Evaluating the Use of LLMs for Automated Resolution of Web Performance Issues

Gideon Peters

A Thesis

In the Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Master of Applied Science (Software Engineering)

at Concordia University

Montréal, Québec, Canada

July 2025

Abstract

Evaluating the Use of LLMs for Automated Resolution of Web Performance Issues

Gideon Peters

Concordia University, 2025

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 study highlights LLMs' clear feasibility in web performance engineering workflows, particularly for semantic concerns. 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.

Acknowledgments

Completing my MASc. degree has been an incredibly rewarding journey, one shaped by the guidance, collaboration, and support of many remarkable people.

First and foremost, I am deeply grateful to my supervisor, Dr. Emad Shihab, for his constant encouragement, patience, and insightful feedback. His mentorship has been invaluable in helping me grow both as a researcher and as a person.

I would also like to thank my postdoc, Dr. SayedHassan Khatoonabadi, whose exceptional collaboration and thoughtful guidance have been a steady source of inspiration throughout this work.

My sincere appreciation goes to my colleagues in the DAS Lab. Your camaraderie and generosity made the lab a place where ideas could flourish. A special thank you to Mayra, Samuel, Caren, Alor, Jasmine, and Chaima for your advice, encouragement, and all the thoughtful conversations that made this journey brighter.

I am also grateful to the CREATE program for supporting my research and professional development, and to Lori, the program coordinator, for her kind assistance and guidance every step of the way.

Finally, I owe my deepest gratitude to my parents, siblings and friends for their unwavering love and belief in me. Your support has been the foundation that carried me through every challenge.

Dedication

To my beloved parents, Dr. Omale Peters and Major. Grace Peters, for their sacrifices, unwavering support and guidance. And to my siblings, Favour and Isaiah Peters, for pushing me to defy boundaries every time.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Listings

Chapter 1

Introduction and Research Statement

1.1 Introduction

Web applications have become one of the primary ways users consume content on the internet [1]. Therefore, the importance of performant web applications cannot be overemphasized [2]. A webpage's performance is a core nonfunctional requirement, as it impacts the overall user experience, engagement, and conversion ratios [3, 4]. 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 [5]. 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) [6, 7].

This thesis focuses on the client-side, specifically the Document Object Model (DOM), which is central to how browsers interpret, render, and interact with webpages [8, 9]. 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 [6, 7, 10]. Optimizing the DOM is challenging [11], 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 [12]. However, the growing complexity of web applications demands more sophisticated solutions [1].

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 [13, 14, 15]. Trained on massive corpora, including HTML documents from public repositories, LLMs are uniquely positioned to tackle challenges in web development [16, 17]. Their applications extend beyond code generation to include tasks like web security [18], automated testing [19], and accessibility improvements [20]. 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 prevalence of the initial audits before and after modification. In summary, we aim to answer the following research questions:

RQ1: How effective are LLMs for resolving web performance issues in the DOM? Performance optimization can be tedious, requiring web developers to run performance tests, and implement required fixes [21, 22]. 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 web performance issue resolution? 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.

This thesis highlights various insights for web developers, LLM providers, and the web development research community on the task of automating web performance issues resolution. By focusing on the DOM—a language-agnostic structure that all frameworks must adhere to for browser rendering, we address performance bottlenecks at their core, independent of specific languages or frameworks. This approach enables

broader applicability and ensures solutions can be integrated into the CI/CD pipeline before production, improving both user experience and developer efficiency by reducing iterative optimization cycles, thus saving time for other meaningful tasks [19].

1.2 Research Statement

Motivated by the complexity and iterative nature of web performance optimization—and the critical importance of performance for user experience and accessibility—the goal of this MASc thesis is to investigate whether large language models (LLMs) can reliably automate the resolution of performance issues in real-world webpages. We state our research statement as follows:

> Web performance optimization is a time-consuming process requiring specialized expertise to diagnose issues and implement precise modifications in complex DOM structures. This thesis systematically evaluates the ability of LLMs for the automated resolution of web performance issues. Specifically, it examines whether LLMs can resolve these issues measured by standardized audits across diverse webpages, characterizes the types and patterns of their modifications, and assesses their reliability for integration into automated optimization workflows.

1.3 Thesis Overview

In this section, we provide an overview of the work presented in this thesis and highlight the main themes of each chapter.

Chapter 2: Background and Related Work

This chapter introduces key concepts and technologies foundational to this work. We begin by explaining the Document Object Model (DOM), its hierarchical structure, and how different node types—such as tags, text, scripts, and stylesheets—contribute to the complexity and performance characteristics of webpages. We discuss how DOM depth and size impact rendering efficiency and responsiveness. We then describe the role of performance audits, focusing on Lighthouse as our primary evaluation tool. Lighthouse audits produce structured reports with scores, descriptions, and actionable details on performance, accessibility, and

SEO. Finally, we survey prior research applying large language models to HTML understanding, web content generation, and automated optimization, identifying gaps that motivate our research.

Chapter 3: Methodology

To rigorously evaluate whether LLMs can automate the resolution of web performance issues, we develop a comprehensive study design. This chapter details the construction of a dataset comprising 15 production webpages selected from the Alexa Top 500 list, reflecting diverse structures and performance profiles. We describe our extraction of complete DOM trees, their division into manageable chunks to accommodate LLM output constraints, and our validation process ensuring reassembly fidelity using Tree Edit Distance. We present the criteria used to select nine diverse LLMs varying in architecture, model size, and token limits. Additionally, we outline our Lighthouse configuration, including filtering of audits to focus on actionable performance issues across seven defined categories. Finally, we explain our use of the Audit Incidence Ratio (AIR) to benchmark improvements and regressions resulting from model-generated modifications.

Chapter 4: Results

This chapter presents our quantitative assessment of LLM performance across multiple audit categories. We find that all models demonstrate strong semantic understanding, achieving a 100% reduction in SEO and accessibility-related issues. However, performance outcomes vary significantly across other categories. High-performing models, such as Qwen2.5-32B-Instruct and GPT-4.1, achieve substantial improvements in latency and resource usage, while GPT-4o-Mini consistently introduces regressions in runtime and visual stability. These results illustrate both the promise and the variability of LLMs in addressing complex performance bottlenecks.

Additionally, to understand why different models produce divergent outcomes, this chapter analyzes the specific DOM modifications each LLM performs. We categorize changes using structured diffs, reporting frequencies of additions, removals, type changes, and positional reordering. We define metrics such as the Element Addition-to-Removal Ratio (EATRR) and Positional Change Dominance (PCD) to quantify modification strategies. Our findings show that most models adopt an additive strategy, introducing new elements to improve performance, while GPT-4o-Mini stands out for its disruptive reordering of existing nodes. High-performing models concentrate modifications on textual changes and shallow DOM levels,

correlating with greater improvements in load times and resource efficiency.

Chapter 5: Discussion

This chapter discusses the implications of our findings for both research and practice. We first highlight that LLMs exhibit strong capabilities in semantic enhancements, particularly for SEO and accessibility improvements, suggesting their immediate utility for automated compliance tasks. However, we identify pervasive challenges in maintaining visual stability, as many models inadvertently introduce layout shifts through duplications or aggressive reordering. Our analysis shows that additive modification strategies can contribute to DOM bloat, especially when duplicating assets like SVG paths, while disruptive strategies lead to costly browser reflows and degraded performance. The most effective models rely on targeted textual modifications and shallow-depth changes, such as adding `defer` and `async` attributes to script elements. These findings underscore the need for hybrid approaches combining LLM-driven suggestions with automated validation in CI/CD pipelines. We recommend that practitioners deploy LLMs selectively, focusing on scenarios where their strengths are well established, and complement them with safeguards for more sensitive performance areas. Finally, we outline opportunities for future work to enhance spatial reasoning in LLMs and refine prompt engineering strategies to better control the scope of modifications.

Chapter 6: Threats to Validity

We outline limitations inherent to this thesis. Internally, model performance is sensitive to prompt design, and our chunking approach—while necessary to fit model constraints—may result in some loss of global context. We mitigated these risks through standardized prompts and rigorous structural validation of reassembled DOMs. Externally, while our dataset spans diverse popular webpages and our evaluation includes nine state-of-the-art models, results may not generalize to all types of web applications or future model releases. Additionally, reliance on Lighthouse as the primary audit tool does not capture all user-experience factors, suggesting the need for complementary assessments in future research.

Chapter 7: Conclusion and Future Work

Finally, this chapter summarizes the contributions of the thesis and suggests directions for continued exploration. We conclude that LLMs show significant promise for automating aspects of web performance

optimization but require careful integration, targeted deployment, and robust validation to avoid unintended regressions. Future work should focus on developing hybrid systems combining LLM reasoning with deterministic checks, improving modelsâĂŹ hierarchical and spatial understanding of the DOM, and expanding evaluation frameworks to incorporate direct user experience metrics and deployment considerations.

1.4 Thesis Contributions

The main contributions of this thesis are as follows:

- 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 this thesis and facilitate future research on this topic, we publicly share our scripts and dataset online [23].

1.5 Related Publications

The work presented in this thesis has been submitted to the following venue for review:

- **Gideon Peters**, SayedHassan Khatoonabadi, and Emad Shihab. Evaluating the Use of LLMs for Automated Resolution of Web Performance Issues. *Submitted to International Conference on Software Engineering 2026 (ICSE'26)*.

