

# Evaluating the Use of LLMs for Automated DOM-Level Resolution of Web Performance Issues

Gideon Peters  
Concordia University  
Montreal, Canada  
gi\_peter@mail.concordia.ca

SayedHassan Khatoonabadi  
Concordia University  
Montreal, Canada  
sayedhassan.khatoonabadi@concordia.ca

Emad Shihab  
Concordia University  
Montreal, Canada  
emad.shihab@concordia.ca

## Abstract

Users demand fast, seamless webpage experiences, yet developers often struggle to meet these expectations within tight constraints. Performance optimization, while critical, is a time-consuming and often manual process. One of the most complex tasks in this domain is modifying the Document Object Model (DOM), which is why this study focuses on it. Recent advances in Large Language Models (LLMs) offer a promising avenue to automate this complex task, potentially transforming how developers address web performance issues. This study evaluates the effectiveness of nine state-of-the-art LLMs for automated web performance issue resolution. For this purpose, we first extracted the DOM trees of 15 popular webpages (e.g., Facebook), and then we used Lighthouse to retrieve their performance audit reports. Subsequently, we passed the extracted DOM trees and corresponding audits to each model for resolution. Our study considers 7 unique audit categories, revealing that LLMs universally excel at SEO & Accessibility issues. However, their efficacy in performance-critical DOM manipulations is mixed. While high-performing models like GPT-4.1 delivered significant reductions in areas like *Initial Load*, *Interactivity*, and *Network Optimization* (e.g., 46.52% to 48.68% audit incidence reductions), others, such as GPT-4o-mini, notably underperformed, consistently. A further analysis of these modifications showed a predominant additive strategy and frequent positional changes, alongside regressions particularly impacting *Visual Stability*. Our findings define safe areas for automation (e.g., SEO and accessibility) and reveal the limits of DOM-level resolution, underscoring the need for hybrid, validated workflows. However, it critically underscores the need for careful model selection, understanding their specific modification patterns, and robust human oversight to ensure reliable web performance improvements.

## CCS Concepts

• **Software and its engineering** → **Automated analysis**.

## Keywords

Generative AI, HTML Understanding, Web Application, Web Performance, Performance Engineering

## 1 Introduction

Web applications have become one of the primary ways users consume content on the internet [9]. Therefore, the importance of performant web applications cannot be overemphasized [42]. A webpage's performance is a core non-functional requirement, as it impacts the overall user experience, engagement, and conversion ratios [23, 57]. As such, web performance engineering remains an unnegotiable component of the web development process. It

requires a deep understanding of both the browser engine and application use cases [62]. Performance optimization involves various considerations including hardware (CPU and memory usage), server (API response times), and client-side factors (DOM size, image optimizations, on-demand loading, and omni-channel experience) [13, 50].

This study focuses on the client-side, specifically the Document Object Model (DOM), which is central to how browsers interpret, render, and interact with webpages [7, 66]. The DOM also significantly impacts hardware and server performance—complex DOM structures increase CPU and memory usage, slowing performance—especially on resource-limited devices. Additionally, large DOM payloads can strain server response times [13, 28, 50]. Optimizing the DOM is challenging [60], requiring detailed analysis and targeted modifications to balance functionality and performance. Traditionally, addressing DOM inefficiencies has relied on manual interventions and automated tools with limited scope [40]. However, the growing complexity of web applications demands more sophisticated solutions [9].

Large Language Models (LLMs) present a promising approach to address these challenges. They have transformed numerous software engineering tasks by leveraging their ability to understand and generate human-like text [30, 32, 33]. Trained on massive corpora, including HTML documents from public repositories, LLMs are uniquely positioned to tackle challenges in web development [11, 19]. Their applications extend beyond code generation to include tasks like web security [61], automated testing [45], and accessibility improvements [39]. However, the effectiveness of LLMs for web performance optimization, particularly in modifying the DOM to address performance issues, has not yet been systematically explored.

To fill this knowledge gap, we aim to explore the usefulness and challenges of using LLMs for automating web performance resolutions. For this purpose, we extract the DOM trees of 15 popular webpages, and we generate audit reports for these extracted DOM trees. We then assess the effectiveness of nine state-of-the-art LLMs, including GPT-4.1, Claude 3.7 Sonnet, DeepSeek R1 & V3, and GPT-4o-mini—to resolve these audits by passing the audit along with the DOM tree to the model. Finally, we generate new audit reports for the modified DOM trees and compare the incidence of the initial audits before and after modification. In summary, we aim to answer the following research questions:

### RQ1: How effective are LLMs in resolving web performance issues through automated DOM-level modifications?

Performance optimization can be tedious, requiring web developers to run performance tests, and implement required fixes [31, 43]. We explore the ability of LLMs to resolve

performance issues identified by Lighthouse audits across 15 webpages. Our findings indicate that LLMs achieved a 100% reduction in SEO & Accessibility issues. However, for performance-critical issues, effectiveness was mixed and highly model-dependent, with some models showing significant gains while others notably introduced regressions, particularly impacting visual stability.

**RQ2: What is the nature of changes made by LLMs for automated DOM-Level resolution of web performance issues?** To understand how LLMs modify the DOM, we analyzed differences between the original and LLM-modified HTML pages for nine state-of-the-art models across 15 webpages. We identified modifications including element and attribute additions, removals, type changes, and positional shifts. Most LLMs used a predominantly additive strategy, with GPT-4o-mini uniquely removing more elements than it added. Frequent positional changes also occurred, typically at shallower DOM depths.

Our study offers insights for web developers, LLM providers, and researchers on automating web performance remediation. We focus on DOM-level edits to compare models under identical, language-agnostic inputs and maintain reproducibility across sites. This baseline is essential since every web stack ultimately renders to the DOM [38]. However, performance regressions can also stem from the interaction of DOM, CSS, and JavaScript; thus, our findings should be applied conservatively to SPA or dynamic contexts and verified through CI/CD checks. Focusing on DOM-level changes enables direct, interpretable fixes that integrate seamlessly into existing workflows while avoiding build- or server-side confounds [45].

**Our Contributions.** In summary, we make the following contributions in this paper:

- We conducted extensive experiments using nine LLMs on DOM trees from 15 popular webpages, providing a comprehensive evaluation of their effectiveness in automated web performance issue resolution.
- We provide a token-aware chunking strategy for DOM trees based on a predefined token threshold to enable processing by LLMs for the task of web performance issue resolution.
- We identified seven distinct audit categories, and provide a detailed quantitative analysis of LLM changes implemented with respect to these audits.
- We synthesize actionable insights and implications for web developers, LLM providers, and the research community, guiding future development towards more robust and reliable AI-driven web performance optimization.
- To promote the reproducibility of our study and facilitate future research on this topic, we publicly share our scripts and dataset online [6].

**Paper Organization.** The rest of the paper is organized as follows. Section 2 outlines the necessary background for our study. Section 3 details the study design. Sections 4 and 5 examine the effectiveness of LLMs in resolving web performance issues in the DOM and the nature of changes implemented. Section 6 discusses our findings. Section 7 outlines the study limitations. Section 8 reviews related literature and Section 9 concludes the paper.

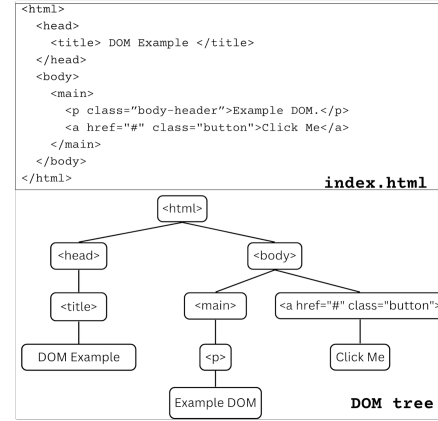


Figure 1: Example structure of a DOM tree

## 2 Background

This section introduces key concepts and technologies relevant to our study.

### 2.1 DOM Trees

The Document Object Model (DOM) represents a webpage’s structure and content as a tree-like hierarchy of nodes, where each node corresponds to an HTML element, attribute, or text [67]. This hierarchy enables programmatic access and manipulation of webpage elements. The DOM is fundamental to web development, allowing dynamic updates and interaction with web content. Through DOM manipulation, developers can: (i) dynamically alter webpage structure, style, and content, and (ii) respond to user interactions. Figure 1 illustrates a DOM tree, showing its HTML code representation on the top and its hierarchical structure on the bottom.

