big data and cognitive computing

MDPI

Article

Leveraging Large Language Models for Sustainable and Inclusive Web Accessibility

Manuel Andruccioli, Barry Bassi, Giovanni Delnevo * and Paola Salomoni

Department of Computer Science and Engineering, University of Bologna, 47522 Cesena, Italy; manuel.andruccioli@unibo.it (M.A.); barry.bassi@unibo.it (B.B.); paola.salomoni@unibo.it (P.S.)
* Correspondence: giovanni.delnevo@unibo.it

Abstract

The increasing complexity of modern web applications, which are composed of dynamic and asynchronous components, poses a significant challenge for digital inclusion. Traditional automated tools typically analyze only the static HTML markup generated by frontend and backend frameworks. Recent advances in Large Language Models (LLMs) offer a novel approach to enhance the validation process by directly analyzing the source code. In this paper, we investigate the capacity of LLMs to interpret and reason dynamically generated content, providing real-time feedback on web accessibility. Our findings show that LLMs can correctly anticipate the presence of accessibility violations in the generated HTML code, going beyond the capabilities of traditional validators, also evaluating possible issues due to the asynchronous execution of the web application. However, together with legitimate issues, LLMs also produced a relevant number of hallucinated or redundant violations. This study contributes to the broader effort of employing AI with the aim of improving the inclusivity and equity of the web.

Keywords: web accessibility; large language models; human–AI interaction; digital sustainability

Academic Editors: Fabrizio Marozzo and Riccardo Cantini

Received: 6 August 2025
Revised: 18 September 2025
Accepted: 24 September 2025
Published: 26 September 2025

Citation: Andruccioli, M.; Bassi, B.; Delnevo, G.; Salomoni, P. Leveraging Large Language Models for Sustainable and Inclusive Web Accessibility. *Big Data Cogn. Comput.* **2025**, *9*, 247. https://doi.org/10.3390/bdcc9100247

1. Introduction

The Web has moved from an information-sharing system for scientists and has become an essential tool that influences people's daily lives, professions, education, and social connections [1]. Considering its ever-increasing central role in society, it is important that its technological evolution is guided by principles of inclusivity and fairness [2]. In fact, while the Web can help in addressing sustainability challenges, it needs to follow sustainability principles [3]. On the one hand, it can have a central role in pursuing United Nations Sustainable Development Goals (SDGs), that take account of the economic, social, and ecological dimensions of sustainable development. On the other hand, the development and deployment of Web technologies must adhere to sustainable and ethical practices [4]. Guaranteeing accessibility, reducing both the digital divide and the environmental impact are all objectives mainly related to the SDG 10, which seeks to reduce inequalities within and among countries [5]. For these reasons, the Web must go beyond being a mere tool for sustainability, but incorporate it in its design, implementation, and usage. It is of fundamental importance that future Web systems empower everyone, particularly those from vulnerable and marginalized communities [6].

Web accessibility, which ensures that people with disabilities can perceive, understand, navigate, and interact with the Web, lies at the heart of this vision [7]. How-

ever, despite decades of standards and advocacy, accessibility remains an ongoing challenge—particularly in dynamic, component-based, and media-rich web environments [8]. Existing efforts have primarily focused on static content and compliance-driven evaluations, which often fall short in capturing the interactive and adaptive nature of modern applications [9]. This mismatch between established accessibility practices and the realities of contemporary web development highlights a significant research gap [10]. This gap calls for new approaches that can scale with the complexity of modern web development and bridge the divide between technical implementation and inclusive design [11].

Recent advances in Large Language Model (LLM) and Multimodal AI offer a promising path forward [12]. These models, exemplified by systems such as OpenAI's ChatGPT and Google's Gemini, have shown remarkable potential in a variety of domains, including code generation [13], natural language understanding [14], and image interpretation [15]. Their growing integration into software development workflows opens new possibilities for building more accessible, human-centered web systems [16]. Yet, these technologies also raise important questions about bias, sustainability, and the limits of automation in ethically sensitive contexts [17].

LLMs hold promise in streamlining the validation process for HTML code's adherence to accessibility standards. Their adeptness at interpreting code enables them to scrutinize web content, identifying potential accessibility shortcomings such as absent alternative text, inadequate semantic structuring, or other deviations from accessibility guidelines. This potential application stands to improve the accessibility auditing process while concurrently elevating the overall quality of web content. Ultimately, this ensures the content caters inclusively to the diverse needs of all users.

This paper investigates how LLMs can support web developers in designing accessible web applications, focusing on their ability to validate and improve accessibility at both the code and content level. Through a series of controlled experiments, we assess their performance in scenarios involving dynamically generated content and asynchronous data handling. We analyze the extent to which these tools can understand and reason about accessibility constraints, and we reflect on their potential—and limitations—as responsible AI agents in web development. In doing so, this work contributes to the broader conversation on how emerging web technologies can be leveraged for social good. Our findings aim to inform researchers and practitioners seeking to integrate LLMs into accessibility workflows in a safe, equitable, and sustainable manner—supporting the Web's evolution into a truly inclusive platform for all.

The remainder of the paper goes as follows. Section 2 establishes the background of web accessibility and the role of LLMs in such a context. Section 3 then details the proposed approach. The results of the experiments are presented in Section 4. Following this, Section 5 interprets these results in the context of existing literature and explores the implications for web development practices and accessibility tooling. Finally, Section 6 summarizes the main contributions of our work and suggests avenues for future research in the pursuit of more accessible web environments.