1.6 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 provides the necessary background for this thesis as well as a review of related works relevant to this thesis. Then, Chapter 3 present our methodology, detailing our quantitative evaluation of how effectively different LLMs resolve web performance issues across multiple audit categories, and examining the nature and patterns of DOM modifications performed by these models to understand the underlying factors contributing to their observed performance outcomes. Chapter 4 highlights our results and corresponding findings. Chapter 5 discusses the implications of this work. Chapter 6 outlines the limitations of the thesis. Chapter 7 summarizes the thesis and discusses the key directions for future work.

Chapter 2

Background and Related Work

In this chapter, we provide an overview of the concepts and studies relevant to this thesis. First, we introduce the fundamental principles and technologies of the DOM and web performance audits. Then, we review related work on applying large language models (LLMs) to HTML understanding, content generation, and automated optimization. At the end of each section, we explain how our thesis contributes to the body of knowledge.

2.1 Document Object Model (DOM)

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 [24]. 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 2.1 illustrates a DOM tree, showing its HTML code representation on the left and its hierarchical structure on the right.

Key characteristics of a DOM tree include:

- **Root Node:** The tree begins with the `<html>` element as its root, which has `<head>` and `<body>` as children.
- **Parent-Child Relationships:** Elements are hierarchically organized, with parent nodes containing

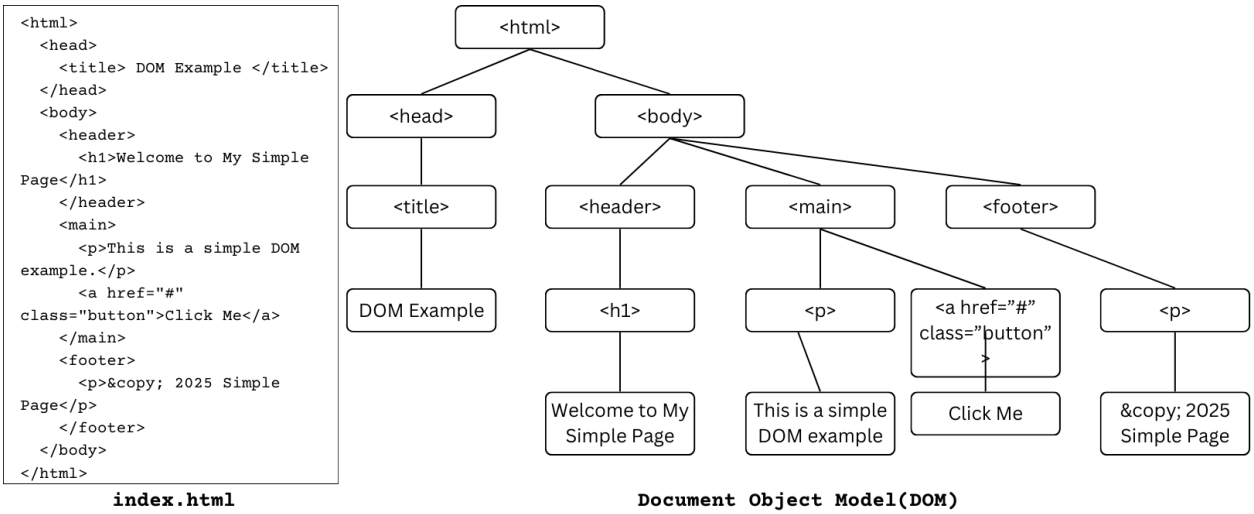

Figure 2.1: Example structure of a DOM tree

child nodes (e.g. <body> contains <header>, <main>, and <footer>).

- **Sibling Relationships:** Nodes at the same hierarchical level are siblings (e.g. and <a> within <main>).

Furthermore, there are various element types present in DOM trees. These element types include:

- **Text:** Represents textual content within HTML elements (e.g. "Hello, world!" in Hello, world!).
- **Comment:** HTML comments, enclosed within <!- ->, provide additional context or documentation but are not rendered in the browser.
- **Tag:** These are the core elements of the DOM structure, representing HTML tags like , <a>, or <table>. Each tag may have attributes (e.g., Link is an element with a href attribute that has the value example.com).
- **Script:** Embeds or references JavaScript (e.g. <script src="app.js"> </script>).
- **Stylesheet:** Represents CSS rules for styling (e.g., inline <style>p { color: red; }</style> or externally <link rel="stylesheet" href="styles.css">).

These element types are combined in various ways across webpages [25]. In performance optimization, DOM tree size and complexity significantly impact load time and responsiveness, as larger and more deeply

nested trees require greater computational resources [26, 27].

Our thesis leverages this understanding of the DOM to assess how LLMs can modify these structures to address performance bottlenecks.

2.2 Performance Audits and Lighthouse

Performance audits systematically evaluate webpages to assess their performance, identify bottlenecks, and recommend improvements [28]. These audits aim to ensure sites meet performance goals critical to user experience, search rankings, and business outcomes [21].

Our work utilizes `Lighthouse`, an open-source Chromium-based tool developed by Google for the measurement and improvement of webpage performance [29, 30, 31]. We selected `Lighthouse` due to its extensibility and widespread community adoption [21, 2, 32]. 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) [33, 2, 21]. Each audit highlights an issue or suggestion based on standard practices and is keyed by its name. An example audit structure is shown in Listing 2.1, with the following properties:

- **id**: A unique identifier identical to the audit key.
- **title**: A brief summary of the audit's purpose.
- **description**: A detailed explanation of what the audit assesses and its significance.
- **score**: A numeric or categorical value indicating the audit's result (e.g. `pass`, `fail`, or `needing improvement`).
- **scoreDisplayMode**: Denotes how the score is interpreted (e.g. `informative`, `notApplicable`, `manual`, or `error`). We exclude audits with `notApplicable`, `manual`, and `informative`, or `binary` scores of 1 as it indicates as a pass and these audits do not require any resolution in the DOM [33] unlike a binary score of 0 which indicates a needed resolution.
- **displayValue**: A contextual measurement supplementing the score (e.g., "2.5s" for loading time).

- **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. Audits with these location parameters are termed *location-specific*, while those without are *location-non-specific*.

```
{
  "is-on-https": {
      "id": "is-on-https",
      "title": "Does not use HTTPS",
      "description": "All sites should be ...",
      "score": 0,
      "scoreDisplayMode": "binary",
      "displayValue": "1 insecure request found",
      "details": {
        "type": "table",
        "headings": [{
            "key": "url",
            "valueType": "url",
            "label": "Insecure URL"
        }],
        "items": [{
            "url": "http://ajax.googleapis...."
        }]
      }
    },
  }TF
}
```

Listing 2.1: Example structure of a single Lighthouse performance audit

This thesis uses Lighthouse audits both as a benchmark to measure improvements and as a structured reference for guiding LLM modifications.

2.3 LLMs for HTML Understanding

LLMs are increasingly used for HTML understanding, parsing raw HTML for tasks like web-based automation and browser-assisted retrieval [34, 35, 19]. This capability hinges on an LLM's understanding of HTML's semantic structure, tag-based syntax, and hierarchical organization (forming the DOM tree). Beyond basic comprehension, LLMs have demonstrated proficiency in diverse applications, including information extraction from web pages (e.g., identifying product details or contact information) [36, 37], content summarization [38], and even generating or correcting HTML snippets based on natural language prompts [39]. Their ability to process and interpret the complex, nested structure of HTML enables them to perform tasks that previously required specialized parsers or human intervention.

Notably, Gur et al. [34] 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. This transferability underscores the generalizable pattern recognition capabilities of LLMs across different structured data formats. However, challenges persist, particularly concerning the scalability of processing very large and complex HTML documents within typical LLM context windows, and their inherent limitation in directly interpreting visually dynamic content rendered by JavaScript. While LLMs excel at understanding the static HTML structure, integrating real-time visual rendering information remains an area of ongoing research.

This thesis 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. This requires a deeper, actionable interpretation of HTML elements in context, moving beyond mere classification to resolving performance bottlenecks like render-blocking resources or inefficient asset loading strategies. For instance, an LLM might analyze the attributes of an `<img>` tag and its surrounding context to suggest optimal image formats or lazy loading, or identify superfluous `<script>` tags that negatively impact page load times. This level of interpretation demands a nuanced understanding of how individual HTML elements contribute to the overall web rendering process and user experience.

While this prior work predominantly focuses on classification, comprehension, and general HTML manipulation, our thesis extends this line by examining whether LLMs can translate their HTML understanding into targeted, actionable performance optimizations, addressing a critical gap in current web development practices.

2.4 LLMs for Web Customization

LLMs have emerged as powerful tools for customizing web content, extending their utility beyond mere content generation to active manipulation of web interfaces. This capability leverages their deep understanding of natural language combined with their ability to interpret and generate structured code like HTML, CSS, and JavaScript.

Prior work has demonstrated several key applications in this domain. Calò and De Russis [39] pioneered the use of LLMs to facilitate entire website creation directly from natural language descriptions, showcasing their robust HTML comprehension by accurately translating high-level design intents into functional web pages. This indicated LLMs' capacity to not only understand HTML's structure but also to synthesize it coherently. Similarly, Li et al. [40] explored the application of LLMs for on-the-fly User Interface (UI) customization, enabling style-related Document Object Model (DOM) changes through intuitive natural language commands. Their work highlighted the potential for LLMs to democratize web design, allowing users without technical expertise to modify visual elements like colors, fonts, and layouts. Beyond visual styles, LLMs are also being explored for more intricate web content personalization, tailoring information or recommendations based on user profiles, and even for improving web accessibility by generating modifications that comply with standards like WCAG [20].