It originates with the `<html>` element as its **root node**, which serves as the parent to both the `<head>` and `<body>` elements. This hierarchy extends through **parent-child relationships**, where elements are nested within one another (e.g., `<body>` typically contains `<header>`, `<main>`, and `<footer>`). Additionally, nodes at the same hierarchical level exhibit **sibling relationships**, such as `<p>` and `<a>` elements found within a `<main>` section.

Furthermore, DOM trees contain various element types: **Text** for content (e.g., *Example DOM* in `<p>Example DOM</p>`); **Comment** for unrendered documentation (e.g., `<!-- -->`); **Tag** as core HTML elements (e.g., `<p>`) with optional attributes (e.g., *class* in `<a class="button">Link</a>`); **Script** for embedding or referencing JavaScript (e.g., `<script src="app.js"></script>`); and **Stylesheet** for CSS rules (e.g., `<style>p { color: red; }</style>`).

These element types are combined in various ways and for different purposes across webpages [15]. In performance optimization, DOM tree size and complexity significantly impact webpage load time and responsiveness, as larger and more deeply nested trees demand greater computational resources for rendering and processing [27, 35].

## 2.2 Performance Audits

Performance audits systematically evaluate webpages to assess their performance, identify bottlenecks, and recommend improvements [58]. These audits aim to ensure sites meet performance goals, such as fast load times, smooth interactions, and efficient resource usage. These are critical for user experience, search engine rankings, and business outcomes [31].

Our study utilizes Lighthouse, an open-source Chromium-based tool developed by Google for the measurement and improvement of webpage performance [24, 25, 55]. We selected Lighthouse due to its extensibility and widespread community adoption [21, 31, 42]. It analyzes webpages, generating actionable reports on performance, accessibility, SEO, and progressive web apps. Lighthouse offers numerous configuration flags allowing tailored audits for specific use cases, environments, or requirements. Lighthouse audit reports include key performance metrics like First Contentful Paint (FCP), Largest Contentful Paint (LCP), and Cumulative Layout Shift (CLS) [14, 31, 42]. Each audit highlights an issue or suggestion based on standard practices and is keyed by its name and contains the following properties:

- **id**: A unique identifier identical to the audit key.
- **title**: A brief summary of the audit.
- **description**: A detailed explanation of what the audit assesses and its significance.
- **score**: A numeric or categorical value indicating the audit’s result.
- **scoreDisplayMode**: Denotes how the score is interpreted, with values; informative, notApplicable, manual, or error).
- **displayValue**: A contextual measurement supplementing the score.
- **details**: Provided when an audit fails, offering insight into the issue and potential resolutions. This may include responsible element types, value headings, or affected items like specific DOM elements, location parameters, resource URLs, or data points.

## 3 Study Design

In this section, we describe our dataset, the performance audits, the environment configurations, considerations for our LLM selection, the chunking strategy, and the evaluation metric used in this study.

### 3.1 Dataset

To conduct our study, we first select 15 real-world webpages at random from the Alexa Top 500 list [5], which features top-ranked webpages on the web. We chose this list due to its popularity and prior use in research [12, 41, 46]. Each webpage selected is a homepage, the main entry point for users. Since homepages typically receive the highest traffic [26, 49, 59], optimizing their performance is particularly relevant to our study.

Our dataset comprises webpages from four different categories: *Shopping webpages* (4), *Professional webpages* (2), *Social webpages* (6), and *Entertainment webpages* (3). The full breakdown can be found in our replication package [6]. Table 1 presents descriptive statistics for the webpages in our dataset. DOM Tree Depth, the maximum depth of nested HTML elements, ranging from 4 to 32, indicating diverse structural complexity. The number of chunks (# chunks), varying from 2 to 17, reflecting varied content modularity across webpages. *Total Audits* averaged 25.5 from 18 to 39, providing

substantial per-page data. Finally, *LHS* (Lighthouse Score) averaged 44.7%, ranging from 12% to 90%, highlighting significant variability in webpage performance.

**Table 1: Descriptive Statistics of the Webpages in Our Dataset**

Statistic	Mean	Minimum	Maximum
DOM Tree Depth	18	4	32
# Chunks	5.2	2	17
Total Audits	25.5	18	39
LHS (%)	44.7	12	90

Figure 2 shows the entire workflow for our experiments. For each webpage, it comprises the following six main stages:

- (1) **DOM Extraction**: We begin by extracting *original DOM trees* from the webpage. Python’s requests package was used to fetch webpages, and their DOM trees were then extracted and parsed with BeautifulSoup [51].
- (2) **DOM Chunking**: To accommodate the LLMs’ context window and output token limitations, we split the DOM tree into smaller chunks to obtain the *original DOM chunks*.
- (3) **Initial Audit Report Generation**: The extracted DOM tree is then passed to Lighthouse to generate *initial audit reports*, which establish a benchmark for issues to be resolved by the LLMs.
- (4) **LLM Modification**: Each chunk is then provided to the LLM, along with the corresponding audit reports, instructing it to make modifications to resolve the identified issues. This process is applied to every chunk with the LLM returning the *modified DOM chunk* in every iteration.
- (5) **Re-assembly**: After processing all original chunks, the modified chunks are reassembled into a complete *modified DOM tree*.
- (6) **Post-Modification Audit Report Generation**: A subsequent audit report is generated from this reassembled tree to capture the *audits after LLM modification*. This allows for a quantitative comparison between the initial audit reports and those obtained from the modified DOM trees to assess the LLM’s effectiveness.

The subsequent sections provide a detailed explanation of this workflow and our considerations.

### 3.2 LLM Selection

To evaluate the potential of LLMs in automating web performance issue resolution, we selected a diverse set of state-of-the-art LLMs that vary in reasoning capability, architecture, context window limits, max output tokens, and model size. Table 2 provides a detailed list of the models and their specifications. Our goal is to compare how different LLMs process DOM trees in conjunction with performance audits, and to understand how model characteristics influence the types and quality of generated modifications, thereby enhancing the generalizability of our findings. All LLMs used zero-shot prompting with the same prompt structure (provided in the replication package). We fixed temperature = 0.0 to minimize variance [22]. Max output tokens and chunk sizes followed the smallest-window model (GPT-4o-mini) to ensure fairness across models.

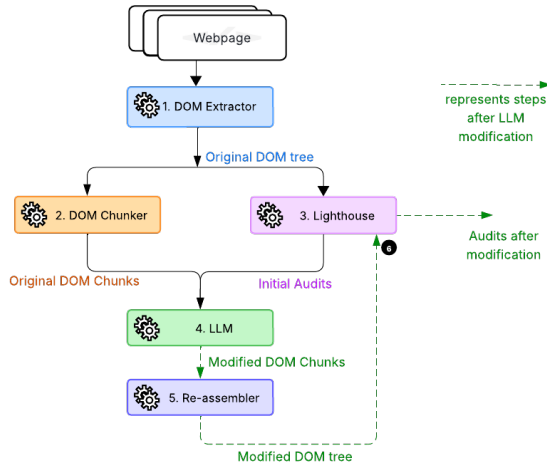


Figure 2: Overview of our experiment workflow

Table 2: LLM Models Evaluated and Their Specifications

Model	Reasoning	Max O.T.	Context Window	Size
Claude 3.7 Sonnet (R)	Yes	128K	200K	–
Claude 3.7 Sonnet	No	128K	200K	–
DeepSeek V3	No	32K	131K	671B
DeepSeek R1	Yes	32K	128K	685B
Llama3.3 70B	No	40K	128K	70B
GPT-4.1	No	32K	1M+	–
o4-mini	Yes	100K	200K	–
GPT-4o-mini	No	16K	128K	–
Qwen2.5 32B-Instruct	No	128K	131K	32B

### 3.3 DOM Chunking

To accommodate the varying output token limits of the LLMs in our evaluation, we employ a conservative chunking strategy. This approach is based on the model with the smallest maximum output size, specifically GPT-4o-mini, which has an output token limit of 16K [47]. This ensures all models can process identical DOM chunks for a consistent and fair comparison.