2. Background and Related Work

Several works investigated the application of LLMs for web accessibility. Othman et al. [18] study explored the potential of LLMs, specifically ChatGPT, to automatically remediate web accessibility issues and improve compliance with Web Content Accessibility Guidelines (WCAG) 2.1 guidelines. The research involved using the WAVE tool to identify accessibility errors on two non-compliant websites, subsequently employing ChatGPT for automated remediation. The effectiveness of this LLM-driven approach was evaluated by comparing its results against manual accessibility testing, offering insights for stakeholders

aiming to enhance web accessibility for individuals with disabilities. In parallel to this, the study by Duarte et al. [12] explores the potential of LLMs to automate the detection of heading-related accessibility barriers on webpages, addressing limitations in existing automated tools. The authors evaluate three models—Llama 3.1, GPT-4o, and GPT-4o mini—using a controlled set of modified webpages derived from a WCAG-compliant baseline. They identify seven common heading-related barriers (e.g., missing headings, incorrect hierarchy) and design prompts to query the LLMs. Results reveal model-specific strengths: Llama 3.1 excels in detecting structural and semantic barriers (e.g., heading hierarchy), while GPT-4o mini performs better at identifying nuanced problems like accessible names. However, all models struggle with detecting missing headings, underscoring the need for complementary approaches. Then, in the study of Delnevo et al. [19], authors investigate the potential of LLMs, particularly ChatGPT 3.5, to enhance web accessibility by automating the evaluation and correction of HTML code. Research focuses on critical HTML elements such as forms, tables, and images, assessing their compliance with WCAG accessibility standards. Using a standardized prompt, they evaluate both accessible and non-accessible code snippets, highlighting cases where LLMs successfully identify and rectify issues—such as missing labels in forms or improper semantic structure in tables—while also noting limitations, such as inconsistent suggestions for equivalent accessibility attributes (e.g., scope vs. headers in tables). Moreover, López-Gil and Pereira [16] explore the potential of LLMs to automate the evaluation of WCAG success criteria that traditionally require manual inspection. Focusing on three specific criteria—1.1.1 (Non-text Content), 2.4.4 (Link Purpose), and 3.1.2 (Language of Parts)—the authors develop LLM-based scripts to assess accessibility compliance. Their approach leverages models like ChatGPT, Claude, and Bard to analyze HTML content, compare textual descriptions, and validate language tags. The results demonstrate that LLMs achieve an 87.18% accuracy in detecting issues missed by conventional automated tools, highlighting their potential to augment accessibility testing. Finally, Huynh et al. [20] instead tackled the problem of image accessibility. They presented SmartCaption AI, a tool that leverages LLMs for generating context-aware descriptions of images on web pages. Unlike traditional methods that rely solely on image-to-text algorithms, SmartCaption AI enhances accuracy by first summarizing the surrounding webpage content, providing the LLM with necessary contextual cues. This ensures that generated captions are not only precise but also relevant to the broader page context. The descriptions are then embedded into the webpage's structure, allowing screen readers to seamlessly vocalize them. The proposed system is implemented as a Chrome extension.

From an educational point of view, Aljedaani et al. [21], investigate the integration of LLMs, specifically ChatGPT 3.5, into software engineering education to enhance students' awareness and practical skills in digital accessibility. The study addresses persistent gaps in accessibility compliance, attributed to lack of awareness, knowledge, and practical training among developers. Through an assignment, 215 students evaluated their own websites using accessibility checker tools (WAVE and AChecker), identified violations, and employed ChatGPT to remediate issues. Results demonstrated that 60% of students resolve most violations and 30% of students fixed all violation with ChatGPT, with a general understanding and confidence improvement in implementing accessibility standards.

With regard to mobile applications, Taeb et al. [22] delved into exploring the utilization of LLMs for assessing the accessibility of mobile applications. The researchers introduce a novel system designed to streamline the evaluation process. This system operates by initially taking a manual accessibility test as input, subsequently leveraging an LLM in conjunction with pixel-based UI Understanding models to execute the test. The outcome is a structured video presentation, divided into chapters for enhanced navigation. The

generated video undergoes analysis using specific heuristics aimed at identifying and flagging any potential accessibility issues.

Finally, different perspectives have been investigated in [23,24] by Aljedaani et al. [23]. They did not employ LLMs for accessibility evaluation but they investigated the accessibility of the code generated by ChatGPT, focusing on its compliance with WCAG. The authors conducted an empirical evaluation involving 88 web developers who asked to ChatGPT to generate websites, which were then analyzed for accessibility violations using tools like AChecker and WAVE. The results revealed that 84% of the generated websites contained accessibility issues related to perceivability of the interface (e.g., text resizing, color contrast). While ChatGPT (GPT-3.5) demonstrated a 70% success rate in fixing violations in its own code and 73% in third-party GitHub projects, it struggled with specific guidelines—particularly those concerning multimedia content, visual design, layout and spacing, multimodal navigation, and alternative input methods. Instead, Acosta-Vargas et al. [24] present a comprehensive evaluation of the accessibility of generative AI applications with a specific focus on inclusivity for people with disabilities. Their study evaluates 50 popular generative AI tools–including ChatGPT, Copilot, Midjourney, and DALL-E 2–through a dual-method approach combining automated analysis using the WAVE tool and expert-led manual reviews, based on WCAG 2.2 criteria. The evaluation targets four core principles of accessibility: perceivability, operability, understandability, and robustness. The research identifies significant accessibility barriers in many generative AI applications, particularly related to contrast errors, lack of screen reader support, and inconsistent navigation.