However, a significant challenge in LLM-driven web customization lies in ensuring that the generated modifications are not just syntactically valid or aesthetically pleasing, but also *performant*. While LLMs can readily produce DOM changes, the subtle interplay of these changes with browser rendering engines, network conditions, and user device capabilities often goes unaddressed. For instance, an LLM might suggest adding a complex shadow effect to many elements, which, while visually appealing, could introduce significant reflows and repaints, thereby degrading user experience. The ability to identify render-blocking resources, optimize image loading, or streamline CSS delivery from raw HTML requires an understanding of web performance best practices that extends beyond typical language or code generation tasks.

This thesis goes a step further by focusing specifically on web performance optimization within the context of customization. We evaluate how well LLMs can modify the DOM to ensure faster style changes for subsequent UI customizations, or more broadly, to improve overall web performance. This demands a critical shift from merely producing desired outputs to generating *computationally efficient* ones. It involves training or prompting LLMs to understand the performance implications of various HTML and CSS constructs, such

as the impact of inline styles versus external stylesheets, the efficiency of different image formats and loading attributes, or the cascading effects of DOM manipulations on rendering pipelines.

We build on this foundation by evaluating whether LLMs can produce not just syntactically valid but performance-improving DOM changes, aiming to bridge the gap between AI-driven web customization and robust web performance engineering.

2.5 LLMs for Web-Based Automation and Information Extraction

The application of LLMs has significantly advanced the fields of web-based automation and information extraction. These domains heavily rely on an the models' ability to accurately understand and interact with web interfaces, often represented by the DOM.

In web automation, LLMs have proven instrumental in enhancing the robustness and flexibility of various tasks, from automated testing to repetitive data entry. Nass et al. [19] notably demonstrated how LLMs can drastically improve web element identification within Graphical User Interface (GUI) test automation. By leveraging their deep contextual awareness, LLMs can localize web elements not just by their explicit IDs or XPaths, but by understanding their semantic role and relationship to surrounding elements within the HTML structure, even in the presence of dynamic content or minor structural changes. This capability moves beyond brittle, hard-coded selectors to more intelligent, natural language-driven element targeting, paving the way for more resilient automation scripts [41]. Other work in this area includes LLMs generating browser automation scripts from natural language descriptions of desired actions, simplifying complex web workflows for non-programmers [39].

Simultaneously, LLMs have made substantial inroads into information extraction from web pages. While traditional methods often relied on rule-based systems or supervised machine learning with extensive feature engineering, LLMs can directly process raw HTML. Research like WebFormer by Wang et al. [42] highlights the critical importance of not only analyzing text content but also extensively leveraging HTML's structural and layout information for accurate data extraction. LLMs can discern intricate relationships between elements, understand table structures, and extract specific entities (e.g., product details, event times, addresses) even from unstructured or semi-structured web content [36, 37]. This allows for more adaptable and less labor-intensive information retrieval across diverse websites. However, most of these applications typically involve a read-only interaction with the DOM, focusing on understanding or identifying rather than actively

modifying the underlying structure for specific goals beyond extraction.

This thesis expands these applications by evaluating LLMs' capacity to make actionable DOM modifications, specifically to resolve complex, performance-related issues. Unlike existing work that uses LLMs for element identification or data retrieval, our approach pushes the boundary towards prescriptive changes. This requires the LLM to not just understand the HTML, but to resolve performance issues from an audit by implementing concrete structural or attribute changes within the DOM that directly address those issues. This involves a much deeper level of actionable interpretation and problem-solving.

Our thesis extends these capabilities by systematically investigating whether LLMs can autonomously modify DOM structures to resolve performance issues identified by audits, bridging the gap between passive HTML understanding and active web optimization.

2.6 Chapter Summary

This chapter introduced the core principles of DOM representation, performance auditing with Lighthouse, and prior work showing that LLMs can parse, generate, and manipulate web content. However, while these studies validate the feasibility of LLMs for HTML comprehension and generation, their ability to perform precise, performance-targeted DOM modifications remains unexplored. To address this gap, this thesis systematically evaluates whether LLMs can improve real-world webpage performance across multiple metrics while maintaining structural integrity.

Chapter 3

Methodology

In this chapter, we present the study design of our thesis, including the selection of real-world webpages, our methodology for conducting performance audits and LLM modifications, and the evaluation metrics used.

3.1 Study Design

This section describes our dataset, the performance audits, the environment configurations, considerations for our LLM selection, the chunking strategy, and the evaluation metric used in this thesis.

3.1.1 Dataset

To conduct our work, we first select 15 real-world webpages at random from the Alexa Top 500 list [43], which features top-ranked webpages on the web. We chose this list due to its popularity and prior use in research [44, 45, 46]. Each webpage selected is a homepage, the main entry point for users. Since homepages typically receive the highest traffic [47, 48, 49], optimizing their performance is particularly relevant to this thesis.

Our dataset comprises webpages from four different categories: *Shopping webpages (4)*, *Professional webpages (2)*, *Social webpages (6)*, and *Entertainment webpages (3)*. Table 3.1 details these selections sorted in order of their Alexa ranking, alongside their Lighthouse Scores(LHS). We also show the number of location-specific audits for each site (Audits w/ Location) and location-non-specific audits (Audits w/o Location). Additionally, it highlights the total number of chunks each webpage was divided into for our experiments. More details on the chunking strategy used are provided later in Chapter 3.

Table 3.1: Webpages and original Lighthouse performance summary

Webpage	**Category**	**Ranking**	**Ahrefs**	**# Chunks**	**Audits w/o Location**	**Audits w/ Location**	**Total Audits**	**LHS (%)**
Youtube	Entertainment	2	2	3	20	9	29	30
Facebook	Social	5	4	2	13	5	18	90
X(ex. Twitter)	Social	7	17	2	13	5	18	42
Linkedin	Professional	12	35	4	14	12	26	78
Reddit	Social	13	5	9	18	9	27	32
Github	Professional	27	99	9	13	12	25	47
Aliexpress	Shopping	40	24	7	25	14	39	14
Pinterest	Social	47	4	2	15	13	28	26
Ebay	Shopping	49	93	17	19	16	35	12
Netflix	Entertainment	54	27	2	14	11	25	34
Quora	Social	58	38	4	12	9	21	54
Twitch	Entertainment	84	170	2	16	10	26	37
Medium	Social	132	33	2	12	8	20	75
Walmart	Shopping	171	89	3	15	10	25	49
Airbnb	Shopping	277	12	10	15	6	21	51

Table 3.2 presents descriptive statistics for the webpages in our dataset. **DOM Tree Depth**, the maximum depth of nested HTML elements, ranging from 4 to 32, indicates diverse structural complexity. The **number of chunks** (# chunks), varying from 2 to 17, reflects 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 3.2: 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 3.1 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` [50].

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 all original chunks are processed, 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

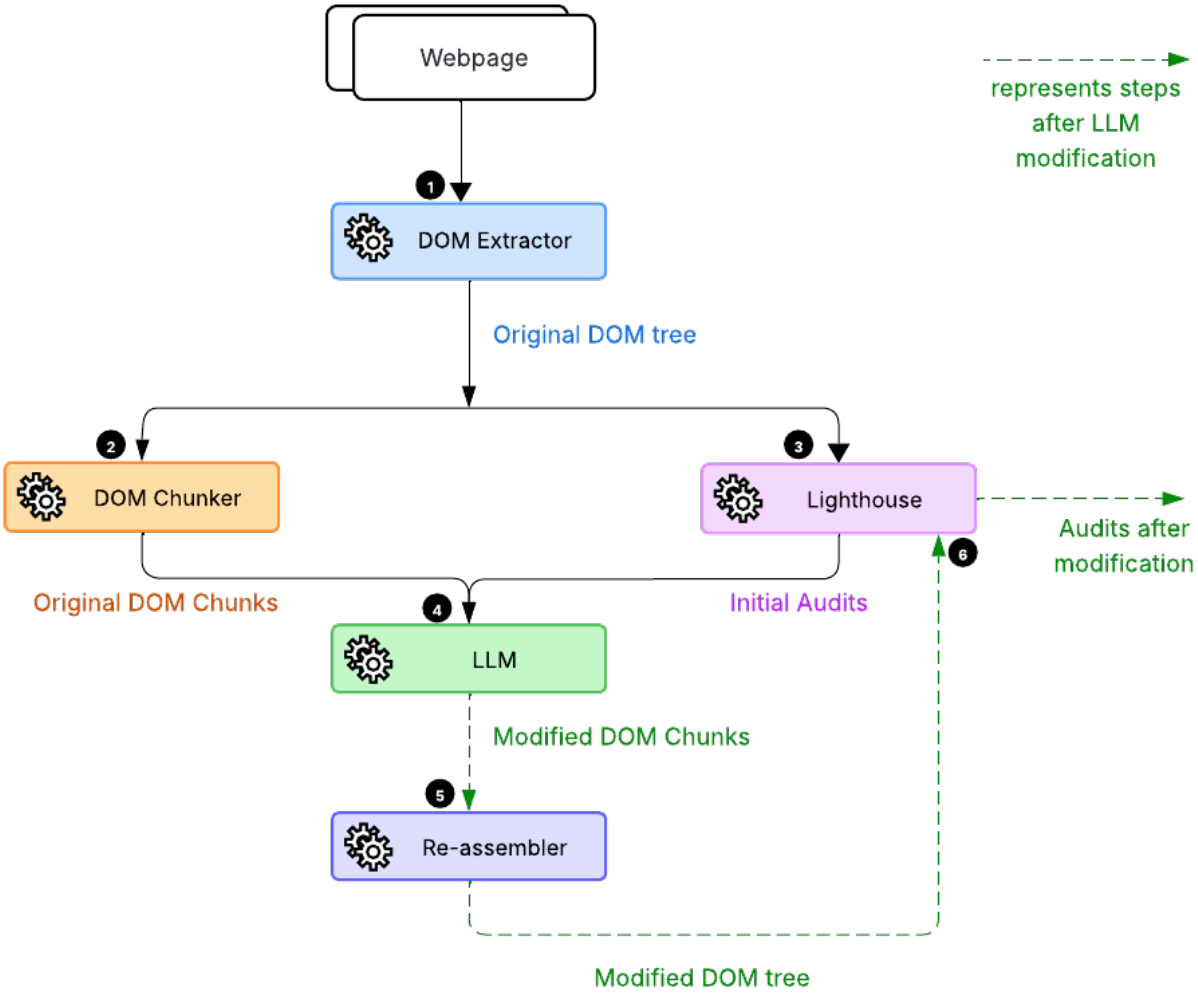

Figure 3.1: Overview of our experiment workflow

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.