We implement this by limiting DOM chunks to 15K tokens, reserving 1K tokens for LLM-induced modifications. Token counts are estimated using OpenAI’s Tiktoken package [48]. Chunks exceeding 15K tokens are recursively split to ensure all webpage elements are assessed.

This strategy prevents token truncation, incomplete responses, resource inefficiencies during inference, and error propagation in downstream evaluations. Our approach traverses the DOM tree in a depth-first search, grouping nodes into chunks that never exceed the 15K threshold. This is done with attention to semantic structure and element types; for example, specific preservation strategies were applied to certain element types:

- **Text Nodes:** We preserved these text elements as-is, explicitly excluding them from any splitting or chunking operations to maintain the integrity of inline text content.

- **Comment:** HTML comments were left untouched to avoid losing useful annotations or developer metadata. We applied no chunking or transformations to these nodes.
- **Tag:** Before splitting, we stored all initial tag attributes for comparison and use during reassembly, ensuring tags and their associated attributes remained intact. The split was then performed recursively, accurately representing every element and its descendants. Each chunk was uniquely identified by a UUID to ensure accurate reassembly.
- **Script:** We also stored script elements before chunking, re-incorporating them during HTML reassembly. This preserved the logic and interactivity defined by scripts.
- **Stylesheet:** Similarly, style rules were stored before chunking and merged back during reassembly. Preserving these styles maintains the visual fidelity of the webpage.

To validate the integrity of our chunking strategy, we reassembled all chunks before any modifications, confirming that the structure and content remained unchanged. To ensure there are no unintended alterations, we checked a popular metric known as **Tree Edit Distance** to quantify the difference between the original and reassembled DOMs [36, 56]. For all webpages processed, we observed a **Tree Edit Distance of 0**. This indicates that the reassembled DOMs were identical to the original DOMs, with no structural or content alterations.

### 3.4 Performance Audits

To create an initial benchmark for what issues we attempt to resolve, we generate Lighthouse audit reports for the DOM trees before they are modified by the LLMs. We also generate audit reports for the LLM-modified DOM trees. These are used in our quantitative analysis. For the audit generation process, we make use of the following Lighthouse configuration flags:

- **headless:** Allowing Chrome to operate without a Graphical User Interface (GUI) and ensuring lower consumption of CPU and memory resources. Consequently, it facilitates the automation of the Lighthouse analysis limiting any interactions with the webpages during the process.
- **no-sandbox:** This disables the sandbox feature in Chrome which isolates web content and process. This is useful for our study as it bypasses security restrictions that may come up in our environment that could affect how pages are rendered.
- **disable-gpu:** Forcing the browser to render pages using the CPU instead of the GPU. This is to ensure consistency of results based on the allocated CPU resource in our environment.

From the audit reports generated, we exclude audits with *score-DisplayMode* values of *notApplicable*, *manual*, and *informative*, or binary scores of 1 as it indicates a pass, and these audits do not require any resolution in the DOM [14], unlike a binary score of 0, which indicates a needed resolution. This resulted in 67 *unique audits*, each of these audits as well as their descriptions can be found in our replication package [6]. All of these audits were manually analyzed, and a classification was agreed upon by two authors, and any conflicts were resolved by the third author. This resulted in the establishment of seven audit categories. The categories are as follows:

- **Initial Load Performance:** Describes how quickly a page’s essential content loads, e.g., “*First Contentful Paint*” measures the time it takes for the first text or image to appear on the screen.
- **Interactivity Performance:** Focuses on how responsive the page is to user interactions, e.g., “*Time to Interactive*” measures when the page becomes fully interactive, indicating when a user can reliably interact with the page.
- **Runtime Performance:** Assesses how efficiently JavaScript and other resources are executed during runtime. e.g., “*JavaScript Execution Time*” measures the duration of JavaScript operations and their impact on page speed.
- **Resource Optimization:** Evaluates the effectiveness of resource usage such as scripts, images, and stylesheets, e.g., “*Unminified JavaScript*” flags large, uncompressed JavaScript files that could be optimized to reduce their size and improve performance.
- **Network Optimization:** Measures the efficiency of network requests, including the number of requests and their size, e.g., “*Reduce Server Response Time*” focuses on reducing latency and optimizing server performance to decrease load time.
- **Visual Stability:** Focuses on preventing unexpected layout shifts during page load, e.g., “*Cumulative Layout Shift*” tracks the unexpected shifting of elements as the page loads, impacting the user experience.
- **SEO & Accessibility:** Covers audits related to SEO and accessibility, e.g., “*Accessibility Improvements*” flags issues that affect the usability of the website for users with disabilities, such as missing aria labels. These audits are relevant from a semantic point of view as opposed to the hierarchical context of the DOM [54].

In this paper, we use *performance-critical* to mean audits that directly affect user-perceived latency, responsiveness, runtime efficiency, network/asset delivery, and layout stability. Concretely, this includes Initial Load Performance, Interactivity Performance, Runtime Performance, Resource Optimization, Network Optimization, and Visual Stability. By contrast, SEO & Accessibility are treated as semantic audits rather than performance-critical.

### 3.5 Environment Configurations

To ensure the reproducibility and consistency of our performance audits, we utilized a Docker-isolated environment [44]. This approach mitigates variability stemming from diverse hardware configurations (CPU, RAM, GPU, etc.) by providing a standardized execution context, a common practice in web development for consistent builds.

Our Docker environment was hosted on a MacBook Pro 2018 featuring a 2.3GHz Quad-Core Intel Core i5 processor and 16GB of RAM. Key software versions used include Docker v27.0.3, Node v21.5.0, Lighthouse 12.2.0, and the python:3-9-slim Docker image runtime. The Docker container was allocated 1GB of shared memory to ensure sufficient resources for the Chromium browser used by Lighthouse.

### 3.6 Benchmarking & Evaluation Metric

To calculate the distribution of performance issues across our dataset, we introduce the derived *audit incidence ratio (AIR)* metric. It is a practical adaptation of reporting practices in tools like Lighthouse, which summarize how often specific audits are detected across sites

to inform optimization priorities [24]. The *AIR* of an audit provides a quantitative measure of the extent to which it is observed in the dataset. A higher ratio indicates that the audit is more prevalent and affects a larger portion of the webpages, suggesting it could be a critical area to address for overall performance improvement. In contrast, a lower *AIR* suggests that the issue affects fewer webpages. We define it as follows:

$$AIR = \frac{W_a}{W_{total}} \quad (1)$$

where  $W_a$  is the number of unique webpages containing audit  $a$ , and  $W_{total}$  is the total number of webpages in the dataset.

To benchmark our approach, we compare the original *AIR* for the extracted DOM trees with the *AIRs* observed after applying modifications implemented by the LLMs. We call this comparison the percentage change in *AIR*, calculated as follows:

$$\% \text{ change in AIR} = \frac{M - O}{O} \times 100 \quad (2)$$

## 4 RQ1: How effective are LLMs in resolving web performance issues through automated DOM-level modifications?

**Motivation.** The iterative and time-consuming nature of web performance optimization presents a significant challenge for developers and negatively impacts user experience when performance is poor [8, 9, 17, 31, 46]. Investigating how LLMs emergent capabilities could automate the resolution of these issues offers a promising avenue for significant advancement in web development—our work specifically validates their ability to resolve web performance issues by making necessary changes to the DOM. We study LLM-driven DOM-level resolution to web performance issues, isolating this layer to avoid confounding build or server-side effects.

**Approach.** To assess the effectiveness of LLMs in resolving web performance issues in the DOM, we utilized the audits generated for the originally extracted webpages (see Section 3). Subsequently, each webpage is split into chunks to address LLMs’ output token limitation, as detailed in our study design (Section 3). For each webpage, we then iteratively pass each chunk and the performance audits to the LLMs. Through zero-shot prompting [34], we instruct the LLMs to make the necessary changes to resolve contributing factors to these issues. Our prompt includes the audit key, title, description, and details (if any). Our replication package contains the full prompt structure utilized [6].

Our prompt design specifically tailored the input to guide the LLMs in understanding complex DOM structures and performance audit requirements, considering their unique processing characteristics. It incorporates the following considerations:

- To address LLMs’ inherent output token limitations, we designed our input strategy to feed the DOM in chunks, ensuring the LLM understood the incremental nature of the content and avoided changes that could disrupt the hierarchical DOM structure.
- We explicitly requested the LLM to specify modified sections and describe the changes, a crucial step for validating LLM-generated modifications and facilitating easy identification of affected areas.
- We provide some context to the LLM about the possibility that the DOM tree being processed is likely to be minified, uglified,

or compressed as it is from a production website [63]. This is important as it lets the LLM know that some styles or scripts are already processed, hence further similar processing should be avoided to preserve the functionality of the webpage.

