tranco list

Each line of the MAML data structure is a hash map containing key-value pairs, where the key represents an element's property and the value is the property's assigned value. The use of a hash map data structure ensures that accessing the value of a property has a time complexity of *O(1)*. In the above example, the MAML page has a single element of type "image", with attributes related to the position of where that image should be displayed on the web page viewport (i.e., the x and y coordinates of the upper left corner pixel of the image). In addition, MAML also specifies the z coordinates to establish element order in terms of depth, whereas the size of the displayed element is represented by the width (w) and height (h). Finally, the image element also specifies the URL to the image source as well as the alternative text.

The resulting MAML file (or the MAML version of a web page) is a collection of MAML data structures separated by a newline character (\n) and has an extension of *.maml*.

3.2 Supported Elements

In designing MAML, our focus is to support a wide variety of web page components. As such, MAML currently supports the following list of elements:

1. **Text**: This element is used to add textual content on the web page. It can be customized with different fonts, sizes, colors, and alignments to match the design of the page.
2. **Shape**: This element is used to create geometric shapes such as rectangles and ellipses. It can be utilized for decorative purposes or as a background for other elements.
3. **Text Field**: This element provides an input field for users to enter text. It is commonly used for forms, search bars, and other interactive components that require user input.
4. **Button**: This element creates a clickable button that can trigger actions or navigate to different pages.
5. **Dropdown**: This element allows the creation of a dropdown menu with a list of options. It is useful for navigation menus, filters, or any other scenario where a selection from multiple options is required.
6. **Image**: This element is used to display images on the webpage. It supports PNG, JPEG, and WEBP image formats.
7. **Carousel**: This element creates a slideshow or carousel of images that are automatically cycled through. It is often used for showcasing multiple images in a limited space.
8. **Video**: This element enables the embedding of video content into the web page.
9. **Script**: This element allows adding MAMLScript, a scripting language we have developed to be used with the MAML data structure (See Section 3.4). It is used to add dynamic content updates on the web page.

3.3 Properties of a MAML Element

3.3.1 Mandatory Properties. Each MAML element contains a few mandatory properties. Table 1 shows these properties alongside their descriptions.

Property	Description
type	type of element
x	x-position of element in pixels
y	y-position of element in pixels
z	z-position of element as integer
w	width of element in pixels
h	height of element in pixels
display	whether to make the item visible or not

Table 1: Mandatory Properties of a MAML Element

3.3.2 Additional Properties. Based on the type of element, each MAML element has its own set of additional properties, as demonstrated in Table 2.

Element	Available Properties
text	id, text, fontFamily, textAlign, fontSize, color, fontStyle, fontWeight, textDecoration, display
shape	id, backgroundColor, borderRadius, display
text-field	id, placeholder, backgroundColor, display
button	id, text, display
dropdown	id, options, display
image	id, img, objectFit, display
carousel	id, img, display
script	code

Table 2: Additional Properties of MAML Elements

3.3.3 Exceptions. In order to support additional functionality and interactivity within the MAML specifications, a few exceptions are required. These exceptions are:

- The *script* element does not include the required properties: x, y, z, w, h, and display. This is because these values are of no use to the script, given that it is not a visible element and is not associated with a location on the screen.
- The *src* property of an *img* element can optionally be another hash map with two keys - *source* & *thumbnail*, each containing URL values. The *source* stores the URL of the image in its original size, while *thumbnail* stores the URL of a compressed version of the same image to be used as a thumbnail. If the value of the *src* property is a string instead of a hash-map, same URL is used for both source and the thumbnail.

3.4 MAMLScript

Given that MAML's main goal is aimed at reducing the complexity of web pages and providing a more flatter structure to these pages, we decided not to support JavaScript in MAML. There have been many studies in the literature that show the complexity of JavaScript on today's web, and how it is one of the main reasons behind a bloated web, especially in the context of developing regions [10–13, 29]. Instead, we opted to design a simpler scripting language that can support a limited set of functionality and page interactivity, which we call "MAMLScript".

MAMLScript is included at the end of the MAML file within a *script* element that has a property named *code*. The value of this property contains the MAMLScript code. When the MAML file is parsed, this element gives information about the dynamic updates applied to various elements. This method simplifies the mapping of actions to specific page components, thereby facilitating a more streamlined interactivity framework. The following example demonstrates the script element:

```
{
 "type":"script",
 "code":"MAMLScript is included here."
}
```

3.4.1 Structure. The structure of MAMLScript closely mirrors familiar programming constructs found in languages like JavaScript, making it accessible to a broad range of developers. This design choice not only reduces the learning curve but also enables developers to leverage their existing coding skills when working with MAML files.

Below is an example of a MAMLScript:

```
on("click", "button1") {
    show("image2");
    hide("image1");
    swap(val("input3"), "text3");
}
```

In this example, the MAMLScript configures a sequence of actions that are executed in response to the click event on "button1". Upon activation, the script first makes `"image2"` visible using the `show("image2")` trigger. Concurrently, it hides `"image1"` from view with the `hide("image1")` trigger, ensuring that `"image2"` takes its place on the screen. Finally, the script swaps the text of the element `"text3"` to the value of the text input field `"input3"` using the `swap(val("input3"), "text3")` trigger.

3.4.2 Listeners and Triggers. MAMLScript uses an Event-Driven Programming (EDP) paradigm. Each functionality is determined by using one listener followed by one or more triggers. In the same example mentioned in Section 3.4.1, *on* is used to listen to an event "click" on element id "button1". Three functions–`show`, `hide`, and `swap`–are then triggered on "image2", "image1", and "text3" one by one. Additionally, MAMLScript supports nested triggers—triggers that return values can be used as a value for another trigger as illustrated by the `swap` and `val` triggers used together in the same example.

Table 3 shows the available listeners with their usage, and Table 4 shows the available triggers with their usage.

Listener	Usage
click	on("click", *element_id*) { [triggers...] }
change	on("change", *element_id*) { [triggers...] }
keydown	on("keydown", *element_id*, *key_name*) { [triggers...] }
reach	on("reach", *element_id*) { [triggers...] }
timer	on("timer", *seconds*) { [triggers...] }

Table 3: Listeners and their Usage

Trigger	Usage
val	val(*element_id*);
show	show(*element_id*);
hide	hide(*element_id*);
swap	swap(*content, element_id*);

Table 4: Triggers and their Usage

3.4.3 Supported Functionalities. Currently, MAMLScript supports the following functionalities that can be used in conjunction with each other:

(1) Listening to a click event.
(2) Listening to changes in value of input elements.
(3) Listening to a key press.
(4) Listening for a user reaching a specific location on the page by scrolling.
(5) Listening to the completion of a countdown timer.
(6) Retrieving the value from an input field or a dropdown selector.
(7) Toggling the visibility of an element on or off.
(8) Swapping the content of an element with a different value.