3.1.2 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 3.3 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.

Table 3.3: 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

3.1.3 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 [51]. 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 [52]. 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, reincorporating 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.

Algorithm 1 outlines our *token-aware HTML chunking strategy*, which recursively traverses the DOM and segments content into context-constrained units. The algorithm takes an HTML document (D) and a maximum token limit ($T_{\max}$) as input, aiming to produce a list of chunks (C).

The process begins by calling `SplitHTML(D)` (line 4), which iterates through the document's head and body sections, initiating the chunking process for each via `ProcessSection` (line 6). The core logic resides within the `ProcessNode(e)` function (line 15), which recursively traverses the DOM tree.

For **comment or text nodes** (line 16), the algorithm calculates their token length (t_e). If adding this to the current chunk's token count (t) would exceed $T_{\max}$ (line 18), a `StartNewChunk()` operation is triggered (line 18), ensuring no chunk surpasses the limit. The node is then appended to the current chunk (c), and t is updated (line 19).

For **tag nodes** (line 20), the algorithm first checks if the opening and closing tag bounds alone ($t_{\texttt{open+close}}$) would push the current chunk over $T_{\max}$ (line 22). If so, a new chunk is started. The opening tag is appended (line 23), and then `ProcessNode` is recursively called for each of the tag's children (line 24), preserving the hierarchical structure. Finally, the closing tag is appended (line 25).

The `StartNewChunk(name)` function (line 29) handles the creation of new chunks. It first calls `FinishChunk()` to finalize any existing chunk (line 30), then initializes a new empty chunk (c) with a **unique UUID** and its token count (t) reset to 0 (lines 31-32). This UUID ensures faithful reassembly by allowing us to accurately track and reintegrate elements like scripts, styles, and tag attributes during reconstruction. The `FinishChunk()` function (line 35) simply appends a completed chunk to the final list C if it contains content. This recursive and token-aware splitting ensures that each chunk's estimated token length does not exceed our predefined 15K tokens threshold.

Algorithm 1 HTML Chunking Algorithm

Require: HTML document D, token limit $T_{\max}$
Ensure: List of chunks C

$C \leftarrow [\,]$, $c \leftarrow$ `None`, $t \leftarrow 0$
function SplitHTML(D)
    for all $s \in \{D.\texttt{head}, D.\texttt{body}\}$ **do**
        if $s \neq$ `None` **then** ProcessSection(s, section name)
        end if
    end for
    FinishChunk
    return C
end function
function ProcessSection(s, name)
    StartNewChunk(name)
    for all $e \in s.children$ **do**
        ProcessNode(e)
    end for
end function
function ProcessNode(e)
    if e is comment or text **then**
        $t_e \leftarrow$ token length of e
        if $t + t_e > T_{\max}$ **then** StartNewChunk
        end if
        Append e to c, update t
    else if e is tag **then**
        $t_{\text{open+close}} \leftarrow$ token length of tag bounds
        if $t + t_{\text{open+close}} > T_{\max}$ **then** StartNewChunk
        end if
        Append opening tag to c
        for all $child \in e.children$ **do** ProcessNode($child$)
        end for
        Append closing tag to c
    end if
end function
function StartNewChunk($name$)
    FinishChunk
    $c \leftarrow$ new chunk with UUID, label $name$, empty content
    $t \leftarrow 0$
end function
function FinishChunk
    if $c \neq$ `None` and $c.\texttt{content} \neq \emptyset$ **then**
        Append c to C
    end if
end function

During reassembly, we reintegrate the stored scripts, styles, and attributes into their placeholders, reconstructing the original HTML with its semantic and functional integrity preserved. While the reassembly process is not part of the chunking algorithm, it is available in our replication package. To validate the integrity of our chunking and reassembly, we reassembled all chunks before any modifications, confirming that the structure and content remained unchanged.

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 [53, 54]. 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.

3.1.4 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 research 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 *scoreDisplayMode* 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 [33], 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 [23]. 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.Âă
- **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 [55].

3.1.5 Environment Configurations

To ensure the reproducibility and consistency of our performance audits, we utilized a Docker-isolated environment [56]. 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.

3.1.6 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 [29]. 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:

$$\text{AIR} = \frac{W_a}{W_{\text{total}}} \tag{3.1}$$

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:

$$\%\text{ change in AIR} = \frac{M - O}{O} \times 100 \tag{3.2}$$

where M is the *AIR* after LLM modification and O is the original *AIR*. This metric quantifies the effectiveness

of LLMs in resolving identified performance issues. A negative percentage change indicates an improvement (i.e., a reduction in the incidence of an audit), while a positive change suggests a degradation in performance.

Chapter 4

Results

In this chapter, we present the results of our thesis. For each research question, we include the motivation behind it, describe our approach, and report the corresponding findings.

4.1 RQ1: How effective are LLMs for resolving web performance issues in the DOM?

4.1.1 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 [21, 46, 1, 57, 58]. 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.

4.1.2 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 Chapter 3). Subsequently, each webpage is split into chunks to address LLMs' output token limitation, as detailed in our study design Chapter 3). For each webpage, we then iteratively pass each chunk and the performance audits to the LLMs. Through zero-shot prompting [59], 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).

The prompt utilized can be found in Section 7.2.4. It is specifically tailored 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 [60]. 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*.

4.1.3 Results

Table 4.1 highlights the percentage change in *AIRs* after LLM modification of the webpages in our dataset. It presents the various audit categories (see Chapter 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 4.1, 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 [61, 20], 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 [61, 62].

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

Table 4.1: 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**

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 4.1, 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 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.

4.2 RQ2: What is the nature of changes made by LLMs for automated web performance issue resolution?

4.2.1 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.

4.2.2 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 [63]. 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:** This captures modifications to the content or properties of existing items that remain in place. This category provides further detail:
  - **attr_changes:** Specific changes to the values of existing attributes (e.g., `width="100"` to `width="50"`).
  - **tag_changes:** Alterations to an element's HTML tag name (e.g., to <p>).
  - **text_changes:** Modifications to the textual content within an HTML element or a standalone text node.
  - **positional_changes:** This critical metric quantifies changes in the order or relative placement of items within a sequence, capturing reordering or shifts due to additions/deletions.

4.2.3 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 changes performed by the LLMs across the dataset, as specified by `Deepdiff`. Table 4.2 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).

Table 4.2: Summary of Deepdiff Results by LLM Model with Calculated Ratios (including Depth stats)

Model	EATRR	PCD	Attributes		Elements		Types	Change Depth			Values Changed			
			Added	Removed	Added	Removed	Changed	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

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 4.2, 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 [64, 65]. 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 4.2 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 [65]. 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 [64]. 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.

Chapter 5

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.

5.1 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 [61, 62]. 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 [66]. Some examples of related LLM changes we identified through manual inspection include adding `alt` descriptions to image elements [67] and introducing additional `meta` elements [68].

This can significantly reduce the manual effort and expertise required to maintain compliance with necessary web standards [69, 70]. 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 thesis 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 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 [64]. Future research must, therefore, intensively focus on enhancing LLMs' spatial reasoning and visual prediction capabilities within the context of DOM manipulation.

5.2 Unpacking LLM Modification Strategies: Additive vs. Disruptive Approaches

Our analysis of DOM modification types revealed distinct strategies employed by the LLMs in this thesis, which are 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 [65]. 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 [65].

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 [65].

5.3 Overall Implications for Automated Web Performance Resolution

The findings collectively paint a nuanced picture of LLMs as powerful, yet currently incomplete, tools for fully automated web performance resolution. Their strong performance in semantic understanding makes them helpful for a significant subset of web optimization tasks, especially for SEO & Accessibility. However, for more complex performance issues, particularly visual stability, a more discerning and nuanced approach is required. The key implications are:

- **Hybrid Approaches are Essential:** Given the mixed results, a hybrid approach combining LLM-driven optimization with robust post-processing validation and potentially human oversight is crucial. Automated pipelines should incorporate CI/CD checks for performance metrics, especially visual stability and latency, to catch and rectify any regressions introduced by LLM modifications.

- **Targeted LLM Deployment:** Organizations can strategically deploy LLMs for specific, well-defined web performance tasks where their strengths are proven, such as semantic optimization. For more sensitive performance areas, careful model selection and rigorous testing are paramount. As demonstrated by the contrast between high-performing models and GPT-4o-Mini, not all LLMs are created equal for web performance tasks. Characterizing a model's typical DOM modification patterns (e.g., EATRR, PCD, depth of changes) can be predictive of its performance outcomes and should inform model selection.