The main contributions of the related works are summarized in Table 1.

Table 1. Summary of related works.

Authors	Focus Area	Models/Tools	Key Findings
Othman et al. [18]	Automated remediation of WCAG 2.1 violations on websites	WAVE, ChatGPT	ChatGPT used to fix accessibility errors; effectiveness evaluated vs. manual testing. Provided promising results but highlighted the need for refinement.
Duarte et al. [12]	Detection of heading-related barriers	Llama 3.1, GPT-4o, GPT-4o mini	Models show strengths in different tasks (Llama: hierarchy, GPT-4o mini: nuanced names). All struggled with missing headings.
Delnevo et al. [19]	Evaluation and correction of HTML accessibility	ChatGPT 3.5	Focused on forms, tables, images. LLMs corrected many issues (e.g., labels, table semantics) but produced inconsistent attribute fixes.
López-Gil & Pereira [16]	Automating WCAG manual inspection tasks	ChatGPT, Claude, Bard	Evaluated criteria 1.1.1, 2.4.4, 3.1.2. Achieved 87.18% accuracy in detecting issues missed by automated tools.
Huynh et al. [20]	Image accessibility via context-aware captions	SmartCaption AI, LLM + webpage context	Generated captions enriched with page context improved relevance and integration with screen readers.
Aljedaani et al. [21]	Education: integrating accessibility in software engineering	ChatGPT 3.5, WAVE, AChecker	Students remediated violations with LLMs: ∼60% fixed most, ∼30% fixed all. Increased awareness and skills.
Taeb et al. [22]	Accessibility assessment of mobile apps	LLM + pixel-based UI understanding	Produced structured video presentations to flag accessibility issues. Showcased hybrid LLM–vision approach.
Aljedaani et al. [23]	Accessibility of ChatGPT-generated websites	ChatGPT, WAVE, AChecker	Approximately 84% of generated sites had issues (contrast, multimedia, navigation). ChatGPT fixed ∼70% of its own violations. Struggled with complex guidelines.
Acosta-Vargas et al. [24]	Accessibility of generative AI applications	WAVE, manual evaluation	Analyis of 50 popular generative AI tools (e.g., ChatGPT, Copilot, Midjourney, and DALL-E 2). Found widespread issues in perceivability, screen reader support, and navigation consistency across tools.

3. Materials and Methods

This section introduces the research questions that drove this study and describes the dataset used in the various experiments, the prompts, and the LLMs used.

3.1. Research Questions

To systematically evaluate the capabilities of LLMs to identify and assess accessibility issues, we structured our study around two primary research questions:

- RQ1: "How well do LLMs evaluate the accessibility of dynamically generated web content implemented with different JavaScript frameworks (i.e., Vue, React, Angular) and server-side languages (i.e., PHP)?"
- RQ2: "How consistent and reliable are LLM evaluations when systematically assessing accessibility across multiple versions of table components?"

3.2. Dataset Description

To comprehensively evaluate the capabilities of LLMs in assessing web accessibility, we curated a diverse dataset designed to address our two primary RQs.

For RQ1, which examines the LLMs' performance on dynamically generated content, we developed code snippets that create tables using various JavaScript frameworks and the server-side language PHP. These snippets were designed to showcase different approaches to dynamic content generation, allowing us to evaluate the LLMs' adaptability across diverse coding paradigms. We selected Vue [25], Angular [26], and React [27]—three of the most widely used frontend frameworks—alongside PHP, still the dominant server-side language [28]. Vue, React, and Angular share common characteristics, such as a component-based architecture, data-binding mechanisms, and a focus on reusability, making them well-suited for modern web applications [29]. Each framework, however, brings its own strengths: Vue emphasizes simplicity and flexibility, Angular offers a full-featured, TypeScript-based structure with built-in tools for scalability, and React, developed by Facebook, prioritizes performance with its virtual DOM and declarative UI design. PHP, on the other hand, remains a versatile and widely adopted server-side scripting language, enabling dynamic content generation and powering popular CMS platforms like WordPress and Drupal. This diverse selection ensures our dataset reflects a broad range of web development paradigms, from client-side interactivity to server-side content rendering.