- We explicitly instruct the LLM to avoid any changes to the order, styles, and functionalities of the scripts present. This is done to preserve the core functionalities of the webpage.
- We constrain the LLMs to use the right formatting of modification comments in the respective sections, e.g., the HTML comment formatting for regular HTML elements and the style comment formatting for style scripts. This is done to avoid any parsing or build issues when the DOM tree is reassembled.

All modified chunks are then reassembled and a final audit report is generated on the updated webpage. This step helps to determine if the issues identified in the initial audit report have been resolved. To present this clearly, we conducted a quantitative analysis, comparing the audit reports of the original webpage with those of the modified webpage by calculating the % change in *AIR*.

**Results.** Table 3 highlights the percentage change in *AIRs* after LLM modification of the webpages in our dataset. It presents the various audit categories (see Section 3), the different models evaluated, and the percentage change in *AIR* after modification. Negative percentages indicate successful issue resolutions and positive percentages suggest a regression of webpage performance. To easily identify performance, the worst regressions in each audit category are colored **red**, and the best are **green**. Our findings are detailed below:

**Finding 1: LLMs universally excel at semantic understanding, resolving all SEO & Accessibility issues.** As shown in Table 3, all LLMs achieved a **100.00%** reduction for *SEO & Accessibility* audits. These types of issues typically do not rely on a comprehensive DOM context but rather on the semantic clarity and structural correctness of elements. This finding highlights the ability of LLMs to effectively understand and manipulate the semantic structure inherent within DOM elements, thus identifying and resolving issues crucial for SEO and web accessibility. This proficiency suggests that LLMs can be reliably employed to automate the correction of semantic markup, alt attributes for images [39, 69], ARIA roles, and other accessibility-related improvements without the need for exhaustive context. These capabilities are particularly valuable for maintaining compliance with accessibility standards and improving discoverability through search engines, aligning with prior research that emphasizes automated semantic validation and enhancement [68, 69].

**Finding 2: High-performing LLMs (e.g., Qwen2.5 32B-Instruct, GPT-4.1) deliver significant, broad latency and optimization gains.** Models such as **Qwen2.5 32B-Instruct** and **GPT-4.1** consistently demonstrated substantial improvements across multiple performance dimensions, positioning them as highly effective optimizers. These models, alongside **Claude 3.7 Sonnet(R)**, **Llama3.3 70B**, **Deepseek R1**, **Deepseek V3**, and **o4-mini** largely contributed to uniform decreases across all three latency audits, by substantial margins (Initial Load: from **-18.57%** to **-64.36%**; Interactivity: from **-5.00%** to **-64.99%**; Runtime: from **-11.11%** to **-88.65%**). While *Runtime performance* saw the most significant individual reductions (up

to **-88.65%** for **Qwen2.5 32B-Instruct**), the variability in improvements was notably higher (standard deviation  $\sigma = 40.3$ ) compared to Initial Load ( $\sigma = 22.8$ ) and Interactivity ( $\sigma = 24.1$ ), indicating that even among improving models, the degree of enhancement varied.

Beyond latency, these high-performing LLMs also delivered significant gains in web asset delivery and transfer efficiency. Our analysis reveals a largely positive trend with consistent improvements in *Network Optimization*. All evaluated LLMs demonstrated an ability to improve Network Optimization, with the reductions in *AIR* ranging from **-14.35%** (**Claude 3.7 Sonnet** and **Claude 3.7 Sonnet(R)**) to **-64.68%** (**Qwen2.5 32B-Instruct**). For *Resource Optimization*, most LLMs also delivered positive changes, with reductions ranging from **-4.94%** (**GPT-4o-mini**) to **-55.26%** (**Qwen2.5 32B-Instruct**). While **Claude 3.7 sonnet(R)** showed a slight increase of **+4.23%**, the overall trend for the majority was positive. These findings underscore that LLMs largely possess the capability to enhance web asset delivery and transfer efficiency, demonstrating their advanced ability to generate DOM modifications that lead to more efficient asset delivery and consumption.

**Finding 3: GPT-4o-Mini presents a unique case of performance regression, particularly for user-facing latency and resource efficiency.** In stark contrast to other LLMs, **GPT-4o-mini** consistently introduced significant overhead, leading to a notable *regression* in crucial user-facing latency metrics and resource efficiency. Specifically, **GPT-4o-mini** increased the *AIR* for all three user-facing speed audits: Initial Load (**+24.84%**), Interactivity (**+7.88%**), and dramatically for Runtime Performance (**+58.97%**). These increases represent the "Worst regression" cases for each of the audit categories. Manual inspection revealed these setbacks were primarily due to duplicated SVG path data that inflated payload size and paint cost, as well as the addition of new elements that bloat page sizes, indicating specific challenges this LLM faced in maintaining DOM integrity while optimizing. Furthermore, **GPT-4o-mini** contributed to increased visual instability (**+28.33%**), reinforcing its tendency to introduce unintended DOM changes that degrade user experience. These outcomes highlight that while larger LLMs can translate performance optimization prompts into tangible latency savings, smaller LLMs may introduce new bottlenecks rather than eliminate existing ones. Accordingly, any production pipeline that relies on automated web issue resolution should pair model selection with post-hoc validation to prevent unintended speed/latency regressions.

**Finding 4: Visual stability remains a significant challenge for most LLMs, with a majority introducing regressions.** The *Visual Stability* audit captures unexpected layout shifts that harm perceived smoothness and can lead to frustrating user experiences. While some LLMs demonstrated proficiency in optimizing load times and network efficiency, the ability to maintain or improve *Visual Stability* proved to be a pervasive challenge for the majority of LLMs. As shown in Table 3, a clear trend emerged: most of the LLMs evaluated introduced some visual instability. Most notably, **Deepseek R1** showed the most substantial regression in this category. Manual inspection revealed that many of these regressions stem from seemingly minor insertions or attribute changes that inadvertently alter element dimensions or flow. Common culprits include duplicated assets or scripts, changes in class names tied



**Table 3: Percentage change in audit incidence ratio results (-ve values indicate reductions in incidents and represent improvements; +ve values indicate increases in incidents and represent regressions).**

Audit Category	Claude 3.7 sonnet(R)	Claude 3.7 sonnet	Deepseek R1	Deepseek V3	GPT-4.1	GPT-4o-mini	Llama3.3 70B	o4-mini	Qwen2.5 32B-Instruct
SEO & Accessibility	-100.00	-100.00	-100.00	-100.00	-100.00	-100.00	-100.00	-100.00	-100.00
Initial Load Performance	-18.57	-14.72	-23.97	-30.67	-46.52	24.84	-28.71	-26.22	-64.36
Interactivity Performance	-5.00	6.09	-21.58	-21.09	-35.69	7.88	-54.69	-9.30	-64.99
Runtime Performance	-11.11	-9.13	-32.62	-47.54	-37.38	58.97	-76.15	-31.83	-88.65
Network Optimization	-14.35	-14.35	-33.40	-30.09	-48.68	-17.13	-35.05	-37.96	-64.68
Resource Optimization	4.23	-8.47	-34.58	-11.33	-32.28	-4.94	-30.70	-20.50	-55.26
Visual Stability	30.00	21.67	38.13	14.64	-22.02	28.33	-39.29	-6.03	-35.83

to cascading style sheet (CSS) styles, and the removal of scripts necessary for proper rendering. These issues highlight the inherent complexity of DOM manipulation and the difficulty LLMs currently face in consistently understanding spatial relationships within a webpage.

In contrast, four models achieved significant reductions in visual instability. The success of these models, particularly **Llama3.3 70B** with its nearly 40% reduction, suggests that careful, targeted chunk-level modifications by certain LLM architectures may preserve or even enhance visual stability. However, their performance stands as an exception rather than the norm in this evaluation. These findings strongly underscore the critical need for robust post-processing checks whenever LLMs are employed for any form of DOM modification.