- **Need for Enhanced LLM Capabilities:** Future research should focus on addressing the identified limitations, particularly in LLMs' hierarchical and spatial understanding of the DOM to improve

visual stability. This might involve developing new architectures that incorporate more sophisticated representations of the layout and spatial relationships during training. Furthermore, exploring prompt engineering techniques that explicitly guide LLMs to consider visual impacts could be beneficial.

In conclusion, LLMs hold immense promise for revolutionizing web performance optimization by automating complex and tedious tasks. However, realizing this potential requires a clear understanding of their current strengths and weaknesses, necessitating a thoughtful integration into existing development workflows and continued research to bridge the existing gaps in their capabilities.

Chapter 6

Threats to Validity

In this chapter, we discuss threats to the validity of our research and explain how we addressed potential limitations in our study design. We categorize these threats as internal and external validity.

6.1 Internal Validity

The internal validity of our thesis is primarily affected by two factors.

First, the performance of LLMs is inherently sensitive to prompt engineering and input design. Different prompt formulations can lead to substantially different outputs. To mitigate this threat, we carefully crafted standardized zero-shot prompts that explicitly described the optimization goals and provided clear instructions for modifications, as described in Chapter 3. We applied identical prompt structures across all evaluated LLMs to minimize variability caused by prompt inconsistencies.

Second, processing large and complex DOM trees necessitated a chunking strategy to fit within LLM context window limitations. This approach could theoretically lead to context loss between chunks, potentially affecting the quality and consistency of modifications. We mitigated this threat by designing a conservative, token-aware chunking strategy that preserved structural integrity, stored metadata for reassembly, and ensured logical divisions that respected semantic boundaries. To further validate the correctness of the chunking and reassembly process, we computed Tree Edit Distances between the original and reconstructed DOM trees prior to any modifications [53, 54], consistently observing a distance of zero. This provides confidence that our approach maintained the original structure without unintended alterations.

Future work could explore alternative chunking techniques (e.g., hierarchical context caching or retrieval-

augmented generation) to further mitigate any remaining risk of context fragmentation during optimization tasks.

6.2 External Validity

The external validity of our thesis relates to the extent to which our findings can be generalized to other web applications, LLMs, and performance audit tools. We identify three primary factors:

First, our evaluation focused on a set of 15 real-world webpages sampled from the Alexa Top 500 list [43]. While this dataset was intentionally diverse in structure, content, and purpose, it may not fully capture all types of web applications, such as highly dynamic single-page applications or sites using unconventional rendering strategies. To mitigate this, we selected webpages across multiple categories (shopping, social, entertainment, professional) and ensured a range of DOM depths, chunk counts, and performance profiles.

Second, while we evaluated nine state-of-the-art LLMs with diverse architectures, reasoning capabilities, and token limits, our findings represent a snapshot of these models at the time of our experiments. As newer LLMs are released, their performance characteristics may differ in important ways, including improved reasoning about hierarchical structures or increased context capacity. Although we expect our findings on strengths (e.g., semantic optimization) and weaknesses (e.g., visual stability regressions) to remain relevant, their frequency and magnitude may shift. Future work can explore how these trends evolve with next-generation models and fine-tuned variants.

Third, this thesis exclusively used Lighthouse as the audit framework to generate performance reports before and after LLM modification [29]. While Lighthouse is widely adopted and considered the industry standard, other tools (e.g., WebPageTest, PageSpeed Insights) may surface different issues or weight metrics differently. To mitigate this limitation, we adopted Lighthouse's recommended configuration flags to ensure consistent and reproducible results and to align our metrics with prior research [2, 71, 4]. However, our evaluation did not directly measure end-user experience in production environments (e.g., computational cost, inference latency, perceived rendering smoothness) [72, 73]. These dimensions remain important and should be considered in future studies combining automated DOM optimization with user experience assessments and real-world deployment benchmarks.

Chapter 7

Conclusion and Future Work

In this chapter, we present a summary of the thesis and contributions to the field of web performance optimization through automated DOM manipulation using Large Language Models (LLMs). We also discuss directions for future research at the end of the chapter.

7.1 Conclusion

Our evaluation details the strengths and limitations of LLMs in automated web performance resolution, highlighting their effective applications and challenges for DOM manipulation.

We evaluated nine state-of-the-art LLMs for automated web performance issues resolution. The models demonstrated universal proficiency in SEO & Accessibility optimization, leveraging strong semantic and structural DOM understanding for such issues. This aligns with prior research on automated semantic validation and enhancement [61, 62], suggesting LLMs are powerful tools for tasks like adding `alt` attributes or `meta` elements, thereby reducing manual effort in maintaining web standards.

However, their impact on other performance-critical issues (latency, network, resource) was highly variable; most models introduced visual instability due to a limited hierarchical and spatial understanding of the DOM. This limitation poses a significant barrier to full automation, as seemingly minor insertions or attribute changes can inadvertently alter 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 [64].

We observed LLMs predominantly employ an additive DOM modification strategy, sometimes leading to DOM bloat due to the introduction of more elements than they remove (e.g., duplicated SVG paths). This can counteract performance gains. Notably, GPT-4o-mini exhibited a unique and disruptive strategy, characterized by removing more elements than it adds, coupled with a high percentage of changes in depth (PCD). 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, 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 [65].

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. 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 [65].

In conclusion, LLMs are powerful, yet currently incomplete, tools for fully automated web performance resolution. Their strong performance in semantic understanding makes them helpful for a significant subset of web optimization tasks, especially for SEO & Accessibility. However, for more complex performance issues, particularly visual stability, a more discerning and nuanced approach is required. This necessitates a nuanced LLM deployment approach, prioritizing enhancements in visual stability and precise, depth-aware modifications. Until these advancements, robust post-hoc validation and potentially human oversight are crucial. The ideal future involves hybrid human-AI approaches where LLMs augment developers, automating where they excel and ensuring precision elsewhere for a faster, more accessible web.

7.2 Future Work

This thesis represents a foundational step into the automated resolution of web performance issues using Large Language Models (LLMs). Our findings illuminate several promising avenues for future research,

building upon the capabilities and challenges identified in this work.

7.2.1 Enhancing Visual Stability and Spatial Understanding

The challenges observed in maintaining visual stability following LLM-generated modifications underscore a critical area for improvement. Future research could investigate novel architectural designs for LLMs or advanced prompt engineering techniques specifically aimed at enhancing their hierarchical and spatial understanding of the Document Object Model (DOM). This might involve explicitly guiding LLMs to consider the visual impact of their proposed modifications on page layout and rendering. Potential avenues include incorporating visual feedback loops during the optimization process or developing training methodologies that leverage datasets augmented with visual difference metrics. Such approaches would enable LLMs to better anticipate and mitigate negative visual side effects, ensuring that performance optimizations do not compromise user experience.

7.2.2 Promoting Efficient DOM Modification Strategies

Our findings highlighted the prevalence of additive DOM modification strategies and, in some instances, the issue of DOM bloat, particularly through the duplication of assets. This indicates a clear need for LLMs that can employ more efficient and less disruptive modification strategies. Future work could explore methods to encourage LLMs to prioritize modifications that inherently minimize DOM size, reduce unnecessary reflows and repaints, and promote resource reuse. This could potentially be achieved by integrating DOM tree size, complexity metrics, or even performance audit scores directly into the LLMs' objective functions during fine-tuning or reinforcement learning. Encouraging more parsimonious and intelligent modifications would lead to cleaner, more maintainable, and truly optimized web code.

7.2.3 Assessing Developer Utility and Real-World Impact

While this thesis rigorously evaluates the quality of LLM-generated modifications through quantitative metrics like audit incidence ratios and qualitative manual inspection, it does not fully assess their practical utility for developers in real-world debugging or ongoing maintenance tasks. Future research could conduct comprehensive user studies involving web developers to evaluate the practical impact of LLM-generated optimizations on their workflows, debugging efficiency, and overall website maintainability. This includes

assessing whether the automatically suggested modifications truly lead to a better developer experience, reduce manual effort, and contribute to more robust and easily manageable web applications in production environments. Such studies would provide invaluable insights into the adoption and real-world applicability of LLM-driven performance tools.

7.2.4 Exploring Multi-Modal Approaches for Holistic Understanding

Finally, exploring the potential of multi-modal LLMs that can process both code (such as the DOM structure, HTML, CSS, and JavaScript) and visual representations of webpages could significantly enhance their ability to reason about complex issues like visual stability and layout changes. Such advanced models could bridge the current gap between semantic understanding of code and visual comprehension of its rendered output. By simultaneously analyzing the underlying code and the visual appearance of a page, multi-modal LLMs could potentially offer a more holistic understanding of web performance, enabling more accurate diagnoses and more effective, visually aware performance optimizations. This direction holds promise for addressing the most challenging aspects of web optimization that involve the interplay of code and visual rendering.

Bibliography

[1] Enrico Bocchi, Luca De Cicco, and Dario Rossi. Measuring the quality of experience of web users. *ACM SIGCOMM Computer Communication Review*, 46(4):8–13, 2016.

[2] Thomas McGill, Oluwaseun Bamgboye, Xiaodong Liu, and Chathuranga Sampath Kalutharage. Towards improving accessibility of web auditing with google lighthouse. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1594–1599, 2023. doi:10.1109/COMPSAC57700.2023.00246.