To investigate the consistency and reliability of LLM evaluations across component versions (RQ2), we utilized a dataset of 25 distinct implementations of Vue table components. These components, developed by students from the Web Applications and Services course at the University of Bologna, featured variations in accessibility features, API usage (Composition vs. Options), and overall structure. This diverse dataset provided a rich foundation for assessing the LLMs' ability to track and evaluate accessibility changes across iterative component development. While we initially explored accessibility evaluations across Vue, React, Angular, and PHP, a more focused approach was necessary for a systematic evaluation of component variations. Specifically, we required a substantial number of implementations centered on a consistent component type (tables), to ensure a robust comparative analysis. To achieve this, we collaborated with the student developers, whose coursework extensively covered Vue, making it the practical choice for our in-depth study. Furthermore, given the architectural similarities among modern JavaScript frameworks like Vue, React, and Angular, we posit that findings derived from a thorough analysis of Vue components are likely to offer insights applicable across these frameworks. It is reasonable to hypothesize that the general trends and patterns observed in LLM evaluations of Vue components would manifest similarly in React and Angular, particularly concerning common accessibility patterns and challenges. Additionally, focusing on a

single framework enabled us to achieve a higher degree of granularity in our analysis, allowing for a deeper exploration of specific accessibility nuances and their impact on LLM evaluations. This decision also reflected practical considerations, such as the resources and time constraints inherent in conducting a comprehensive analysis across multiple frameworks. By prioritizing depth over breadth in this phase of our research, we aimed to provide a more detailed and actionable understanding of LLM capabilities in assessing web component accessibility.

The dataset has been deeply described in [30]. It is organized into two main directories, each supporting a different dimension of our evaluation. The first directory, Dynamic Generated Content, contains handcrafted snippets in Angular, React, Vue, and PHP. For each technology, two tabular components were implemented: an accessible version, compliant with WCAG guidelines, and an invalid version, intentionally embedding common accessibility violations such as missing ARIA attributes, poor semantic markup, or inadequate keyboard support. Each snippet is accompanied by source code, a `labels.csv` file reporting the violations identified through manual inspection, and automated accessibility reports generated by the Mauve tool on the rendered HTML. This dual perspective enables both code-level and output-level evaluation of accessibility.

The second directory, Vue Table Components, consists of 25 independent Vue projects developed by students. Each project provides four table variants: minimal and accessible implementations created with both the Options API and the Composition API. The projects are based on heterogeneous datasets (minimum 50 rows $\times$ 5 attributes), spanning domains such as music, movies, environmental measurements, and sports. This collection introduces variability in architectural choices (e.g., single-file vs. modular components) and accessibility practices, providing a diverse and realistic benchmark for analyzing how LLMs handle accessibility in real-world development scenarios.

All code used in this study has been reported in a GitHub repository (https://github.com/manuandru/LLM-WebAccessibility.git, accessed on 6 August 2025).

3.3. Prompt Design and Methodology

To ensure a consistent and thorough evaluation of LLM performance across our research questions, we carefully crafted tailored prompts for each scenario. Each prompt aligns with the specific goal of the corresponding research question, ensuring that the model's output remains focused on the accessibility criteria under investigation. To enhance the effectiveness of these prompts, we employed various prompt engineering techniques, including role prompting [31]. This involved instructing the LLM to adopt specific roles, such as an accessibility expert or a code reviewer, to guide its analysis and responses. By defining clear roles and providing precise instructions, we aimed to elicit more accurate and relevant evaluations of the accessibility issues present in our dataset. Figure 1 provides a comprehensive overview of our methodology.

For RQ1, the code snippets of the Dynamic Generated Content folder, containing table components implemented with the four technologies previously mentioned, are used. Their source codes are analyzed using GPT-4o mini and Gemini 2.0 Flash (Gemini) to evaluate the compliance of the generated code with the WCAG 2.2. The raised accessibility issues are then manually evaluated. Legitimate issues were then fixed by following the suggestions proposed by the LLMs and tested again to verify their applicability.

A different pipeline has been followed for RQ2. The data used for the experiments were the 25 implementations of Vue Table Components. Initially, LLMs analyzed them to retrieve WCAG 2.2 AA guideline violations. Three of the authors manually reviewed each violation to classify it as correct, redundant, or hallucination. Then, the same analysis

has been carried out using LLMs, including o3-mini to understand the LLMs' capacity in verifying their own output and a systematic reliability analysis has been carried out.

Below, we describe the design and rationale for each prompt, detailing how these techniques were implemented to optimize the LLM's performance.