These functionalities support interactive features that are prioritized in web development, based on our evaluation of the top 100 websites from Tranco's list[30]. For instance, responding to user inputs and interaction events such as clicks, key presses, and changes in input fields enables the creation of responsive interfaces that dynamically react to user actions. This is crucial for forms, real-time validations, and interactive controls.

The following were identified to be the most commonly used interactive features in our evaluation, with the first seven features currently supported by MAML:

(1) Dropdown menu
(2) Loading new content when the page reaches its bottom
(3) Video player
(4) Carousel
(5) Component that appears after scrolling below a certain point
(6) Countdown timer
(7) Notification window
(8) Animation triggered by scrolling
(9) Auto-animation
(10) Toggle button that changes page theme
(11) Display a video preview when hovering over the thumbnail the thumbnail

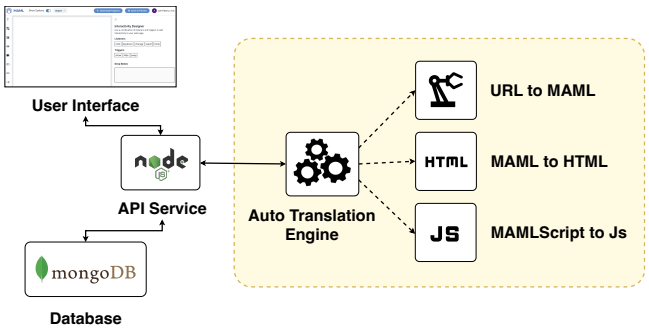


Figure 2: MAML Editor's Architecture

3.4.4 Future Considerations. As MAMLScript continues to evolve, the goal is to standardize it as a robust, developer-friendly language that can serve as a standard for interactivity in web development. To this end, future versions are expected to include more sophisticated features such as the ability to make API requests and enhanced error handling capabilities. MAMLScript might include capabilities for dynamic styling changes akin to CSS pseudo-classes. This means that MAMLScript could allow developers to define conditions under which certain styles are applied, mimicking the behavior of CSS pseudo-classes like :hover or :active, which change styles based on user interactions. These advancements aim to make MAMLScript not only a tool for creating interactive content but also a comprehensive solution for developing complex web applications that are resource efficient.

4 MAML EDITOR

To enable the creation of web pages that adhere to MAML specifications, a specialized web page editor is essential. Existing commercial web page editors lack the necessary customization options, such as the ability to export web pages in a specific format. Therefore, we have developed a dedicated MAML Editor that facilitates the creation of web pages using the MAML format. This can simplify and facilitate the easy creation of MAML pages for web developers who are interested in creating MAML pages.

The MAML Editor (see Figure 3) is a web application designed for both experienced and inexperienced web developers, meaning that no prior web development experience is required to use the editor. The editor features a drag-and-drop interface to design the layout of the web page and add interactivity to the elements. In this section, we discuss the user workflow and the implementation of the MAML Editor.

4.1 User Workflow

To begin creating pages with the MAML Editor, users are required to sign in using Google OAuth 2.0 (or "Sign in with Google"). This login step is crucial, as it enables the storage of

their last saved work and ensures a seamless user experience. Once logged in, users can: 1) create a MAML page from scratch; 2) import an existing *.maml* file from their local machine and customize it; or 3) import a URL that converts an existing public web page to the MAML format using the "Auto Translation Engine" (See Section 4.2.4). To design the layout, users are required to drag and drop elements from the sidebar on the left. Once an element is dropped, users see options to change the position, styles, and additional properties of the element.

To add interactivity to the elements, users can use the "Interactivity Designer". Users are required to drag and drop listeners and triggers to add event-driven behavior to the elements. For example, a user can set up a button to hide a specific image when clicked. The interactivity designer provides a visual interface for defining these interactions, making it easy for users to add dynamic functionality to their web pages.

Once the page is complete, users can either: 1) export the page into a MAML file or 2) save their page to the cloud and preview an equivalent HTML file generated by parsing the MAML file.

4.2 Implementation

We implemented the MAML Editor as a web application that can run on any modern web browser. We used a microservices architecture as demonstrated in Figure 2, primarily consisting of a User Interface (UI) developed using `Next.js` and `TypeScript`, a `Node.js API service` to handle APIs and authentication, a `MongoDB` database for data storage, and an `Auto Translation Engine` developed using Python and Selenium [5] for converting existing web pages into the MAML format. In this section, we describe the design and implementation of each service in detail.

4.2.1 User Interface. The MAML Editor's user interface is inspired by other popular web editors, such as Wix [38], Elementor [21], Webflow [37], etc. Figure 3 shows a sample screenshot of the MAML editor user interface. It features a large canvas (Figure 3a) with an initial size of *1200px x 800px*, whose height can be increased as elements are added. A toolbar (Figure 3b) is positioned on the left side of the canvas, allowing users to add various elements to their page. On the right side, an "Interactivity Designer" (Figure 3f) is placed to help users add dynamic content updates.

The header at the top contains a few useful tools: "Import" (Figure 3c), "Download Progress" (Figure 3d), and "Save & Preview" (Figure 3e). Users can import a MAML file or a MAML-converted version of an existing web page using the import menu. The download progress button creates a MAML file of the page the user is designing, allowing them to save their progress on their local machine. The save & preview button saves the MAML page to the cloud and opens an HTML preview of the page in a new tab.

The interactivity designer (Figure 3f) can be opened using a toggle button placed on the right side of the canvas. Users can drag listeners and triggers into the interactivity designer to add event-driven behavior to the elements without needing to write a single line of MAMLScript code. Available listeners include: *click*, *change*, *keydown*, *reach*, and *timer*, each with specific functionalities. For example, the *click* listener can be used to execute specified triggers when an element is clicked. Similarly, users can utilize various triggers such as *val*, *show*, *hide*, and *swap* to manipulate the content or visibility of elements. For instance, the *show* trigger can be used to make an element visible. These triggers can be combined with listeners in the interactivity designer to create interactive behaviors on the web page, all through a user-friendly drag-and-drop interface.

4.2.2 Web Server. The backend web server is primarily responsible for managing APIs that facilitate several key functions:

1. **Authenticate:** This API authenticates a Google user by verifying a token retrieved by using the "Sign in with Google" feature and returns a JSON Web Token (JWT) for the user interface to use for all other API requests.
2. **Upload Image:** This API manages the upload and storage of images used in the MAML pages. All uploaded images are stored in an AWS S3 bucket.
3. **Translate:** This API interacts with the Auto Translation Engine (see Section 4.2.4), which is responsible for converting existing web pages into the MAML format, and converting a MAML page to an equivalent HTML file.
4. **Save:** This API allows users to save their MAML pages. The saved progress is stored in a MongoDB database, enabling users to continue their work from where they left off, across different devices or sessions.