[3] Shailesh Kumar Shivakumar. 3 - optimizing performance of enterprise web application. In Shailesh Kumar Shivakumar, editor, *Architecting High Performing, Scalable and Available Enterprise Web Applications*, pages 101–141. Morgan Kaufmann, Boston, 2015. ISBN 978-0-12-802258-0. doi:https://doi.org/10.1016/B978-0-12-802258-0.00003-2. URL https://www.sciencedirect.com/science/article/pii/B9780128022580000032.

[4] Muhammad Arif Faizin, Muhammad Nevin, and Umi Laili Yuhana. Indonesia e-government website performance and accessibility evaluation using automated tool lighthouse. In *2024 2nd International Conference on Software Engineering and Information Technology (ICoSEIT)*, pages 210–215, 2024. doi:10.1109/ICoSEIT60086.2024.10497521.

[5] Jasper van Riet, Ivano Malavolta, and Taher A. Ghaleb. Optimize along the way: An industrial case study on web performance. *Journal of Systems and Software*, 198:111593, 2023. ISSN 0164-1212. doi:https://doi.org/10.1016/j.jss.2022.111593. URL https://www.sciencedirect.com/science/article/pii/S0164121222002692.

[6] Morgan Persson. Javascript dom manipulation performance: Comparing vanilla javascript and leading javascript front-end frameworks, 2020.

[7] Dariusz Chęć and Ziemowit Nowak. The performance analysis of web applications based on virtual dom and reactive user interfaces. In *Engineering Software Systems: Research and Praxis*, pages 119–134. Springer, 2019.

[8] Lauren Wood, Arnaud Le Hors, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Gavin Nicol, Jonathan Robie, Robert Sutor, et al. Document object model (dom) level 1 specification. *W3C recommendation*, 1, 1998.

[9] Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. Dompletion: Dom-aware javascript code completion. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, page 43âĂŞ54, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330138. doi:10.1145/2642937.2642981. URL https://doi.org/10.1145/2642937.2642981.

[10] Utkarsh Goel, Stephen Ludin, and Moritz Steiner. Web performance with android's battery-saver mode. *arXiv preprint arXiv:2003.06477*, 2020.

[11] BM Subraya and SV Subrahmanya. Object driven performance testing of web applications. In *Proceedings First Asia-Pacific Conference on Quality Software*, pages 17–26. IEEE, 2000.

[12] Şevval Seray Macakoğlu and Serhat Peker. 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)*, pages 1–8. IEEE, 2022.

[13] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620*, 2023.

[14] Vassilka D. Kirova, Cyril S. Ku, Joseph R. Laracy, and Thomas J. Marlowe. 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*, SIGCSE 2024, page 666âĂŞ672, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704239. doi:10.1145/3626252.3630927. URL https://doi.org/10.1145/3626252.3630927.

[15] 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. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea Preprints*, 2024.

[16] Tommaso Calò and Luigi De Russis. Leveraging large language models forÂăend-user website generation. In Lucio Davide Spano, Albrecht Schmidt, Carmen Santoro, and Simone Stumpf, editors, *End-User Development*, pages 52–61, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-34433-6.

[17] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36, 2024.

[18] Rebeka Tóth, Tamas Bisztray, and László Erdodi. Llms in web-development: Evaluating llm-generated php code unveiling vulnerabilities and limitations. *arXiv preprint arXiv:2404.14459*, 2024.

[19] Michel Nass, Emil Alégroth, and Robert Feldt. Improving web element localization by using a large language model. *Software Testing, Verification and Reliability*, page e1893, 2023.

[20] Juan-Miguel López-Gil and Juanan Pereira. Turning manual web accessibility success criteria into automatic: an llm-based approach. *Universal Access in the Information Society*, pages 1–16, 2024.

[21] Tjaša Heričko, Boštjan Šumak, and Saša Brdnik. Towards representative web performance measurements with google lighthouse. In *Proceedings of the 2021 7th Student Computer Science Research Conference*, page 39, 2021.

[22] Thomas McGill, Oluwaseun Bamgboye, Xiaodong Liu, and Chathuranga Sampath Kalutharage. Towards improving accessibility of web auditing with google lighthouse. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1594–1599. IEEE, 2023.

[23] Anonymous. Automated resolution of web performance issues using llms: A case study of gpt-4o-mini, January 2025. URL https://doi.org/10.5281/zenodo.14785704.

[24] Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document object model (dom) level 3 core specification, 2000.

[25] World Wide Web Consortium et al. Document object model (dom) level 3 core specification. 2004.

[26] Andreas B Gizas and Sotiris P Christodoulou. Performance-optimized pages' architecture, navigation and images techniques for jquery mobile sites. In *Proceedings of the 19th Panhellenic Conference on Informatics*, pages 371–377, 2015.

[27] Simo Kuparinen. Improving web performance by optimizing cascading style sheets (css): literature review and empirical findings. *Helsinki University Library*, 1(2), 2023.

[28] Shailesh Kumar Shivakumar. Modern web performance optimization. *Methods, Tools, and Patterns to Speed Up Digital Platforms*, 2020.

[29] Chrome for Developers. Lighthouse overview, 2016. URL https://developer.chrome.com/docs/lighthouse/overview. Accessed: 2024-09-04.

[30] Justin Scherer. *Hands-on JavaScript High Performance: Build Faster Web Apps Using Node. js, Svelte. js, and WebAssembly*. Packt Publishing Ltd, 2020.

[31] Google for Developers. Pagespeed insights, 2016. URL https://pagespeed.web.dev. Accessed: 2024-09-04.

[32] Raimundo N.V. Diniz-Junior, Caio CÃľsar 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. Evaluating the performance of web rendering technologies based on javascript: Angular, react, and vue. In *2022 XVLIII Latin American Computer Conference (CLEI)*, pages 1–9, 2022. doi:10.1109/CLEI56649.2022.9959901.

[33] Google Chrome. Lighthouse - understanding the results, 2024. URL https://github.com/GoogleChrome/lighthouse/blob/main/docs/understanding-results.md.

[34] Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. Understanding html with large language models. *ArXiv*, abs/2210.03945, 2022. URL https://api.semanticscholar.org/CorpusID:252780086.

[35] Aman Ahluwalia and Suhrud Wani. Leveraging large language models for web scraping. *arXiv preprint arXiv:2406.08246*, 2024.

[36] Prateek Sancheti, Kamalakar Karlapalem, and Kavita Vemuri. Llm driven web profile extraction for identical names. In *Companion Proceedings of the ACM Web Conference 2024*, pages 1616–1625, 2024.

[37] Zixuan Li, Yutao Zeng, Yuxin Zuo, Weicheng Ren, Wenxuan Liu, Miao Su, Yucan Guo, Yantao Liu, Xiang Li, Zhilei Hu, et al. Knowcoder: Coding structured knowledge into llms for universal information extraction. *arXiv preprint arXiv:2403.07969*, 2024.

[38] CP Afsal and KS Kuppusamy. Websumm: A chrome extension for summarizing web content using llms for visually impaired users. *SN Computer Science*, 6(2):1–15, 2025.

[39] Tommaso Calò and Luigi De Russis. Leveraging large language models for end-user website generation. In *International Symposium on End User Development*, pages 52–61. Springer, 2023.

[40] Amanda Li, Jason Wu, and Jeffrey P Bigham. Using llms to customize the ui of webpages. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23 Adjunct, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700965. doi:10.1145/3586182.3616671. URL https://doi.org/10.1145/3586182.3616671.

[41] Brian Tang and Kang G Shin. Steward: Natural language web automation. *arXiv preprint arXiv:2409.15441*, 2024.

[42] Qifan Wang, Yi Fang, Anirudh Ravula, Fuli Feng, Xiaojun Quan, and Dongfang Liu. Webformer: The web-page transformer for structure information extraction. In *Proceedings of the ACM Web Conference 2022*, pages 3124–3133, 2022.

[43] Amazon. Amazon alexa top sites. URL https://www.alexa.com/topsites.

[44] Kwame Chan-Jong-Chu, Tanjina Islam, Miguel Morales Exposito, Sanjay Sheombar, Christian Valladares, Olivier Philippot, Eoin Martino Grua, and Ivano Malavolta. 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*, pages 190–199, 2020.

[45] Sonai Mahajan, Negarsadat Abolhassani, Phil McMinn, and William GJ Halfond. Automated repair of mobile friendly problems in web pages. In *Proceedings of the 40th international conference on software engineering*, pages 140–150, 2018.

[46] Frolin S Ocariza Jr, Karthik Pattabiraman, and Benjamin Zorn. Javascript errors in the wild: An empirical study. In *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, pages 100–109. IEEE, 2011.

[47] Gary L Geissler, George M Zinkhan, and Richard T Watson. The influence of home page complexity on consumer attention, attitudes, and purchase intent. *Journal of Advertising*, 35(2):69–80, 2006.

[48] Zizi Papacharissi. The self online: The utility of personal home pages. *Journal of Broadcasting & Electronic Media*, 46(3):346–368, 2002.

[49] Surendra N Singh, Nikunj Dalal, and Nancy Spears. Understanding web home page perception. *European Journal of Information Systems*, 14(3):288–302, 2005.

[50] PyPI. Python beautifulsoup library. URL https://pypi.org/project/beautifulsoup4.

[51] OpenAI. Gpt-4o, 2024. URL https://platform.openai.com/docs/models/gpt-4o.

[52] OpenAI. tiktoken, 2024. URL https://github.com/openai/tiktoken. Accessed: 2024-10-14.

[53] Kilho Shin, Taichi Ishikawa, Yu-Lu Liu, and David Lawrence Shepard. Learning dom trees of web pages by subpath kernel and detecting fake e-commerce sites. *Machine Learning and Knowledge Extraction*, 3 (1):95–122, 2021.