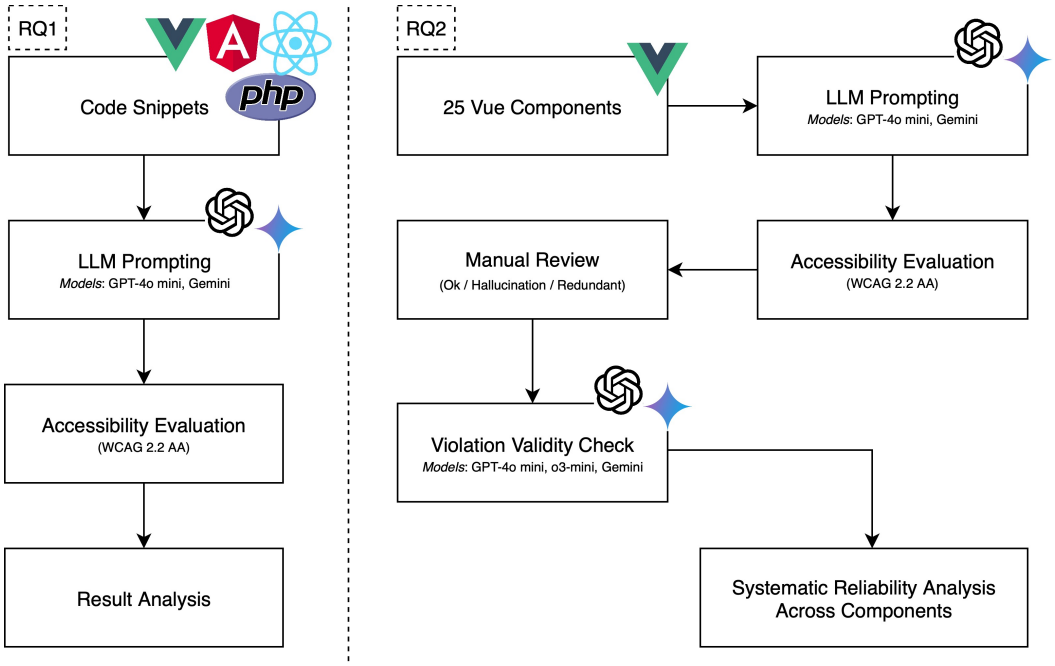

Figure 1. Overview of the methodology employed in the study for evaluating LLM performance on accessibility assessment tasks.

For RQ1, where the focus is on dynamically generated content from various frameworks and server-side languages, we crafted a prompt to evaluate accessibility across different paradigms. This prompt acknowledges the interactive nature of the content and asks the model to assess the resulting output:

> You are a tool for detailed web accessibility validation.
>
> Analyze the following dynamically generated web content.
>
> The content is produced by a code snippet written in one of the following: Vue, React, Angular, or PHP.
>
> Evaluate the resulting output for accessibility issues according to WCAG 2.2 AA guidelines.
>
> For each issue identified, include the following:
> - The affected element or code portion.
> - The corresponding WCAG criterion.
> - A brief explanation of the issue.
> - A recommendation for fixing it.
>
> — Code Snippet —
> $code_snippet$

This design ensures the LLM not only processes the code itself but also infers the resulting structure, focusing on end-user accessibility.

For RQ2, we designed a two-step process to maximize violation detection and identify hallucinations. The prompts support a systematic evaluation of multiple Vue table component implementations, ensuring consistent output formatting. First, a prompt is used to

retrieve WCAG 2.2 AA guideline violations found on individual webpages. The prompt is as follows:

> You are a tool for detailed web accessibility validation.
>
> Is this VUE component accessible according to WCAG 2.2 guidelines (which include previous WCAG 2.0 and 2.1 standards) at the AA level?
>
> Analyze the following VUE files that make up the component.
>
> After the analysis, create a table with the list of WCAG violations you found, indicating the following:
>
> - In the first column, the relative success criterion.
> - In the second column, the part of code analyzed or a summary of it.
> - In the third column, a short description of the problem.
>
> — File: $filename$—
>
> $filecontent$

Then, each violation has been manually evaluated by three reviewers and marked as follows: OK (if it is correct), A (if it is hallucination), and R (if it is redundant).

Second, a more targeted prompt is employed for each identified violation to confirm its validity and distinguish it from a hallucination. This approach, similar to Chain-of-Verification [32], requires the LLM to verify its findings incrementally. By checking each part of its response, the LLM can minimize errors and enhance the accuracy of its results. The prompt for the second step is as follows:

> You are a tool for detailed web accessibility validation.
>
> I ask you to verify the correctness of the following WCAG 2.2 accessibility issue reports (including WCAG 2.1 and 2.0) generated for components of a Vue.js application.
>
> Create a new markdown table about accessibility issues, similar to the one below, by adding 2 columns: Correctness and Description.
>
> In the “Correctness” column, enter the following:
>
> “OK” if the reported issue is actually present on the page.
> “A” if it is a hallucination (an accessibility error reported that does not actually exist).
> “R” if the report is redundant. “Redundant” refers to a suggestion or recommendation provided by the model that, based on the analysis of the rendered HTML code below, does not appear useful or correct.
>
> In the “Description” column, provide a brief justification for your assessment.
>
> Below is the table of reports to verify, followed by the HTML-rendered code of the Vue.js component (rendered using a Node.js server and Chrome browser).
>
> $step 1 result$
>
> $HTML Code Rendered$

This structured output facilitates comparative analysis across the 25 different student-implemented Vue table components, enabling systematic reliability checks.

3.4. Large Language Models

LLMs are self-supervised foundational models that can be prompted or fine-tuned to perform a wide array of natural language processing tasks that once required separate, task-specific systems [33,34]. These models are typically built on the Transformer architecture

and trained on massive corpora, enabling them to generalize across tasks such as text generation, summarization, question answering, and code understanding [35]. Their versatility has led to widespread adoption across diverse fields including healthcare [36], education [37], and software development.

In this study, we employed three LLMs: GPT-4o mini [38], o3-mini [39], and Gemini [40]. These models reflect the current state of scalable instruction-following language models. GPT-4o mini and o3-mini are proprietary models by OpenAI, optimized for performance and efficiency. GPT-4o (short for "omni") is the latest model in the GPT-4 family, designed for multimodal tasks while maintaining strong natural language understanding and generation capabilities. The mini variants used in our experiments are optimized for speed and cost while retaining a high level of linguistic and reasoning competence.

Gemini, developed by Google DeepMind, is a lightweight and fast variant of the Gemini series, following earlier iterations such as LaMDA and PaLM [41,42]. This model is designed to provide rapid responses while maintaining strong instruction-following behavior, making it particularly suitable for real-time evaluation tasks like those required in our accessibility assessment scenarios.