4.2.3 MongoDB Database. The MAML Editor uses a MongoDB database to store authenticated user information and the pages created by them. Each user document includes a unique user identifier, along with metadata such as their email address and the date of account creation. The pages are stored as separate documents, each containing the MAML representation of the page, the user's identifier, and timestamps for creation and last modification.

4.2.4 Auto Translation Engine. This engine is designed to streamline the process of importing existing web pages into the MAML Editor. It can also convert a MAML page to its equivalent HTML page and transpile MAMLScript to JavaScript.

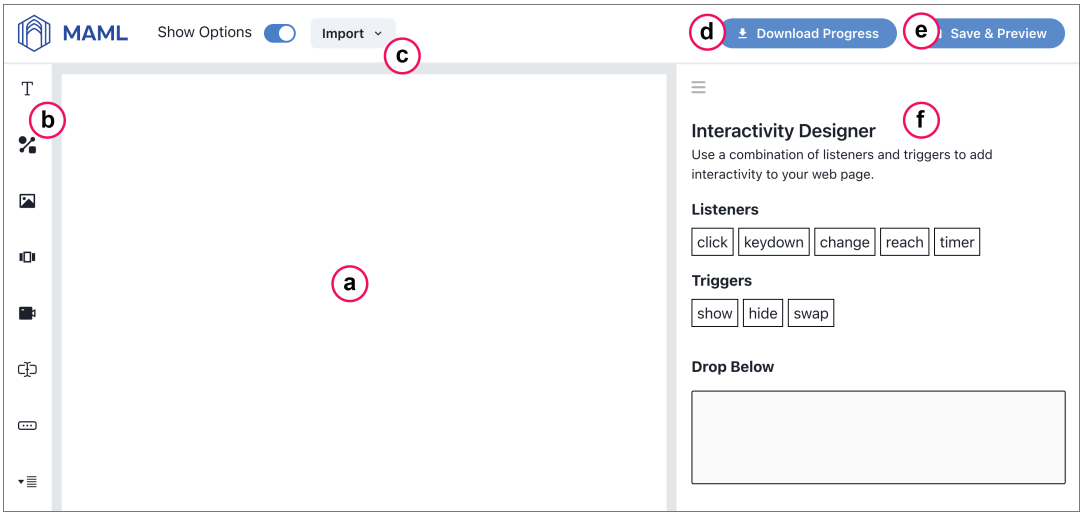


Figure 3: MAML Editor's User Interface

The workflow of the Auto Translation Engine begins when the user inputs the URL of a web page in the *Import -> Import from URL* section of the MAML Editor. The translator initiates a Selenium Google Chrome instance on the server and navigates to the specified web page. To ensure that all resources, especially those on websites using lazy loading, are fully loaded, the script first performs a sequential scroll through the entire web page, from top to bottom, and then returns to the top. Once the entire web page has loaded, the engine conducts a Depth-First Search (DFS) of all HTML elements on the page and extracts necessary information from them. For each supported element, the engine converts it into the corresponding MAML format, while simultaneously recording the element's two-dimensional layout coordinates on the page and its hierarchical positioning in terms of stacking order relative to other elements.

The generated MAML file is then stored on the server, and its corresponding public URL is returned in the API response, enabling it to be imported into the MAML Editor's user interface.

5 EVALUATION

In order to evaluate MAML, we organized a MAML competition aimed at creating the best-looking MAML page in terms of how closely it resembles the original version of the page, as well as in terms of the functionality and interactive elements that mimic the original page. We recruited 20 students from an internationally recognized university to participate in the competition. The competition was conducted in an asynchronous manner, i.e., students were allowed to work on the creation of MAML pages on their own over the course of two weeks. Each student was given a list of pre-determined web pages that we manually vetted to be suitable for the competition, and they were asked to create as many MAML pages as possible as they wanted. The students were given a short introduction to how to use the MAML editor, along with a few instructions.

The primary objectives of the competition were to: 1) assess the performance improvements of MAML pages; 2) evaluate the visual fidelity of MAML pages in comparison to the original pages; 3) test the effectiveness of the MAML editor's drag-and-drop functionality in web page construction; and 4) identify any potential improvements for the editor.

5.1 Competition Setup

To engage students, we distributed posters across campus and posted in the university's "Room of Requirement" Facebook group. Interested participants were invited to fill out a registration form. The competition offered incentives, including prizes such as an iPhone 13, an iPad, and AirPods for the first, second, and third place winners, respectively. We received 20 registrations, which we subsequently emailed five URLs to recreate using the MAML Editor.

5.2 Web Pages Selection Criteria

We announced that recreating each page would take between 30 minutes and an hour. To ensure this timeframe, we compiled a list of websites from various sources ([30–32]) and manually verified each URL by opening it in a browser to confirm that it could be completed within the specified time. A final list of web pages was then assigned to the participants for recreation.

5.3 Pre and Post Competition Survey

Before beginning the creation of the assigned web pages, participants were required to fill out a form asking about their expertise in web development and how important page load time is for them in building a website. We found that the largest percentage of participants were beginners, comprising 31.6%. Moreover, the largest number of participants responded with either 4 or 5 (see Figure 4), meaning that it's crucial for them when building a website. Table 5 shows the distribution of participants' web development skills. Appendix B describes the details of the survey.