**Answer to RQ1:** LLMs universally excel at semantic web issues, achieving a 100.00% reduction in SEO & Accessibility issues. For other audit categories, high-performing models (e.g., Qwen2.5 32B-Instruct, GPT-4.1) deliver significant gains, while notably, GPT-4o-mini consistently increased latency. The majority of LLMs introduced visual instability, underscoring the need for rigorous post-hoc validation for reliable web performance improvement.

## 5 RQ2: What is the nature of changes made by LLMs for automated DOM-Level resolution of web performance issues?

**Motivation.** Building on RQ1’s quantitative analysis of LLM effectiveness, RQ2 aims to understand the **nature of changes** LLMs make to the DOM. We saw LLMs achieve mixed results, with most improving performance while some introduced regressions ranging from increased latency to visual instability. This prompts a deeper dive into how LLMs modify the DOM and why specific outcomes occur. Investigating their DOM manipulations offers crucial insight into LLM’s “black box,” explaining effectiveness, revealing patterns, and informing future development, validation, and trust in automated web solutions.

**Approach.** To understand the nature of the LLM modifications, we parsed the DOM trees into structured JSON formats, we then generated detailed diffs between each original DOM tree and its modified version. This was done for the entire dataset. We used

the open-source Python package `Deepdiff`, a robust tool for quantifying and classifying differences in hierarchical data like JSON DOM trees [70]. This systematic approach allowed us to identify and categorize the specific modifications introduced by each LLM, offering fine-grained insights into their interventions. The primary categories of changes defined by `Deepdiff` that we extracted for our analysis were:

- **dictionary\_item\_added / dictionary\_item\_removed:** These identify the addition or removal of **attributes** (e.g., `id`, `class`, `src`) on an HTML element.
- **iterable\_item\_added / iterable\_item\_removed:** These signify the addition or removal of **child elements or text nodes** within a parent element’s content, often referred to as “element-level” or “node-level” changes.
- **type\_changes:** This category flags instances where the fundamental type of a DOM node (e.g., `<p>` node to a text node) was altered.
- **values\_changed** captures modifications to the content or properties of existing items that stay in place, broken down by type: **attr\_changes** (specific changes to existing attribute values, like `width="100"` to `width="50"`), **tag\_changes** (alterations to an element’s HTML tag name, e.g., `<div>` to `<p>`), **text\_changes** (modifications to textual content within an HTML element or standalone text node), and the critical **positional\_changes** (quantifying reordering or shifts in item placement within a sequence due to additions or deletions).

**Results.** For each LLM, we extracted the changes from all modified webpages, then grouped these changes by type, and finally summed the counts of each change type. We also report the depth of the changes performed by the LLMs across the dataset, as specified by `Deepdiff`. Table 4 highlights the categories of DOM changes; **Attributes** quantify **Added** (new attributes) and **Removed** (deleted attributes) on HTML elements; **Elements** summarize node-level modifications, covering **Added** and **Removed** elements; **Types Changed** indicate instances where a DOM node’s fundamental type was altered; **Change Depth** reports the **Min**, **Max**, and median (**Med**) depth of change within the DOM tree for all modifications; and **Values Changed** break down modifications to existing DOM elements by specific type: **Attr** (attribute value changes), **Tag** (HTML tag name changes), **Pos** (positional changes), and **Text** (text content modifications).

To quantify modification patterns, we introduce two metrics: the **Element Addition-to-Removal Ratio (EATRR)**, which indicates

potential DOM bloat ( $> 0.5$ ) or simplification ( $< 0.5$ ) by comparing added versus removed elements; and **Positional Change Dominance (PCD)**, which measures the proportion of value changes attributed to reordering elements, highlighting disruption to spatial relationships. Our findings are detailed below:

**Finding 5: Most LLMs predominantly employ an additive DOM modification strategy.** As shown in the EATRR column of Table 4, nearly all evaluated LLMs (except GPT-4o-mini and Deepseek R1) show a tendency to add significantly more elements than they remove. For instance, high-performing models like **Claude 3.7 Sonnet(R)**, **Deepseek V3**, **GPT-4.1**, and **o4-mini** exhibit EATRRs ranging from 0.85 to 0.89. This implies that these models, in their pursuit of performance optimization, frequently introduce new elements, rather than primarily refactoring or simplifying the existing DOM. While **Deepseek R1** presents a more balanced approach with an EATRR of 0.50, the overall inclination towards element addition suggests that these LLMs’ optimization often involves enriching the DOM.

**Finding 6: GPT-4o-mini’s exhibits a unique DOM modification strategy; removing more elements than it adds, coupled with the highest positional changes.** Among all evaluated models, **GPT-4o-mini** exhibits a distinctly unique DOM modification strategy. This strategy is characterized by the lowest EATRR at **0.44**, suggesting a tendency to remove or simply maintain existing element counts rather than an additive approach for optimization. Crucially, this is coupled with the highest PCD at **0.77**, indicating that a substantial majority of its modifications involve reordering or shifting existing elements. While other high-performing LLMs predominantly employ an additive strategy for optimization, GPT-4o-mini’s distinct profile means it frequently attempts optimization through extensive and disruptive changes in element positioning. Such large-scale positional changes are known to be costly, often triggering expensive browser reflows and repaints, which directly contribute to visual instability and a poor Cumulative Layout Shift (CLS) score [20, 38]. In contrast, models like Deepseek R1 (PCD 0.31, EATRR 0.50) and o4-mini (PCD 0.39, EATRR 0.89) exhibit different balances of positional changes and additive strategies. This unique combination of high, disruptive positional changes and a non-additive element strategy is a hallmark of GPT-4o-mini’s behavior, which could explain its poor performance in Finding 3.

**Finding 7: LLMs operate across varying DOM depths, often concentrated at shallower levels.** While LLMs make changes at various depths within the DOM tree, the median change depth across most models as shown in Table 4 ranges from 4-6, with minimum depths consistently at 1, and maximums ranging from 12 to 37. Depths over 32 are generally considered excessive for performance [38]. This indicates that LLMs are not just making superficial changes at the root level but are capable of intervening deeper within the DOM structure. However, they do not consistently reach the deepest possible levels. The variability in Depth Max (e.g., Deepseek R1 at 37 vs. GPT-4.1 at 12) suggests differences in how deeply models traverse and modify complex, nested structures. This overall pattern of intervention across various depths is a fundamental characteristic of LLM DOM manipulation.

**Finding 8: Textual modifications are a primary driver of performance and Visual Stability gains for effective LLMs.** High-performing LLMs, such as **Qwen2.5 32B-Instruct** (PCD:

0.47) and **GPT-4.1** (PCD: 0.48), demonstrate that their success in improving performance metrics is strongly tied to the extent of their textual modifications. A very strong negative correlation exists between **Values Changed Text** and all three latency performance categories: **Initial Load** ( $\rho = -0.74$ ), **Interactivity** ( $\rho = -0.96$ ), and **Runtime** ( $\rho = -0.97$ ), as well as **Network Optimization** ( $\rho = -0.80$ ) and **Resource Optimization** ( $\rho = -0.84$ ). This quantitatively suggests that LLMs making more textual changes are associated with greater performance improvements, indicating beneficial optimizations like minification of inline scripts or styles [20]. Furthermore, a strong negative correlation of  $\rho = -0.90$  between **Values Changed Text** and **Visual Stability** for better-performing models in that audit category indicates these textual modifications also contribute to improved visual stability.

**Answer to RQ2:** Most of the LLMs evaluated primarily used an additive strategy for DOM modifications. The effective LLMs achieve performance gains via extensive textual modifications and at shallower DOM depths. In contrast, GPT-4o-mini shows a unique strategy, removing more elements than it adds, coupled with high positional changes. This is observed alongside its consistent performance regressions in RQ1.

## 6 Discussion

Our evaluation details the strengths and limitations of LLMs in automated web performance resolution, highlighting their effective applications and challenges for DOM manipulation. In the following, we discuss the key implications for research and practice.

### 6.1 Proficiency in Semantic Understanding vs the Pervasive Challenge of Visual Stability

The most consistent finding across all LLMs is their semantic understanding of the DOM (Finding 1). This proficiency underscores the LLMs’ ability to grasp the semantic clarity and structural correctness of web elements, aligning with prior research on automated semantic validation and enhancement [68, 69]. The implication here is profound; LLMs can be helpful when integrated into automated workflows for critical web development tasks that primarily involve semantic markup, alt attributes, ARIA roles, and other accessibility-related improvements, as well as for SEO considerations [18]. Some examples of related LLM changes we identified through manual inspection include adding alt descriptions to image elements [3] and introducing additional meta elements [52].