All models were used in zero-shot inference mode via prompting, without any additional fine-tuning. We compared their outputs in terms of consistency, completeness, and alignment with WCAG 2.2 standards. Leveraging multiple LLMs allowed us to examine not only overall performance but also the variation in accessibility evaluation across different architectures and training strategies.

During the preparation of this manuscript, the authors used ChatGPT (GPT-4o) for the purposes of writing assistance. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

4. Results and Discussion

The answers to the two Research Questions are discussed in isolation in the following subsections.

4.1. Validation of Dynamic Generation of HTML Code

In this set of experiments, the evaluation of the accessibility is conducted on the code that generates dynamically the HTML code, taking advantage of frontend JavaScript frameworks (i.e., Vue, Angular, and React) and PHP. The experiments are all focused on the generation of the same HTML table.

4.1.1. Accessible Components

The first set of experiments conducted were focused on components that should produce accessible tables. Table 2 presents the accessibility issues identified by the GPT-4o mini model across dynamically generated table components developed in Angular, React, Vue, and PHP. Although the components were intended to produce accessible tables, several recurring accessibility problems were systematically detected, highlighting common pitfalls even in seemingly well-designed dynamic content.

Four accessibility issues were found. Components in Angular, React, and Vue potentially generate non-unique dynamic IDs in case of songs with the same title, a violation of WCAG 1.3.1 that can disrupt assistive technologies. The recommended solution is to add a unique suffix, like an index, to each ID. Across all frameworks, a lack of descriptive summaries for tables was detected, which impacts WCAG 1.3.1 and 1.1.1 by preventing screen readers from understanding the table's purpose. The LLM suggested using the `aria-describedby` attribute to link to a hidden description. Another gap found in Angular,

React, and Vue components was the absence of loading messages during dynamic content updates. This violates WCAG 4.1.3 and can confuse users of assistive technologies. The proposed fix is to use an aria-live region to announce "Loading..." messages. Finally, a consistent issue across all evaluated technologies was the lack of error messaging under failure conditions, which infringes on WCAG 3.3.1. Without this feedback, users struggle to identify and correct issues. The suggested remedy is to display accessible error messages using ARIA roles like `role = ''alert''` for immediate notification.

Table 2. Accessibility issues with the relative Web Content Accessibility Guideline (WCAG) and recommended fixes for accessible table components in Angular, React, Vue, and PHP as identified by GPT-4o mini.

Components	Issue	WCAG	Fix
Angular, React, Vue	Non-unique dynamic IDs	1.3.1	Add unique suffix like index
All	Missing descriptive summary	1.3.1/1.1.1	Use aria-describedby with hidden description
Angular, React, Vue	No loading message	4.1.3	Use aria-live region with "Loading..."
All	No error message on failure	3.3.1	Display accessible error message with role = "alert"

Then, we evaluate the same components using Gemini. The results are summarized in Table 3. Despite the developers' intention to produce accessible tables, several accessibility deficiencies were highlighted, common to all technologies assessed.

Table 3. Accessibility issues with the relative WCAG and recommended fixes for accessible table components in Angular, React, Vue, and PHP as identified by Gemini.

Components	Issue	WCAG	Fix
All	Incorrect use of <th> within the <tbody>	1.3.1	Change the <th> element within the <tbody> to a <td> element
All	Potentially insufficient contrast for text content	1.4.3	Ensure that the CSS applied to the table cells provides a sufficient contrast ratio
All	Redundant and potentially confusing use of [attr.headers]	1.3.1	Modify the [attr.headers] attribute in the <td> elements to directly reference the id of the appropriate column header
All	Missing summary attribute on the <table> element	1.3.1	Consider adding a summary attribute to the <table> element

Gemini also identified four accessibility issues. The model flagged the use of `<th>` elements within `<tbody>` sections as a violation of WCAG 1.3.1. However, this is a valid practice for creating row headers in complex tables, highlighting a limitation in the model's understanding of advanced HTML semantics. A possible legitimate concern identified was insufficient contrast for text content, which violates WCAG 1.4.3. Anyway, no information was provided about the style of the page. Thus, it is not clear which information was evaluated to detect this issue. Gemini also detected redundant or confusing use of the headers attribute in `<td>` elements. While the suggested fix is to simplify and correctly link to column headers, the headers' attributes of `<td>` were correctly linked to the correspond-

ing table heading IDs. Lastly, the model flagged the absence of the summary attribute on <table> elements, even if this attribute is obsolete in HTML5. To summarize, among the four violations, three of them were false positives, while one was raised without the appropriate information.

To test applicability, the fixes suggested by GPT-4o mini were implemented in the original code and re-evaluated. Gemini suggestions were not implemented because they were deemed incorrect or irrelevant. Re-testing with GPT-4o mini, using the same prompt, confirmed the following:

- Non-unique ID errors disappeared once unique suffixes, based on the relative index, were added.
- Summaries linked with `aria-describedby` were correctly recognized as compliant.
- The introduction of an `aria-live` region eliminated the absence-of-loading-message error.
- Accessible error feedback using `role = ''alert''` was correctly identified as a valid remediation.