Table 5: Participants' web experience

	None	Beginner	Intermediate	Advanced
%	7.9%	31.6%	31.6%	28.9%

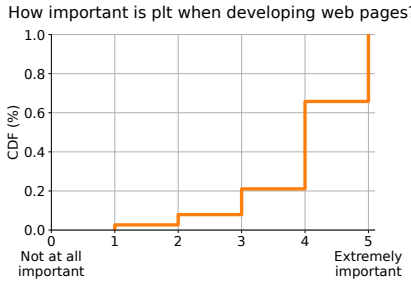


Figure 4: Pre-survey question on the importance of page load time

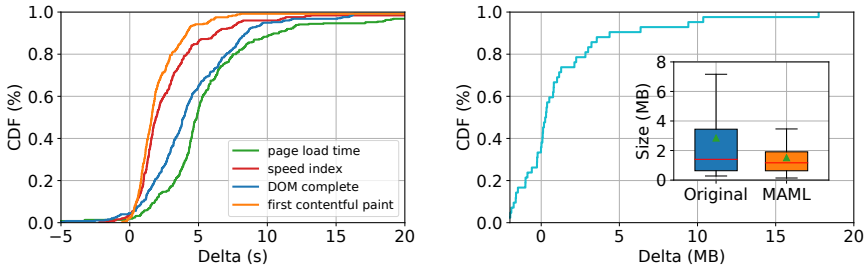
5.4 Quantitative Evaluation

To evaluate the quantitative gains in terms of page load times speedups, and bandwidth savings, we compared the 50 MAML pages against their original counter parts. We utilized the `webpagetest` tool to automate the loading of the pages. Each page was loaded three times over a network configuration of 12 Mbps downlink/uplink rates and 70 ms round-trip-time. Figure 5a shows the cumulative distribution function (cdf) of the delta difference between the original and their MAML pages counterparts. As can be seen, there are four different cdfs each corresponding to a different timing metric: first contentful paint (orange), speed index (red), DOM complete (blue), and page load time (green). The result demonstrates that, on median, MAML is able to reduce the pages' metrics significantly compared to their original versions, achieving a speed index reduction of approximately 2 seconds and a page load time reduction of 5 seconds. As for the page size and bandwidth savings, Figure 5b shows that the MAML pages have a lower average page size compared to their original ones, at least for more than 25 of the 50 created MAML pages. The average page size was reduced from 3 MB down to about 1.7 MB (as can be seen in the boxplots). The reason why some of the MAML pages had a slightly higher page size (left size of the cdf curve below a delta of zero) was due to the fact that our participants did not optimize for the images, where some participants took screenshots of the images from the original pages and saved them in png file format, which increased the overall page size. Despite the larger page size, MAML still had lower page load times across all pages compared to their original versions.

5.5 Qualitative Evaluation

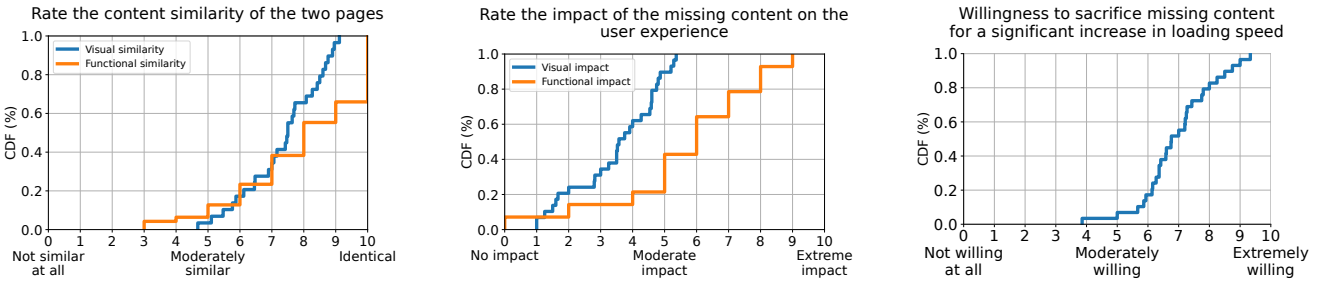
In the previous section, we showed that MAML achieves significant improvements over both the web pages' timing metrics as well as the page size. However, it is crucial to study the impact of this on the quality of the generated pages. For example, it is important to compare the visual similarity between the original pages and the MAML ones. In addition, it is also important to evaluate the functionality and interactivity of the MAML pages when compared to the original ones. As such, we first conducted a study using the popular crowd-sourcing tool Prolific[1], to ask participants to compare screenshots of MAML pages to their original counterparts. The participants were asked to rate the content similarity between the two on a scale from 0 to 10, and were also asked to assess if there was any missing content in the MAML pages. Figure 6(a) shows the cdf of the participants' responses (depicted in blue). The results show that for all pages, the participants rated the pages as at least "moderately similar" or more. In fact, the median rating was 7.5, indicating a high visual similarity of the MAML pages to the original ones. In order to assess the functional similarity, we evaluated the MAML pages ourselves, given that this task is not an easy one and requires a lot of expert attention. The result of this analysis is depicted by the orange cdf in Figure 6(a). Here, 95% of the MAML pages had a moderately functional similarity, with a median score of 8, indicating again a high functional similarity.

Regarding missing content, 50% of the participants indicated the presence of missing content in the MAML pages. However, when we asked them to score the impact of that missing content, the median score was around 3.5, indicating a low impact. In fact, no missing content was ranked as having more than moderate impact (blue cdf in Figure 6(b)). Similarly, we also evaluated the impact of the missing page functionality and found that was below moderate in half of these pages, and was higher in the other half (orange cdf in Figure 6(b)). Finally, we asked the participants if they were willing to tolerate missing content in favor of faster page



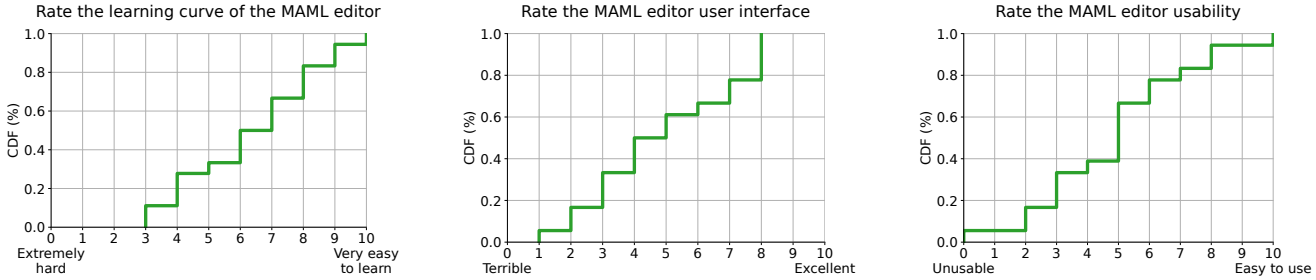
(a) Timing metrics (b) Page size

Figure 5: Quantitative analysis: delta differences between MAML and Original webpages



(a) Visual similarity (b) Missing content impact on user experience (c) Willingness to tolerate missing content in favor of faster loading speed