[54] Manuel Leithner and Dimitris E Simos. Domdiff: Identification and classification of inter-dom modifications. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 262–269. IEEE, 2018.

[55] Hamza Salem, Hadi Salloum, Osama Orabi, Kamil Sabbagh, and Manuel Mazzara. Enhancing news articles: Automatic seo linked data injection for semantic web integration. *Applied Sciences*, 15(3): 1262, 2025.

[56] Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2, 2014.

[57] Anita Crescenzi, Diane Kelly, and Leif Azzopardi. Impacts of time constraints and system delays on user experience. In *Proceedings of the 2016 acm on conference on human information interaction and retrieval*, pages 141–150, 2016.

[58] Marcus Basalla, Johannes Schneider, Martin Luksik, Roope Jaakonmäki, and Jan Vom Brocke. On latency of e-commerce platforms. *Journal of Organizational Computing and Electronic Commerce*, 31 (1):1–17, 2021.

[59] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[60] Hernán Ceferino Vázquez, Alexandre Bergel, Santiago Vidal, JA Díaz Pace, and Claudia Marcos. Slimming javascript applications: An approach for removing unused functions from javascript libraries. *Information and software technology*, 107:18–29, 2019.

[61] Shaomei Wu, Jeffrey Wieland, Omid Farivar, and Julie Schiller. 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*, pages 1180–1192, 2017.

[62] Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and Hong Lin. Ai-generated content (aigc): A survey. *arXiv preprint arXiv:2304.06632*, 2023.

[63] Zepworks. Deepdiff. URL https://zepworks.com/deepdiff/current/diff.html.

[64] Google Developers. Understanding core web vitals and google search results. URL https://developers.google.com/search/docs/appearance/core-web-vitals.

[65] Google Developers (Lighthouse). Avoid an excessive dom size. URL https://developer.chrome.com/docs/lighthouse/performance/dom-size.

[66] Giovanni Delnevo, Manuel Andruccioli, and Silvia Mirri. On the interaction with large language models for web accessibility: Implications and challenges. In *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*, pages 1–6. IEEE, 2024.

[67] Tahani Alahmadi and Steve Drew. Evaluation of image accessibility for visually impaired users. *Journal of Accessibility and Design for All*, 8(2):125–160, 2018.

[68] Konstantinos I Roumeliotis and Nikolaos D Tselikas. An effective seo techniques and technologies guide-map. *Journal of web engineering*, 21(5):1603–1649, 2022.

[69] Accessibility Guidelines Working Group. Web content accessibility guidelines (wcag) 2.1. https://www.w3.org/TR/WCAG21/, June 2018. Recommendation of the Web Accessibility Initiative (WAI).

[70] MDN contributors. <meta>: The metadata element, June 2025. URL https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/meta. MDN Web Docs.

[71] Darius Saif, Chung-Horng Lung, and Ashraf Matrawy. An early benchmark of quality of experience between http/2 and http/3 using lighthouse. In *ICC 2021-IEEE international conference on communications*, pages 1–6. IEEE, 2021.

[72] Bill Albert and Tom Tullis. *Measuring the user experience: Collecting, analyzing, and presenting UX metrics*. Morgan Kaufmann, 2022.

[73] Nikolas Wehner, Monisha Amir, Michael Seufert, Raimund Schatz, and Tobias Hoßfeld. A vital improvement? relating google's core web vitals to actual web qoe. In *2022 14th international conference on quality of multimedia experience (QoMEX)*, pages 1–6. IEEE, 2022.

Appendices

Audit definitions and Categories

Table A.1: Audit names grouped by category

Category	Audit Name
SEO & Accessibility	
	crawlable-anchors
	link-text
	is-crawlable
	meta-description
	hreflang
	aria-prohibited-attr
	aria-hidden-focus
	image-alt
	aria-allowed-attr
	listitem
	list
	aria-dialog-name
	label-content-name-mismatch
	input-button-name
	html-lang-valid

Continued on next page

Category	Audit Name
	aria-tooltip-name
	link-name
Network Optimization	
	uses-rel-preconnect
	uses-http2
	third-party-cookies
	is-on-https
	total-byte-weight
	uses-text-compression
	uses-long-cache-ttl
	redirects
Initial Load Performance	
	charset
	lcp-lazy-loaded
	offscreen-images
	render-blocking-resources
	first-contentful-paint
	speed-index
	largest-contentful-paint-element
	prioritize-lcp-image
	largest-contentful-paint
Visual Stability	
	viewport
	meta-viewport
	image-size-responsive
	image-aspect-ratio

Continued on next page

Category	Audit Name
	font-size
	color-contrast
	target-size
	dom-size
	unsized-images
	font-display
	cumulative-layout-shift
	layout-shifts
Runtime Performance	
	valid-source-maps
	inspector-issues
	errors-in-console
	deprecations
	bootup-time
	mainthread-work-breakdown
	third-party-summary
	no-document-write
Resource Optimization	
	duplicated-javascript
	modern-image-formats
	legacy-javascript
	unminified-css
	uses-optimized-images
	unused-css-rules
	uses-responsive-images
	unused-javascript

Continued on next page

Category	Audit Name
	unminified-javascript
Interactivity Performance	
	uses-passive-event-listeners
	total-blocking-time
	ma-potential-fid
	interactive

Table A.2: Audit data overview

S/N	Audit Name	Audit Description	Count	Websites	Coverage
1	first-contentful-paint	First Contentful Paint marks the time at which the first text or image is painted	15	airbnb, aliexpress, ebay, facebook, github, linkedin, medium, netflix, pinterest, quora, reddit, twitch, twitter, walmart, youtube	100.0
2	speed-index	Speed Index shows how quickly the contents of a page are visibly populated	15	airbnb, aliexpress, ebay, facebook, github, linkedin, medium, netflix, pinterest, quora, reddit, twitch, twitter, walmart, youtube	100.0

Continued on next page

S/N	Audit Name	Audit Description	Count	Websites	Coverage
3	total-blocking-time	Sum of all time periods between FCP and Time to Interactive, when task length exceeded 50ms, expressed in milliseconds	15	airbnb, aliexpress, ebay, facebook, github, linkedin, medium, netflix, pinterest, quora, reddit, twitch, twitter, walmart, youtube	100.0
4	ma-potential-fid	The maximum potential First Input Delay that your users could experience is the duration of the longest task	15	airbnb, aliexpress, ebay, facebook, github, linkedin, medium, netflix, pinterest, quora, reddit, twitch, twitter, walmart, youtube	100.0
5	interactive	Time to Interactive is the amount of time it takes for the page to become fully interactive	15	airbnb, aliexpress, ebay, facebook, github, linkedin, medium, netflix, pinterest, quora, reddit, twitch, twitter, walmart, youtube	100.0

Continued on next page

S/N	Audit Name	Audit Description	Count	Websites	Coverage
6	unused-javascript	Reduce unused JavaScript and defer loading scripts until they are required to decrease bytes consumed by network activity	15	airbnb, aliexpress, ebay, facebook, github, linkedin, medium, netflix, pinterest, quora, reddit, twitch, twitter, walmart, youtube	100.0
7	largest-contentful-paint-element	This is the largest contentful element painted within the viewport	14	airbnb, aliexpress, ebay, facebook, github, linkedin, netflix, pinterest, quora, reddit, twitch, twitter, walmart, youtube	93.33
8	largest-contentful-paint	Largest Contentful Paint marks the time at which the largest text or image is painted	14	airbnb, aliexpress, ebay, facebook, github, linkedin, netflix, pinterest, quora, reddit, twitch, twitter, walmart, youtube	93.33

Continued on next page

S/N	Audit Name	Audit Description	Count	Websites	Coverage
9	errors-in-console	Errors logged to the console indicate unresolved problems	14	airbnb, aliexpress, ebay, facebook, github, linkedin, medium, pinterest, quora, reddit, twitch, twitter, walmart, youtube	93.33
10	uses-text-compression	Text-based resources should be served with compression (gzip, deflate or brotli) to minimize total network bytes	14	airbnb, aliexpress, ebay, facebook, github, linkedin, medium, netflix, pinterest, quora, reddit, twitch, walmart, youtube	93.33
11	legacy-javascript	Polyfills and transforms enable legacy browsers to use new JavaScript features	13	airbnb, aliexpress, ebay, facebook, linkedin, medium, netflix, pinterest, quora, twitch, twitter, walmart, youtube	86.67

Continued on next page

S/N	Audit Name	Audit Description	Count	Websites	Coverage
12	third-party-summary	Third-party code can significantly impact load performance	13	airbnb, aliexpress, ebay, github, linkedin, medium, netflix, pinterest, quora, reddit, twitch, walmart, youtube	86.67
13	mainthread-work-breakdown	Consider reducing the time spent parsing, compiling and executing JS	12	airbnb, aliexpress, ebay, github, linkedin, medium, netflix, pinterest, reddit, twitch, walmart, youtube	80.0
14	bootup-time	Consider reducing the time spent parsing, compiling, and executing JS	11	aliexpress, ebay, github, linkedin, medium, netflix, pinterest, reddit, twitch, walmart, youtube	73.33
15	unused-css-rules	Reduce unused rules from stylesheets and defer CSS not used for above-the-fold content to decrease bytes consumed by network activity	11	airbnb, aliexpress, ebay, github, linkedin, netflix, pinterest, quora, reddit, walmart, youtube	73.33