This can significantly reduce the manual effort and expertise required to maintain compliance with necessary web standards [1, 16]. For developers and organizations, this translates into a powerful tool for proactive maintenance and adherence to best practices, potentially democratizing access to high-quality, accessible web content.

Despite their semantic prowess, a significant and widespread challenge identified in this study is the consistent struggle of most LLMs to maintain or improve **Visual Stability** (Finding 4). A majority of models introduced visual instability. Manual inspection revealed this largely stemmed from seemingly minor insertions or



**Table 4: Summary of Deepdiff Results by LLM Model with Calculated Ratios (including Depth stats)**

Model	EATRR	PCD	Attributes		Elements		Types Changed	Change Depth			Values Changed			
			Added	Removed	Added	Removed		Min	Max	Med	Attr	Tag	Pos	Text
Claude 3.7 Sonnet(R)	0.89	0.42	2	1	90	11	4	1	12	4	10	7	57	24
Claude 3.7 Sonnet	0.88	0.52	3	1	77	11	4	1	12	6	12	7	66	27
Deepseek R1	0.50	0.31	4	7	37	37	7	1	37	4	13	8	37	32
Deepseek V3	0.85	0.53	1	1	106	18	3	1	12	4	10	7	88	31
GPT-4.1	0.89	0.48	2	1	111	14	4	1	12	4	10	8	79	31
GPT-4o-mini	0.44	0.77	2	8	22	28	4	1	12	4	10	8	66	26
Llama3.3 70B	0.67	0.40	3	3	90	45	4	1	24	6	10	8	74	41
o4-mini	0.86	0.39	4	1	133	21	4	1	12	6	14	9	76	29
Qwen2.5 32B-Instruct	0.74	0.47	2	1	70	25	4	1	24	6	10	8	70	45

attribute changes that inadvertently altered element dimensions or flow, e.g., duplicated assets or scripts and prevalent changes in class names tied to CSS styles. This highlights a critical current limitation in LLMs’ hierarchical and spatial understanding of the DOM. While they can semantically understand elements and generate code, they frequently fail to accurately predict the cascading visual effects of their modifications on page layout and rendering.

This limitation poses a substantial barrier to the full automation of web performance resolution, as visual stability is a critical component of performant websites [20]. Future research must, therefore, intensively focus on enhancing LLMs’ spatial reasoning and visual prediction capabilities within the context of DOM manipulation.

## 6.2 Unpacking LLM Modification Strategies: Additive vs. Disruptive Approaches

Our analysis of DOM modification types revealed distinct strategies employed by the LLMs in this study, is primarily additive. Most LLMs predominantly employ an **additive DOM modification strategy** (Finding 5), introducing significantly more elements than they remove. While this approach can facilitate semantic enhancements or performance-critical additions, it carries the inherent risk of contributing to DOM bloat. Manual inspection revealed that a common cause of this bloat was the duplication of already existing SVG paths, which can counteract performance gains in other areas. Furthermore, while LLMs operate across varying DOM depths (Finding 7), some models like Deepseek R1 showed extreme maximum depths of change (up to 37), exceeding the advised maximum depth of 32 to mitigate increased memory usage caused by large DOMs [38]. This indicates that while LLMs are capable of deep interventions, such extreme depth changes, even if related to positional shifts, warrant caution due to potential performance overhead.

In contrast to this additive trend, **GPT-4o-mini exhibited a uniquely disruptive strategy**, characterized by removing more elements than it adds, coupled with the highest PCD (Finding 6). This frequent reordering and shifting of elements, without a corresponding reduction in overall element count, is a likely contributor to its consistent performance regressions observed in RQ1 (Finding 3), especially for user-facing latency and visual stability. Such large-scale positional changes are known to be computationally expensive, often triggering costly browser reflows and repaints, which directly degrade overall user experience [38].

Conversely, the success of high-performing LLMs like *Qwen2.5 32B-Instruct* and *GPT-4.1* in improving performance metrics is strongly

tied to their extensive textual modifications (Finding 8). These models excel when optimizations can be expressed primarily through textual manipulation, often at shallower, more impactful DOM levels. Manual inspection confirmed this, showing these models adding performance-critical attributes (e.g., defer and async for faster script loading times) to link, and script elements often found in the head section of webpages, closer to the DOM root. This strategic placement and modification of existing attributes directly contribute to improved load and runtime performance [38].

## 6.3 Overall Implications for Automated Web Performance Resolution

The findings present LLMs as powerful yet incomplete tools for automated web performance remediation. They excel at *semantic* optimizations (e.g., SEO, accessibility) but remain unreliable for *performance-critical* tasks (e.g., load speed, interactivity), where regressions—especially in visual stability—were frequent. These results highlight the need for careful integration rather than full automation. The key implications are:

- **Hybrid Approaches are Essential:** Given the mixed outcomes, LLM-driven optimization should be paired with automated validation (e.g., Lighthouse) and, when needed, human oversight. CI/CD pipelines must include checks for visual stability and latency to detect and prevent regressions introduced by LLM edits.
- **Targeted LLM Deployment:** LLMs are best applied to well-defined tasks where their strengths are proven, such as semantic optimization. For more sensitive performance areas, careful model selection and rigorous testing are crucial. As shown by differences between high-performing models and *GPT-4o-Mini*, characterizing a model’s typical DOM modification patterns (e.g., EATRR, PCD, depth of edits) can guide safer deployment.
- **Need for Enhanced LLM Capabilities:** Future work should address limitations in hierarchical and spatial reasoning to improve visual stability. Promising directions include architectural changes for better layout awareness and prompt strategies that explicitly account for visual impact.

In conclusion, LLMs hold strong potential to accelerate web optimization but require thoughtful, guarded integration into existing workflows and continued research to close current capability gaps.

## 7 Limitations

**Internal Validity.** Threats to internal validity concern factors that may affect the accuracy of our findings. A key source of bias is the sensitivity of LLM performance to prompt design. We mitigated this by using consistent, well-tested zero-shot prompts across all models (Section 3). Temperature was fixed at 0.0 [22], and single trials were used for cost reasons, which may understate variance. Output and chunk limits matched the smallest context window (GPT-4o-mini) to ensure fairness; future work should explore larger contexts and repeated runs. To prevent context loss when handling large DOM trees, we applied a structured chunking strategy validated through Tree Edit Distance checks [36, 56].

**External Validity.** Threats to external validity concern the generalizability of our findings. These depend on the dataset, LLMs, and audit tools used. We evaluated 15 popular, real-world web-pages [5] that varied widely in use, structure, and complexity, and nine state-of-the-art LLMs differing in size, context limits, and reasoning ability. Although our results represent a snapshot of current model versions, this breadth supports meaningful generalization. To ensure consistency, we used Lighthouse [24] as the sole auditing tool, enabling comparability with prior studies [23, 42, 53]. Our study did not measure direct user experience metrics or production factors such as computational cost or inference latency [4, 65], which remain promising directions for future work.

## 8 Related Work

Recent advancements in LLMs have significantly expanded their applicability to web development, moving beyond traditional natural language processing to encompass a deeper understanding and manipulation of the DOM [30, 32, 33]. We group the related work into three major themes: LLMs for HTML Understanding, LLMs for Web Content Generation and Customization, and LLMs for Web-Based Task Automation and Information Extraction.

### 8.1 LLMs for HTML Understanding

LLMs are increasingly used for HTML understanding, parsing raw HTML for tasks like web-based automation and browser-assisted retrieval [2, 29, 45]. This capability hinges on an LLM’s understanding of HTML’s semantic structure, tag-based syntax, and hierarchical organization (forming the DOM tree). Notably, Gur et al. [29] showed that LLMs pre-trained on natural language can readily transfer to HTML understanding, requiring minimal preprocessing for tasks like semantic classification and description generation. Our study builds on this by assessing how LLMs, using raw HTML, not only comprehend its structure and semantics but also apply this understanding to resolve web performance issues, requiring a deeper, actionable interpretation of HTML elements in context.