Figure 6: MAML webpages vs. Original webpages in terms of content similarity



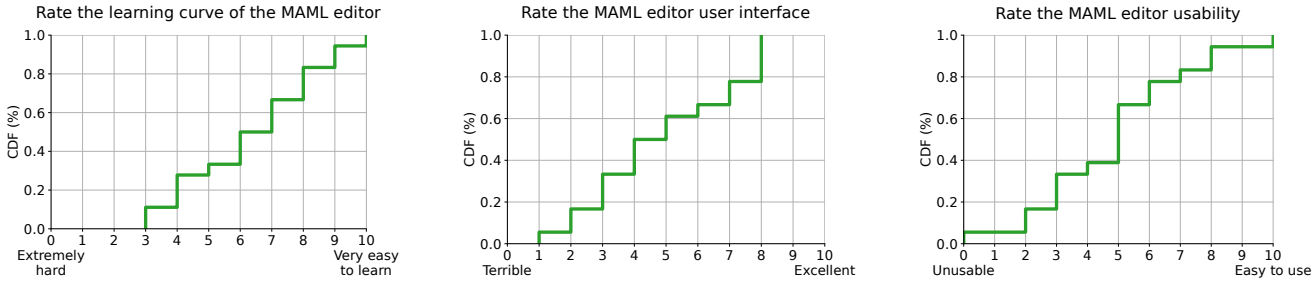
(a) Learning curve (b) Web interface (c) Overall usability

Figure 7: MAML Editor post competition survey results

load times, and in fact, almost all were moderately willing to do so with a median score of 7 (see Figure 6(c)).

5.6 Survey Results

In this section, we wanted to evaluate the competition participants' experience with the MAML editor. As such, we asked the participants to rank the editor's learning curve, the quality of the web interface, and the overall usability of the MAML editor. Figure 8 shows the cdfs of the rating scores for the above questions. The results show that the participants thought that the MAML editor was easy to learn (median score of 7), with slightly lower rating scores on the user interface of the editor (median score of 5). Finally, the participants ranked the that overall usability of the tool relatively high, with more than 60% of them giving a score of 5 and above.



(a) Learning curve (b) Web interface (c) Overall usability

Figure 8: MAML Editor post competition survey results

6 CONCLUSION

In this paper, we introduced the Mobile Application Markup Language (MAML), a web specification language developed with a minimalist flat layout. MAML is specifically designed to reduce computational demands and data transmission needs, facilitating faster loading of web pages in developing regions. The creation of an intuitive web-based user interface for MAML allows developers to easily convert existing complex web pages into MAML's streamlined format. This user interface is crucial for broader adoption and practical usability, enabling developers to make significant contributions to the digital landscape in resource-constrained environments.

Furthermore, we developed an innovative feature within the MAML Editor that automates the conversion of traditional web pages into the MAML format. This automation enables rapid deployment of optimized pages that require minimal customization, thus streamlining the process significantly. Our comprehensive testing and analysis have demonstrated that web pages converted to MAML format not only load faster but also consume less data. These benefits are particularly valuable in bandwidth-constrained regions, potentially transforming web access and usability.

Lastly, our analysis through a user study involving web developers confirms the practical effectiveness and ease of use of the MAML Editor. The positive feedback from developers underscores MAML's potential impact on real-world web development practices, highlighting its significance as a transformative tool for global digital inclusivity. This study paves the way for further research and development in optimizing web technologies to better serve diverse global communities.

REFERENCES

[1] [n.d.]. Prolific | Quickly find research participants you can trust. https://www.prolific.com/. Accessed: 2024-05-15.

[2] 2017. *7 Ways AMP Makes Your Pages Fast.* https://www.youtube.com/watch?v=9Cfxm7cikMY

[3] 2017. *Introducing Instant Articles.* https://media.fb.com/2015/05/12/instantarticles/

[4] 2017. *Opera Browser Advanced Documentation.* http://www.opera.com/docs/

[5] 2017. *SeleniumHQ Browser Automation.* http://www.seleniumhq.org/about/

[6] last accessed: 2024. CITI Program: Research, ethics, and compliance training. https://about.citiprogram.org/.

[7] HTTP Archive. 2019. Third Parties | 2019 | The Web Almanac by HTTP Archive. https://almanac.httparchive.org/en/2019/third-parties. Accessed: 2020-01-2.

[8] Michael Butkiewicz, Harsha V. Madhyastha, and Vyas Sekar. 2011. Understanding Website Complexity: Measurements, Metrics, and Implications. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference* (Berlin, Germany) *(IMC '11)*. ACM, New York, NY, USA, 313–328. https://doi.org/10.1145/2068816.2068846

[9] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V Madhyastha, and Vyas Sekar. 2015. Klotski: Reprioritizing Web Content to Improve User Experience on Mobile Devices.. In *NSDI*. 439–453.

[10] Moumena Chaqfeh, Rohail Asim, Bedoor AlShebli, Muhammad Fareed Zaffar, Talal Rahwan, and Yasir Zaki. 2023. Towards a world wide web without digital inequality. *Proceedings of the National Academy of Sciences* 120, 3 (2023), e2212649120.

[11] Moumena Chaqfeh, Russell Coke, Jacinta Hu, Waleed Hashmi, Lakshmi Subramanian, Talal Rahwan, and Yasir Zaki. 2022. Jsanalyzer: A web developer tool for simplifying mobile web pages through non-critical javascript elimination. *ACM Transactions on the Web* 16, 4 (2022), 1–31.

[12] Moumena Chaqfeh, Muhammad Haseeb, Waleed Hashmi, Patrick Inshuti, Manesha Ramesh, Matteo Varvello, Fareed Zaffar, Lakshmi Subramanian, and Yasir Zaki. 2021. To Block or Not to Block: Accelerating Mobile Web Pages On-The-Fly Through JavaScript Classification. *arXiv preprint arXiv:2106.13764* (2021).

[13] Moumena Chaqfeh, Yasir Zaki, Jacinta Hu, and Lakshmi Subramanian. 2020. JSCleaner: De-Cluttering Mobile Webpages Through JavaScript Cleanup. In *Proceedings of The Web Conference 2020*. 763–773.

[14] Josiah Chavula, Nick Feamster, Antoine Bagula, and Hussein Suleman. 2015. *Quantifying the Effects of Circuitous Routes on the Latency of Intra-Africa Internet Traffic: A Study of Research and Education Networks*. 64–73. https://doi.org/10.1007/978-3-319-16886-9_7

[15] Jay Chen, David Hutchful, William Thies, and Lakshminarayanan Subramanian. 2011. Analyzing and Accelerating Web Access in a School in Peri-urban India. In *Proceedings of the 20th International Conference Companion on World Wide Web* (Hyderabad, India) *(WWW '11)*. ACM, New York, NY, USA, 443–452. https://doi.org/10.1145/1963192.1963358

[16] Jay Chen, Russell Power, Lakshminarayanan Subramanian, and Jonathan Ledlie. 2011. Design and Implementation of Contextual Information Portals. In *Proceedings of the 20th International Conference*

Companion on World Wide Web (Hyderabad, India) *(WWW '11)*. ACM, New York, NY, USA, 453–462. https://doi.org/10.1145/1963192.1963359