Continued on next page

S/N	Audit Name	Audit Description	Count	Websites	Coverage
16	inspector-issues	Issues logged to the ‘Issues‘ panel in Chrome Devtools indicate unresolved problems	10	airbnb, aliexpress, ebay, linkedin, medium, netflix, twitch, twitter, walmart, youtube	66.67
17	render-blocking-resources	Resources are blocking the first paint of your page	9	airbnb, ebay, facebook, github, linkedin, medium, netflix, reddit, youtube	60.0
18	uses-long-cache-ttl	A long cache lifetime can speed up repeat visits to your page	9	aliexpress, ebay, linkedin, medium, netflix, pinterest, reddit, twitch, walmart	60.0
19	third-party-cookies	Support for third-party cookies will be removed in a future version of Chrome	7	aliexpress, ebay, linkedin, medium, twitch, twitter, walmart	46.67
20	deprecations	Deprecated APIs will eventually be removed from the browser	7	aliexpress, ebay, facebook, pinterest, twitch, walmart, youtube	46.67

Continued on next page

S/N	Audit Name	Audit Description	Count	Websites	Coverage
21	uses-responsive-images	Serve images that are appropriately-sized to save cellular data and improve load time	7	aliexpress, ebay, github, pinterest, reddit, twitch, walmart	46.67
22	dom-size	A large DOM will increase memory usage, cause longer [style calculations](https://developers	7	airbnb, aliexpress, ebay, github, pinterest, reddit, youtube	46.67
23	is-on-https	All sites should be protected with HTTPS, even ones that don't handle sensitive data	7	aliexpress, ebay, linkedin, medium, quora, walmart, youtube	46.67
24	target-size	Touch targets with sufficient size and spacing help users who may have difficulty targeting small controls to activate the targets	6	ebay, facebook, github, pinterest, reddit, twitter	40.0

Continued on next page

S/N	Audit Name	Audit Description	Count	Websites	Coverage
25	modern-image-formats	Image formats like WebP and AVIF often provide better compression than PNG or JPEG, which means faster downloads and less data consumption	6	ebay, github, linkedin, pinterest, reddit, twitch	40.0
26	offscreen-images	Consider lazy-loading offscreen and hidden images after all critical resources have finished loading to lower time to interactive	6	aliexpress, ebay, github, linkedin, netflix, pinterest	40.0
27	unsized-images	Set an explicit width and height on image elements to reduce layout shifts and improve CLS	6	aliexpress, ebay, facebook, github, quora, twitch	40.0
28	cumulative-layout-shift	Cumulative Layout Shift measures the movement of visible elements within the viewport	6	aliexpress, ebay, github, pinterest, twitch, walmart	40.0

Continued on next page

S/N	Audit Name	Audit Description	Count	Websites	Coverage
29	color-contrast	Low-contrast text is difficult or impossible for many users to read	5	aliexpress, ebay, facebook, medium, twitter	33.33
30	font-display	Leverage the ‘font-display‘ CSS feature to ensure text is user-visible while webfonts are loading	5	netflix, pinterest, reddit, twitch, twitter	33.33
31	font-size	Font sizes less than 12px are too small to be legible and require mobile visitors to âĂIJpinch to zoomâĂİ in order to read	5	facebook, netflix, quora, twitch, youtube	33.33
32	uses-passive-event-listeners	Consider marking your touch and wheel event listeners as ‘passive‘ to improve your page’s scroll performance	5	linkedin, pinterest, reddit, twitch, youtube	33.33

Continued on next page

S/N	Audit Name	Audit Description	Count	Websites	Coverage
33	total-byte-weight	Large network payloads cost users real money and are highly correlated with long load times	5	aliexpress, ebay, github, pinterest, youtube	33.33

Prompt for HTML Performance Optimization

```
You are a web performance expert and your task is to optimize the HTML code for
    performance issues.

## Task
Due to the huge token size of HTML files,
You are given the HTML in chunks, one at a time.
You are also given the performance issue(s) detected by Lighthouse to resolve, in the
    following format:
Please modify each HTML code chunk to resolve the performance issue(s) given below.
Return the modified HTML code alone, making only necessary changes for performance
    optimization.

## Instructions
Make sure you:
- Remember the code is split into chunks and you are only receiving one chunk at a
    time, there might be some unclosed or cut elements, do not worry about that.
- Consider that a chunk might be a part of a larger element, so the code might not be
    complete.
- Consider that the HTML as a whole is from production and might be minified, uglified
     or compressed.
- DO NOT modify class names
- Do not change chunk IDs ie the prop values for data-chunk-uuid, and chunk_style_[
    uuid]
- DO NOT remove any comments already in the code.
- DO NOT change any styles or functionalities of the code.
- DO NOT change the structure of the code.
- DO NOT change the order of the code.
- DO NOT remove critical elements.
- If any optimizations are made, return `<!-- Optimized by LLM -->` at the beginning
    point of only the modified portion
  and `<!-- End of Optimization: {{audit_key of issue being resolved}} => {{one line
    short description of elements/things resolved}} -->` at the end of the changed
    portion.
  Do not indicate any resolution outside of the End of Optimization comment, where
    there are multiple resolutions being made, seperate them with commas within the End
```

```
    of optimization comment.
- If you cannot make an optimization due to the above reasons but can provide a very
    brief suggestion, do so by adding a comment at the end of the chunk, starting with
    `<!-- Suggestion: {{audit_key of issue being addressed}} ` and ending with ` -->`.
- Prioritize optimizations above suggestions where you can.
- Return ONLY the modified HTML code, no long notes.
- Making only necessary changes for performance optimization.
- If no optimizations are possible, return the original code.
- Never add any additional comments to the code besides the ones for describing where
    optimizations were made by you.
- If the change is within a `<style>` tag, replace the HTML comment with a CSS comment
    .

## Original HTML Chunk
```html
{{content}}

## Performance Issues
{{audit_issues}}
```

Listing 7.1: Prompt for HTML Performance Optimization

Additional results for LLM modifications

Table A.3: Consistency of model improvements across latency audits.

Audit	Models improved	Worst regression	σ
Initial Load	8/9	+24.8%	22.8
Interactivity	7/9	+7.9%	24.1
Runtime	8/9	+58.9%	40.3
```

Audit Category	Attributes Added	Attributes Removed	Elements Added	Elements Removed	Types Changed	Depth Min	Depth Max	Depth Avg	Values Changed Attr	Values Changed Tag	Values Changed Pos	Values Changed Text
SEO & Accessibility	–	–	–	–	–	–	–	–	–	–	–	–
Initial Load Performance	0.07	0.65	-0.50	0.06	0.05	–	-0.27	-0.32	0.07	-0.10	-0.24	-0.68
Interactivity Performance	0.10	0.25	-0.18	-0.46	0.00	–	-0.49	-0.32	0.34	-0.24	-0.21	-0.94
Runtime Performance	-0.04	0.54	-0.46	-0.26	0.03	–	-0.42	-0.46	0.06	-0.11	-0.24	-0.82
Network Optimization	0.02	0.27	-0.24	-0.22	-0.03	–	-0.35	-0.29	0.07	-0.50	-0.25	-0.80
Resource Optimization	-0.20	0.05	0.01	-0.46	-0.31	–	-0.63	-0.37	-0.04	-0.49	0.02	-0.87
Visual Stability	0.06	0.44	-0.45	-0.27	0.36	–	-0.05	-0.58	0.28	-0.41	-0.55	-0.79

Table A.4: Correlation for all Models

Audit Category	Attr Added	Attr Removed	El Added	El Removed	Types Changed	Depth Min	Depth Max	Depth Median	Values Changed Attr	Values Changed Tag	Values Changed Pos	Values Changed Text
SEO & Accessibility	–	–	–	–	–	–	–	–	–	–	–	–
Initial Load Performance	0.38	0.22	-0.01	-0.10	0.17	–	-0.15	-0.12	0.42	-0.32	-0.32	-0.74
Interactivity Performance	0.17	0.01	0.28	-0.49	0.10	–	-0.37	-0.52	0.47	-0.14	-0.21	-0.96
Runtime Performance	0.20	-0.03	0.08	-0.61	0.14	–	-0.40	-0.36	0.41	-0.34	-0.31	-0.97
Network Optimization	0.02	0.27	-0.24	-0.22	-0.03	–	-0.35	-0.29	0.07	-0.50	-0.25	-0.80
Resource Optimization	-0.11	0.20	-0.04	-0.32	-0.31	–	-0.62	-0.25	0.10	-0.34	0.20	-0.84
Visual Stability	0.60	-0.59	0.92	-0.65	–	–	-0.90	-0.17	0.87	0.87	0.57	-0.90

Table A.5: Correlation of Better Performing Models

Audit Category	Attributes Added	Attributes Removed	Elements Added	Elements Removed	Types Changed	Depth Min	Depth Max	Depth Median	Values Changed Attr	Values Changed Tag	Values Changed Pos	Values Changed Text
SEO & Accessibility	–	–	–	–	–	–	–	–	–	–	–	–
Initial Load Performance	–	–	–	–	–	–	–	–	–	–	–	–
Interactivity Performance	-1.0	1.0	-1.0	1.0	–	–	–	-1.0	-1.0	1.0	–	-1.0
Runtime Performance	–	–	–	–	–	–	–	–	–	–	–	–
Network Optimization	–	–	–	–	–	–	–	–	–	–	–	–
Resource Optimization	–	–	–	–	–	–	–	–	–	–	–	–
Visual Stability	0.75	0.64	-0.68	0.62	0.87	–	0.73	-0.31	0.50	0.69	-0.96	-0.02

Table A.6: Correlations for regressing LLMs