### 8.2 LLMs for Web Content Generation and Customization

LLMs have been leveraged to generate and customize web content. Calò and De Russis [10] demonstrated LLMs’ ability to facilitate website creation from natural language descriptions, validating their HTML understanding for this task. Similarly, Li et al. [37] explored LLMs for UI customization, enabling style-related DOM

changes through natural language commands. Our research goes a step further by focusing on web performance optimization. We evaluate how well LLMs can modify the DOM to ensure faster style changes for subsequent UI customizations.

### 8.3 LLMs for Web-Based Task Automation and Information Extraction

LLMs have shown promise in web automation and information extraction. Nass et al. [45] showed LLMs can enhance web element identification in GUI test automation by localizing elements based on contextual awareness. Research on information extraction, such as WebFormer by Wang et al. [64], often focuses on text content but also highlights the importance of leveraging HTML’s structural and layout information for accuracy. Our study expands these applications by evaluating LLMs’ capacity to make actionable DOM modifications to resolve complex, performance-related issues.

## 9 Conclusion

We evaluated nine state-of-the-art LLMs for automated resolution of web performance issues. The models demonstrated universal proficiency in SEO and accessibility optimization, leveraging strong semantic and structural DOM understanding. However, their effectiveness on performance-critical issues (latency, network, resource) was highly variable; most introduced visual instability due to limited spatial reasoning. We observed predominantly additive modification strategies—sometimes leading to bloat—with *GPT-4o-mini* exhibiting uniquely disruptive, high-positional-change behavior. Effective optimizations arose from models capable of targeted, depth-aware edits, underscoring the need for precision and post-hoc validation before deployment. We acknowledge that web performance depends on the interaction of DOM, CSS, and JavaScript. Our DOM-centric scope was a deliberate choice to provide a controlled, reproducible benchmark *without confounding build or server-side factors*. This focus established a foundational baseline for isolating LLM behaviors, while the observed regressions reveal the challenges of automated DOM manipulation. Extending future work to richer CSS/JS contexts is a natural next step toward a more complete understanding of LLM-driven web optimization.

## References

- [1] Accessibility Guidelines Working Group. 2018. Web Content Accessibility Guidelines (WCAG) 2.1. <https://www.w3.org/TR/WCAG21/>. Recommendation of the Web Accessibility Initiative (WAI).
- [2] Aman Ahluwalia and Suhrud Wani. 2024. Leveraging Large Language Models for Web Scraping. *arXiv preprint arXiv:2406.08246* (2024).
- [3] Tahani Alahmadi and Steve Drew. 2018. Evaluation of image accessibility for visually impaired users. *Journal of Accessibility and Design for All* 8, 2 (2018), 125–160.
- [4] Bill Albert and Tom Tullis. 2022. *Measuring the user experience: Collecting, analyzing, and presenting UX metrics*. Morgan Kaufmann.
- [5] Amazon. [n.d.]. Amazon Alexa top sites. <https://www.alexa.com/topsites>
- [6] Anonymous. 2025. *Evaluating the Use of LLMs for Automated DOM-Level Resolution of Web Performance Issues*. <https://doi.org/10.5281/zenodo.14785703>
- [7] Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. 2014. Dompletion: DOM-aware JavaScript code completion. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (Vasteras, Sweden) (ASE ’14)*. Association for Computing Machinery, New York, NY, USA, 43–54. <https://doi.org/10.1145/2642937.2642981>
- [8] Marcus Basalla, Johannes Schneider, Martin Luksik, Roope Jaakonmäki, and Jan Vom Brocke. 2021. On latency of e-commerce platforms. *Journal of Organizational Computing and Electronic Commerce* 31, 1 (2021), 1–17.