[17] Jay Chen, Lakshmi Subramanian, Janardhan Iyengar, and Bryan Ford. 2014. TAQ: Enhancing Fairness and Performance Predictability in Small Packet Regimes. In *Proceedings of the Ninth European Conference on Computer Systems* (Amsterdam, The Netherlands) *(EuroSys '14)*. ACM, New York, NY, USA, Article 7, 14 pages. https://doi.org/10.1145/2592798.2592819

[18] Jay Chen, Lakshminarayanan Subramanian, and Jinyang Li. 2009. RuralCafe: Web Search in the Rural Developing World. In *Proceedings of the 18th International Conference on World Wide Web* (Madrid, Spain) *(WWW '09)*. ACM, New York, NY, USA, 411–420. https://doi.org/10.1145/1526709.1526765

[19] Marshini Chetty, David Haslem, Andrew Baird, Ugochi Ofoha, Bethany Sumner, and Rebecca Grinter. 2011. Why is My Internet Slow?: Making Network Speeds Visible. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. ACM, New York, NY, USA, 1889–1898. https://doi.org/10.1145/1978942.1979217

[20] Marshini Chetty, Srikanth Sundaresan, Sachit Muckaden, Nick Feamster, and Enrico Calandro. 2013. Measuring Broadband Performance in South Africa. In *Proceedings of the 4th Annual Symposium on Computing for Development* (Cape Town, South Africa) *(ACM DEV-4 '13)*. ACM, New York, NY, USA, Article 1, 10 pages. https://doi.org/10.1145/2537052.2537053

[21] Elementor Ltd. [n.d.]. Elementor. https://www.elementor.com. Accessed: 2024-05-12.

[22] Rodérick Fanou, Pierre Francois, and Emile Aben. 2015. *On the Diversity of Interdomain Routing in Africa*. 41–54. https://doi.org/10.1007/978-3-319-15509-8_4

[23] Rodérick Fanou, Gareth Tyson, Pierre Francois, and Arjuna Sathiaseelan. 2016. Pushing the Frontier: Exploring the African Web Ecosystem. In *World Wide Web Conference (WWW)*.

[24] J Gilmore, N Huysamen, and A Krzesinski. 2007. Mapping the african internet. In *Proceedings Southern African Telecommunication Networks and Applications Conference (SATNAC), Mauritius*.

[25] Arpit Gupta, Matt Calder, Nick Feamster, Marshini Chetty, Enrico Calandro, and Ethan Katz-Bassett. 2014. Peering at the internet's frontier: A first look at isp interconnectivity in Africa. *Passive Active Measurement Conference (PAM)* (2014), 204–213.

[26] Sibren Isaacman and Margaret Martonosi. 2009. The C-LINK System for Collaborative Web Usage: A Real-World Deployment in Rural Nicaragua.

[27] Michael Kende and Bastiaan Quast. 2016. Promoting Content In Africa. *ISOC Report* (2016).

[28] Zahir Koradia, Goutham Mannava, Aravindh Raman, Gaurav Aggarwal, Vinay Ribeiro, Aaditeshwar Seth, Sebastian Ardon, Anirban Mahanti, and Sipat Triukose. 2013. First Impressions on the State of Cellular Data Connectivity in India. In *Proceedings of the 4th Annual Symposium on Computing for Development* (Cape Town, South Africa) *(ACM DEV-4 '13)*. ACM, New York, NY, USA, Article 3, 10 pages. https://doi.org/10.1145/2537052.2537064

[29] Jesutofunmi Kupoluyi, Moumena Chaqfeh, Matteo Varvello, Russell Coke, Waleed Hashmi, Lakshmi Subramanian, and Yasir Zaki. 2022. Muzeel: Assessing the Impact of JavaScript Dead Code Elimination on Mobile Web Performance. In *Proceedings of the 22nd ACM Internet Measurement Conference* (Nice, France) *(IMC '22)*. Association for Computing Machinery, New York, NY, USA, 335–348. https://doi.org/10.1145/3517745.3561427

[30] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 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)*. https://doi.org/10.14722/ndss.2019.23386

[31] Christian I Mejía-Escobar, Miguel Cazorla, and Ester Martinez-Martin. 2024. Web Pages Dataset. https://doi.org/10.17605/OSF.IO/7GHD2. https://doi.org/10.17605/OSF.IO/7GHD2

[32] Bezhan Mukhidinov. 2020. List of Top 1000 Websites. https://gist.github.com/bejaneps/ba8d8eed85b0c289a05c750b3d825f61.

[33] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. 2016. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking.. In *NSDI*. 123–136.

[34] William Thies, Janelle Prevost, Tazeen Mahtab, Genevieve T. Cuevas, Saad Shakhshir, Alexandro Artola, Ro Artola, Binh D. Vo, Yuliya Litvak, Sheldon Chan, Sid Henderson, Mark Halsey, Libby Levison, and Saman Amarasinghe. 2002. Searching the World Wide Web in Low-Connectivity Communities.

[35] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2013. Demystifying Page Load Performance with WProf. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation* (Lombard, IL) *(nsdi'13)*. USENIX Association, Berkeley, CA, USA, 473–486. http://dl.acm.org/citation.cfm?id=2482626.2482671

[36] Xiao Sophia Wang, Arvind Krishnamurthy, and David Wetherall. 2016. Speeding up Web Page Loads with Shandian.. In *NSDI*. 109–122.

[37] Webflow, Inc. [n.d.]. Webflow. https://www.webflow.com. Accessed: 2024-05-12.

[38] Wix.com Ltd. [n.d.]. Wix. https://www.wix.com. Accessed: 2024-05-12.

[39] Yasir Zaki, Jay Chen, Thomas Pötsch, Talal Ahmad, and Lakshminarayanan Subramanian. 2014. Dissecting Web Latency in Ghana. In *Proc. of the ACM Internet Measurement Conference (IMC)*. Vancouver, BC, Canada.

A ETHICS

An institutional review board (IRB) approval is granted to conduct the user study, and the authors who conducted the study are CITI [6] certified.

B SURVEY QUESTIONNAIRE

Tables 6, 7, 8, and 9 show the questions asked in the questionnaire with the available answer options.