4.1.2. Non-Accessible Components

After the evaluation of components that should produce accessible tables, we moved to components that produce non-accessible tables. In particular, even if table cells have both row and column headings, neither the scope attribute nor the headers attribute were employed.

Table 4 reports the accessibility issues detected by GPT-4o mini in the table components designed to be non-accessible, implemented with Angular, React, Vue, and PHP. As expected, the number and severity of detected violations increased compared to the components intended to be accessible, confirming the tool's ability to identify meaningful differences in accessibility compliance.

Table 4. Accessibility issues with the relative WCAG and recommended fixes for non-accessible table components in Angular, React, Vue, and PHP as identified by GPT-4o mini.

Components	Issue	WCAG	Fix
All	Missing scope = "col" on headers	1.3.1	Add scope = "col" to column headers
All	Missing scope = "row" on row headers	1.3.1	Add scope = "row" to dynamic title headers
Angular, React, Vue	No loading message	4.1.3	Use aria-live region with "Loading…"
All	No error message on failure	3.3.1	Display accessible error message with role = "alert"
All	No detailed description of table	1.3.1	Add aria-describedby linked to visually hidden content

Two critical issues emerged regarding the semantic structure of tables:

- Missing `scope = ''col''` on column headers: According to WCAG 1.3.1 (Info and Relationships), properly setting the scope attribute in `<th>` elements is crucial to help assistive technologies interpret the relationship between headers and their corresponding cells.
- Missing `scope = ''row''` on row headers: Similarly, for row headers, the absence of `scope = ''row''` further degrades the semantic clarity of the tables. Without these

markers, navigating large tables becomes significantly more difficult for users relying on assistive technologies.

These issues were not present in the accessible versions, highlighting that proper use of the `headers` attribute was effective. It is interesting to notice that GPT-4o mini did not propose the use of the `headers` attribute in combination with the ID of table headers; rather, it preferred to recommend the use of the scope attribute.

The other issues were also detected in the accessible components (i.e., no loading message, no error message on failure, and no detailed description of the table) and were highlighted again, suggesting a consistency of evaluation among the various components.

With regard to the evaluation conducted with Gemini, Table 5 presents the accessibility issues identified in the intentionally non-accessible table components built with Angular, React, Vue, and PHP. In a manner parallel to the findings observed with o3-mini on these same components, beyond the issues previously identified in the accessible versions of these components, Gemini additionally flagged the absence of both `scope` and `id` attributes within the table header cells (`<th>`) located in the <head>. It is important to highlight that the use of the `scope` attribute is enough to ensure the accessibility of the table, and it is not required to be coupled with the `id` attribute.

Table 5. Accessibility issues with the relative WCAG and recommended fixes for non-accessible table components in Angular, React, Vue, and PHP as identified by Gemini.

Components	Issue	WCAG	Fix
All	Missing scope and id attributes for table header cells (<th>) in the <thead>.	1.3.1	To properly associate the header cells with the data cells in each column, the <th> elements in the <thead> should have a scope = “col” attribute and a unique id attribute.
All	Incorrect use of <th> within the <tbody>	1.3.1	Change the <th> element within the <tbody> to a <td> element
All	Potentially insufficient contrast for text content	1.4.3	Ensure that the CSS applied to the table cells provides a sufficient contrast ratio.
All	Missing summary attribute on the <table> element	1.3.1	Consider adding a summary attribute to the <table> element.
All	No detailed description of table	1.3.1	Add aria-describedby linked to visually hidden content.

Again, parallel to what happened for the accessible components to verify the applicability of the recommendations, we implemented the correct suggestions provided in the original code and re-evaluated them with both LLMs with the same prompt. Similarly, all the previously highlighted issues have not been detected, confirming that both the use of the `scope` and `headers` attributes are correctly detected as mechanisms that ensure the accessibility of the table.

4.1.3. Cross-Framework and Language Accessibility Evaluation by LLMs

To answer the RQ1, the results demonstrate that LLM-based models, specifically GPT-4o mini and Gemini, are capable of detecting a wide range of accessibility issues in dynamically generated web components across different JavaScript frameworks (Vue,

React, Angular) and PHP. However, their performance also reveals distinct strengths and limitations.

Overall, both models consistently identified critical accessibility problems affecting dynamically generated tables. Improper scope and headers management for table elements were systematically detected across all frameworks, indicating that LLMs generalize well across different frontend and backend technologies.

When evaluating components intended to be accessible, both GPT-4o mini and Gemini successfully pinpointed subtle but important shortcomings. GPT-4o mini focused more on semantic and interaction aspects (e.g., dynamic ID generation, live updates, error reporting), while Gemini highlighted additional presentation issues, like color contrast problems. However, Gemini also exhibited a notable false positive (incorrectly reporting the use of <th> in <tbody> as a violation), revealing that its knowledge about valid complex table semantics was not fully aligned with modern HTML standards.

In evaluating non-accessible components, both models correctly detected the missing use of scope and headers attributes. This confirms that the models are sensitive enough to distinguish between different levels of accessibility compliance and can detect both superficial and deep structural accessibility flaws.

In summary, LLMs such as o3-mini and Gemini can effectively evaluate the accessibility of dynamically generated web content across different frameworks and languages, identifying both high-level and detailed WCAG violations. However, occasional inaccuracies and outdated suggestions imply that their outputs should be critically reviewed by accessibility experts before being fully trusted in critical evaluation contexts.