- [9] Enrico Bocchi, Luca De Cicco, and Dario Rossi. 2016. Measuring the quality of experience of web users. *ACM SIGCOMM Computer Communication Review* 46, 4 (2016), 8–13.
- [10] Tommaso Calò and Luigi De Russis. 2023. Leveraging large language models for end-user website generation. In *International Symposium on End User Development*. Springer, 52–61.
- [11] Tommaso Calò and Luigi De Russis. 2023. Leveraging Large Language Models for End-User Website Generation. In *End-User Development*, Lucio Davide Spano, Albrecht Schmidt, Carmen Santoro, and Simone Stumpf (Eds.). Springer Nature Switzerland, Cham, 52–61.
- [12] Kwame Chan-Jong-Chu, Tanjina Islam, Miguel Morales Exposito, Sanjay Sheombar, Christian Valladares, Olivier Philippot, Eoin Martino Grua, and Ivano Malavolta. 2020. Investigating the correlation between performance scores and energy consumption of mobile web apps. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*. 190–199.
- [13] Dariusz Chęć and Ziemowit Nowak. 2019. The performance analysis of web applications based on virtual DOM and reactive user interfaces. In *Engineering Software Systems: Research and Praxis*. Springer, 119–134.
- [14] Google Chrome. 2024. Lighthouse - Understanding the Results. <https://github.com/GoogleChrome/lighthouse/blob/main/docs/understanding-results.md>
- [15] World Wide Web Consortium et al. 2004. Document object model (DOM) level 3 core specification. (2004).
- [16] MDN contributors. 2025. <meta>: The metadata element. Mozilla Foundation. <https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/meta> MDN Web Docs.
- [17] Anita Crescenzi, Diane Kelly, and Leif Azzopardi. 2016. Impacts of time constraints and system delays on user experience. In *Proceedings of the 2016 acm on conference on human information interaction and retrieval*. 141–150.
- [18] Giovanni Delnevo, Manuel Andruccioli, and Silvia Mirri. 2024. On the Interaction with Large Language Models for Web Accessibility: Implications and Challenges. In *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*. IEEE, 1–6.
- [19] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems* 36 (2024).
- [20] Google Developers. [n. d.]. Understanding Core Web Vitals and Google search results. <https://developers.google.com/search/docs/appearance/core-web-vitals>
- [21] Raimundo N.V. Diniz-Junior, Caio César L. Figueiredo, Gilson De S.Russo, Marcos Roberto G. Bahiense-Junior, Mateus V.L. Arbex, Lanier M. Dos Santos, Raimundo F. Da Rocha, Renan R. Bezerra, and Felipe T. Giuntini. 2022. Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue. In *2022 XLIII Latin American Computer Conference (CLEI)*. 1–9. <https://doi.org/10.1109/CLEI56649.2022.9959901>
- [22] Jean-Baptiste Döderlein, Nguessan Hermann Kouadio, Mathieu Acher, Djamel Ed-dine Khelladi, and Benoit Combemale. 2025. Piloting Copilot, Codex, and StarCoder2: Hot temperature, cold prompts, or black magic? *Journal of Systems and Software* (2025), 112562.
- [23] Muhammad Arif Faizin, Muhammad Nevin, and Umi Laili Yuhana. 2024. Indonesia E-Government Website Performance and Accessibility Evaluation using Automated Tool Lighthouse. In *2024 2nd International Conference on Software Engineering and Information Technology (ICoSEIT)*. 210–215. <https://doi.org/10.1109/ICoSEIT60086.2024.10497521>
- [24] Chrome for Developers. 2016. Lighthouse Overview. <https://developer.chrome.com/docs/lighthouse/overview> Accessed: 2024-09-04.
- [25] Google for Developers. 2016. PageSpeed Insights. <https://pagespeed.web.dev> Accessed: 2024-09-04.
- [26] Gary L Geissler, George M Zinkhan, and Richard T Watson. 2006. The influence of home page complexity on consumer attention, attitudes, and purchase intent. *Journal of Advertising* 35, 2 (2006), 69–80.
- [27] Andreas B Gizas and Sotiris P Christodoulou. 2015. Performance-optimized pages' architecture, navigation and images techniques for JQuery mobile sites. In *Proceedings of the 19th Panhellenic Conference on Informatics*. 371–377.
- [28] Utkarsh Goel, Stephen Ludin, and Moritz Steiner. 2020. Web performance with android's battery-saver mode. *arXiv preprint arXiv:2003.06477* (2020).
- [29] Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. 2022. Understanding HTML with Large Language Models. *ArXiv abs/2210.03945* (2022). <https://api.semanticscholar.org/CorpusID:252780086>
- [30] Muhammad Usman Hadi, Qasem Al Tashi, Abbas Shah, Rizwan Qureshi, Amgad Muneer, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, et al. 2024. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea Preprints* (2024).
- [31] Tjaša Heričko, Boštjan Šumak, and Saša Brdnik. 2021. Towards representative web performance measurements with google lighthouse. In *Proceedings of the 2021 7th Student Computer Science Research Conference*. 39.
- [32] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620* (2023).
- [33] Vassilika D. Kirova, Cyril S. Ku, Joseph R. Laracy, and Thomas J. Marlowe. 2024. Software Engineering Education Must Adapt and Evolve for an LLM Environment. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) (SIGCSE 2024). Association for Computing Machinery, New York, NY, USA, 666–672. <https://doi.org/10.1145/3626252.3630927>
- [34] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.
- [35] Simo Kuparinen. 2023. Improving web performance by optimizing cascading style sheets (CSS): literature review and empirical findings. *Helsinki University Library* 1, 2 (2023).
- [36] Manuel Leithner and Dimitris E Simos. 2018. DOMdiff: Identification and classification of inter-DOM modifications. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, 262–269.
- [37] Amanda Li, Jason Wu, and Jeffrey P Bigham. 2023. Using LLMs to Customize the UI of Webpages. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) (UIST '23 Adjunct). Association for Computing Machinery, New York, NY, USA, Article 45, 3 pages. <https://doi.org/10.1145/3586182.3616671>
- [38] Google Developers (Lighthouse). [n. d.]. Avoid an excessive DOM size. <https://developer.chrome.com/docs/lighthouse/performance/dom-size>
- [39] Juan-Miguel López-Gil and Juanan Pereira. 2024. Turning manual web accessibility success criteria into automatic: an LLM-based approach. *Universal Access in the Information Society* (2024), 1–16.
- [40] Şevval Seray Macakoğlu and Serhat Peker. 2022. Web accessibility performance analysis using web content accessibility guidelines and automated tools: a systematic literature review. In *2022 international congress on human-computer interaction, optimization and robotic applications (hora)*. IEEE, 1–8.
- [41] Sonai Mahajan, Negarsadat Abolhassani, Phil McMinn, and William GJ Halfond. 2018. Automated repair of mobile friendly problems in web pages. In *Proceedings of the 40th international conference on software engineering*. 140–150.
- [42] Thomas McGill, Oluwaseun Bamgboye, Xiaodong Liu, and Chathuranga Sampath Kalutharage. 2023. Towards Improving Accessibility of Web Auditing with Google Lighthouse. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. 1594–1599. <https://doi.org/10.1109/COMPSAC57700.2023.00246>
- [43] Thomas McGill, Oluwaseun Bamgboye, Xiaodong Liu, and Chathuranga Sampath Kalutharage. 2023. Towards Improving Accessibility of Web Auditing with Google Lighthouse. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 1594–1599.
- [44] Dirk Merkel et al. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux j* 239, 2 (2014), 2.
- [45] Michel Nass, Emil Alégroth, and Robert Feldt. 2023. Improving web element localization by using a large language model. *Software Testing, Verification and Reliability* (2023), e1893.
- [46] Froin S Ocariza Jr, Karthik Pattabiraman, and Benjamin Zorn. 2011. JavaScript errors in the wild: An empirical study. In *2011 IEEE 22nd International Symposium on Software Reliability Engineering*. IEEE, 100–109.
- [47] OpenAI. 2024. GPT-4o. <https://platform.openai.com/docs/models/gpt-4o>
- [48] OpenAI. 2024. tiktoken. <https://github.com/openai/tiktoken> Accessed: 2024-10-14.
- [49] Zizi Papacharissi. 2002. The self online: The utility of personal home pages. *Journal of Broadcasting & Electronic Media* 46, 3 (2002), 346–368.
- [50] Morgan Persson. 2020. JavaScript DOM Manipulation Performance: Comparing Vanilla JavaScript and Leading JavaScript Front-end Frameworks.
- [51] PyPI. [n. d.]. Python BeautifulSoup library. <https://pypi.org/project/beautifulsoup4>
- [52] Konstantinos I Roumeliotis and Nikolaos D Tselikas. 2022. An effective SEO techniques and technologies guide-map. *Journal of web engineering* 21, 5 (2022), 1603–1649.
- [53] Darius Saif, Chung-Horng Lung, and Ashraf Matrawy. 2021. An early benchmark of quality of experience between HTTP/2 and HTTP/3 using lighthouse. In *ICC 2021-IEEE international conference on communications*. IEEE, 1–6.
- [54] Hamza Salem, Hadi Salloum, Osama Orabi, Kamil Sabbagh, and Manuel Mazara. 2025. Enhancing News Articles: Automatic SEO Linked Data Injection for Semantic Web Integration. *Applied Sciences* 15, 3 (2025), 1262.
- [55] Justin Scherer. 2020. *Hands-on JavaScript High Performance: Build Faster Web Apps Using Node.js, Svelte.js, and WebAssembly*. Packt Publishing Ltd.
- [56] Kilho Shin, Taichi Ishikawa, Yu-Lu Liu, and David Lawrence Shepard. 2021. Learning DOM trees of web pages by subpath kernel and detecting fake e-commerce sites. *Machine Learning and Knowledge Extraction* 3, 1 (2021), 95–122.
- [57] Shailesh Kumar Shivakumar. 2015. 3 - Optimizing Performance of Enterprise Web Application. In *Architecting High Performing, Scalable and Available Enterprise Web Applications*, Shailesh Kumar Shivakumar (Ed.). Morgan Kaufmann, Boston,

- 101–141. <https://doi.org/10.1016/B978-0-12-802258-0.00003-2>
- [58] Shailesh Kumar Shivakumar. 2020. Modern web performance optimization. *Methods, Tools, and Patterns to Speed Up Digital Platforms* (2020).
  - [59] Surendra N Singh, Nikunj Dalal, and Nancy Spears. 2005. Understanding web home page perception. *European Journal of Information Systems* 14, 3 (2005), 288–302.
  - [60] BM Subraya and SV Subrahmanya. 2000. Object driven performance testing of Web applications. In *Proceedings First Asia-Pacific Conference on Quality Software*. IEEE, 17–26.
  - [61] Rebeka Tóth, Tamas Bisztray, and László Erdodi. 2024. LLMs in Web-Development: Evaluating LLM-Generated PHP code unveiling vulnerabilities and limitations. *arXiv preprint arXiv:2404.14459* (2024).
  - [62] Jasper van Riet, Ivano Malavolta, and Taher A. Ghaleb. 2023. Optimize along the way: An industrial case study on web performance. *Journal of Systems and Software* 198 (2023), 111593. <https://doi.org/10.1016/j.jss.2022.111593>
  - [63] Hernán Ceferino Vázquez, Alexandre Bergel, Santiago Vidal, JA Díaz Pace, and Claudia Marcos. 2019. Slimming javascript applications: An approach for removing unused functions from javascript libraries. *Information and software technology* 107 (2019), 18–29.
  - [64] Qifan Wang, Yi Fang, Anirudh Ravula, Fuli Feng, Xiaojun Quan, and Dongfang Liu. 2022. Webformer: The web-page transformer for structure information extraction. In *Proceedings of the ACM Web Conference 2022*. 3124–3133.
  - [65] Nikolas Wehner, Monisha Amir, Michael Seufert, Raimund Schatz, and Tobias Hoßfeld. 2022. A vital improvement? Relating Google’s core web vitals to actual web QoE. In *2022 14th international conference on quality of multimedia experience (QoMEX)*. IEEE, 1–6.
  - [66] Lauren Wood, Arnaud Le Hors, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Gavin Nicol, Jonathan Robie, Robert Sutor, et al. 1998. Document object model (dom) level 1 specification. *W3C recommendation* 1 (1998).
  - [67] Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. 2000. Document object model (DOM) level 3 core specification.
  - [68] Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and Hong Lin. 2023. Ai-generated content (aigc): A survey. *arXiv preprint arXiv:2304.06632* (2023).
  - [69] Shaomei Wu, Jeffrey Wieland, Omid Farivar, and Julie Schiller. 2017. Automatic alt-text: Computer-generated image descriptions for blind users on a social network service. In *proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*. 1180–1192.
  - [70] Zepworks. [n. d.]. Deepdiff. <https://zepworks.com/deepdiff/current/diff.html>