Question	Options
How much web development experience do you have?	A. None B. Beginner (understand the basics, can use templates and customize them) C. Intermediate (can develop pages from scratch and write limited JavaScript code for interactivity) D. Advanced (have developed web pages from scratch using modern web development technologies and can write JavaScript code from scratch)
How important is page load time for you when developing web pages? Rate on a scale from 0 to 5.	0 - Not at all important 5 - Extremely Important

Table 6: Pre-competition survey questionnaire

Question	Options
How would you rate the learning curve of the MAML Editor on a scale from 0 to 10?	0 - Extremely Hard 10 - Very easy to learn
Rate MAML Editor's web interface on a scale from 0 to 10.	0 - Terrible 10 - Excellent
Rate the usability of the MAML editor on a scale from 0 to 10.	0 - Unusable 10 - Easy to use

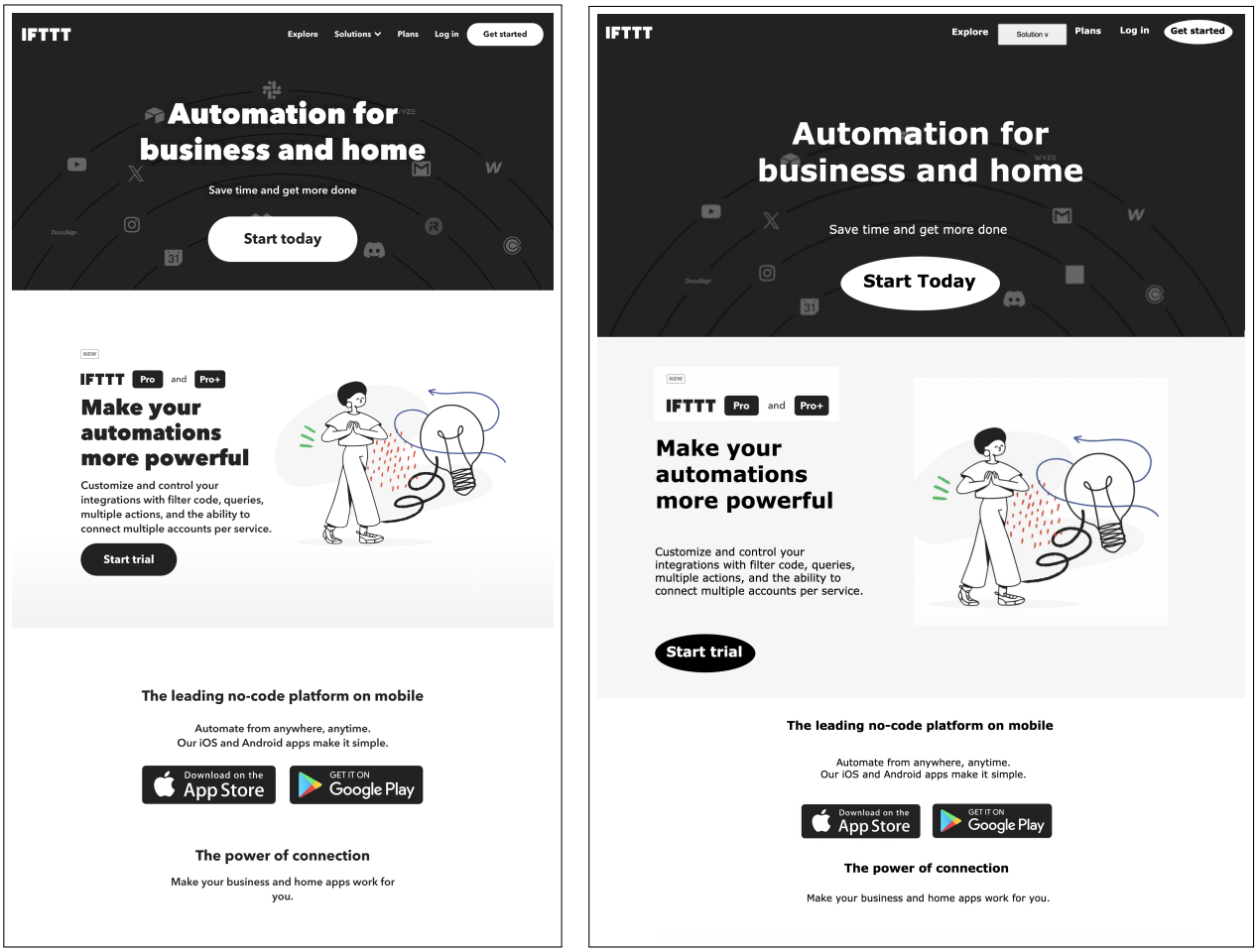
Table 7: Post-competition survey questionnaire

Question	Options
Rate the visual similarity of the two pages on a scale from 0 to 10.	0 - Not similar at all 5 - Moderately similar 10 - Identical
Rate the visual impact of the missing content on the user experience on a scale from 0 to 10.	0 - No impact 5 - Moderate impact 10 - Extreme impact
Rate your willingness to sacrifice missing content for a significant increase in loading speed.	0 - Not willing at all 5 - Moderately willing 10 - Extremely willing

Table 8: Content similarity study questionnaire on Prolific

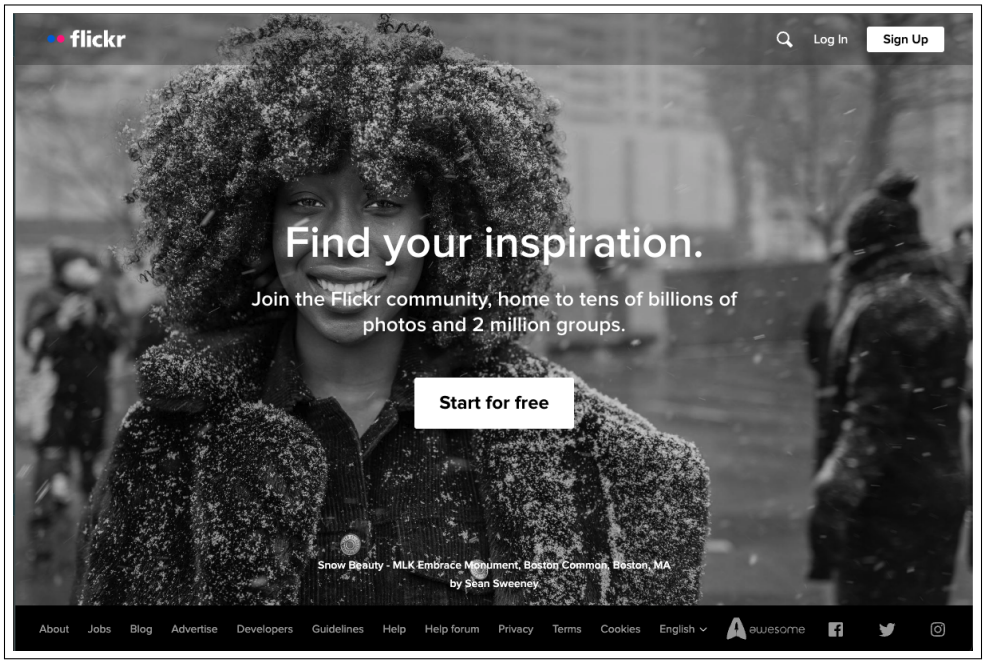
Question	Options
Rate the functional similarity of the two pages on a scale from 0 to 10.	0 - Not similar at all 5 - Moderately similar 10 - Identical
Rate the functional impact of the missing content on the user experience on a scale from 0 to 10.	0 - No impact 5 - Moderate impact 10 - Extreme impact

Table 9: Functional similarity study questionnaire for manual inspection

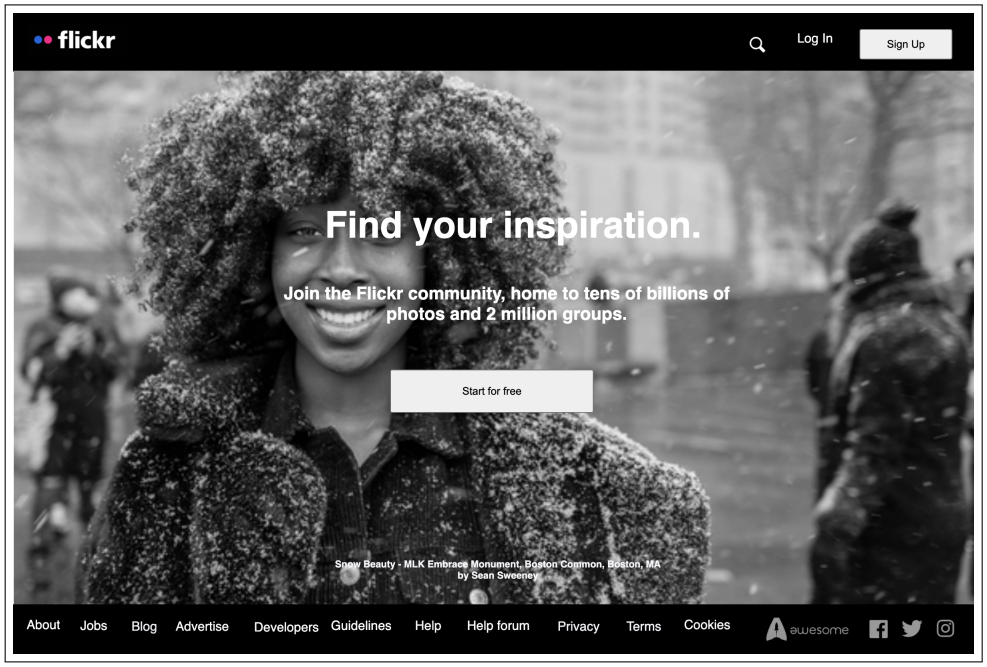


(a) Original page (b) MAML page

Figure 9: Original page vs MAML-converted page of ifttt.com

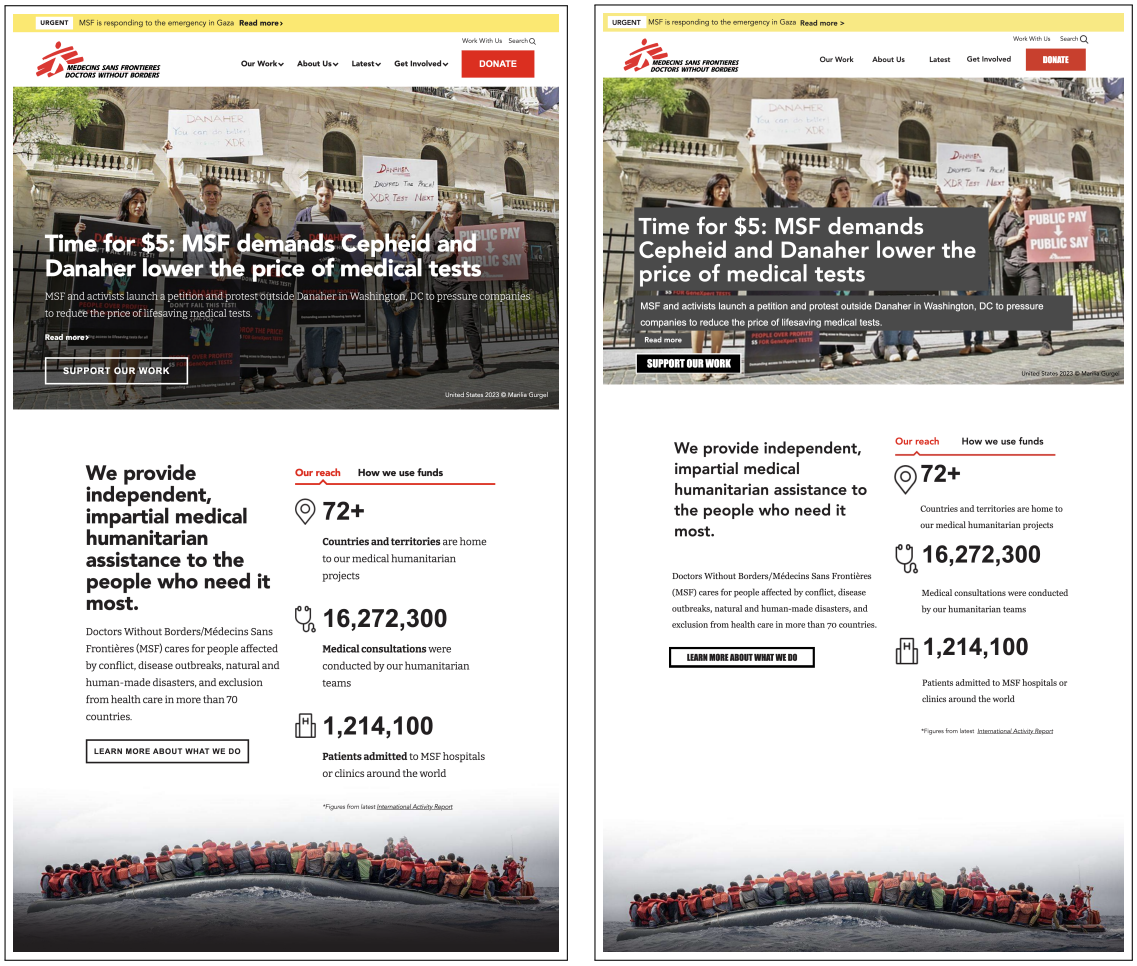


(a) Original page



(b) MAML page

Figure 10: Original page vs MAML-converted page of flickr.com



(a) Original page (b) MAML page

Figure 11: Original page vs MAML-converted page of doctorswithoutborders.org