4.2. Vue Table Components Analysis

4.2.1. Initial Accessibility Violation Detection

In the initial phase of our evaluation, we employed the first prompt detailed in Section 3.3 to analyze accessibility issues using both GPT-4o mini and Gemini, subsequently conducting a manual expert review of the detected issues.

The findings for GPT-4o mini and Gemini are respectively summarized in Tables 6 and 7. GPT-4o mini detected a total of 512 violations across various WCAG criteria. Of these, 103 issues (20.1%) were judged fully correct (OK), while 358 issues (69.9%) were classified as hallucinations (A), meaning that although the model flagged real accessibility concerns, the details or severity were partially incorrect or incomplete. Only a small fraction, 51 issues (10.0%), were classified as redundant (R), indicating false positives where no genuine accessibility problem existed.

Conversely, the results for Gemini showed a different profile. A total of 384 violations were detected, with 113 issues (29.5%) deemed fully correct (OK) and 77 issues (20.1%) classified as hallucinations (A). However, a significant number—193 issues (50.4%)—were considered redundant (R), revealing a greater tendency of Gemini to flag accessibility violations that, upon manual inspection, were found to be incorrect or not impactful.

In both cases, Criterion 1.3.1 (Info and Relationships) yielded the highest number of detections (167 and 189 cases for GPT-4o mini and Gemini, respectively). However, the classification differed notably between the two models: For GPT-4o mini, the majority of these detections were marked as hallucinations, suggesting frequent but only partially accurate identification of semantic issues. For Gemini, a significant portion under Criterion 1.3.1 was instead classified as redundant, indicating a high rate of false positives related to the interpretation of semantic relationships.

It is also noteworthy that Gemini achieved a higher percentage of fully correct detections (29.5%) compared to GPT-4o mini (20.1%), suggesting better precision in those cases where the model correctly identified an accessibility problem. Furthermore, Gemini

demonstrated a considerably lower proportion of hallucinations (20.1% vs. 69.9%), meaning that when it raised issues, they were more often either entirely correct or entirely wrong, with fewer partially accurate findings.

Nevertheless, this advantage came at the cost of a much higher redundancy rate (50.4% vs. 10.0%), indicating that Gemini flagged many issues that were not actually accessibility violations, which could potentially overwhelm developers with unnecessary corrective actions and reduce the practical efficiency of its assessments.

In summary, these results highlight a fundamental trade-off between the two LLMs. While GPT-4o mini tends to identify a broader set of potential accessibility concerns, its inclination towards generating a high number of reports requiring human verification and interpretation represents a significant limitation. Conversely, Gemini demonstrates slightly better precision in correct detections, but this quality is counterbalanced by a considerable risk of over-reporting and potentially hallucinations, undermining the overall effectiveness of the system.

Table 6. Accessibility violations identified by GPT-4o mini grouped by criterion with the corresponding manual classification (OK: correct issue; A: hallucination; R: redundant recommendation).

Criterion	OK	A	R	COUNT
1.3.1	39	120	8	167
4.1.2	7	43	7	57
1.3.2	8	45	0	53
2.4.6	3	28	9	40
2.4.4	6	24	6	36
1.4.3	7	20	3	30
1.1.1	3	16	4	23
2.4.7	0	16	3	19
3.3.2	4	9	4	17
2.4.3	3	10	1	14
4.1.3	9	4	0	13
2.1.1	1	11	0	12
4.1.1	0	5	2	7
3.3.1	6	0	0	6
3.3.3	5	0	0	5
1.4.5	0	1	3	4
1.4.11	0	3	0	3
2.5.3	0	2	0	2
1.3.3	1	1	0	2
3.1.1	0	0	1	1
3.2.2	1	0	0	1
Total	103	358	51	512

Table 7. Accessibility violations identified by Gemini grouped by criterion with the corresponding manual classification (OK: correct issue; A: hallucination; R: redundant recommendation).

Criterion	OK	A	R	COUNT
1.3.1	67	40	82	189
2.4.7	4	6	48	58
1.1.1	12	9	19	40
1.4.11	10	1	16	27
1.4.1	5	8	12	25
4.1.2	7	11	4	22
2.4.6	3	1	1	5
4.1.1	2	1	2	5
1.4.3	0	0	4	4
2.4.4	1	0	3	4
2.4.3	2	0	1	3
3.2.1	0	0	1	1
Total	113	77	193	383

4.2.2. Consistency and Reliability of LLM Accessibility Evaluations

Following the initial detection of accessibility violations, described in the previous subsection, a second verification phase was introduced to validate the correctness of each issue. This step employed a targeted prompt inspired by the Chain-of-Verification approach [32], where each model was required to reassess its own outputs by incrementally analyzing its reported violations in the context of the rendered HTML source code. This method allowed for a more controlled inspection of each LLM's self-consistency and factual grounding.

The results of this second step is summarized in Tables 8 and 9. GPT-4o mini demonstrated limited reliability, achieving an overall accuracy of 34%. While it performed well in recognizing actual accessibility issues ($\text{Recall}_{OK} = 0.83$), its low Precision for that class (0.23) indicates a high false-positive rate. Notably, it exhibited poor consistency in identifying hallucinated issues ($\text{Recall}_{A} = 0.2$), meaning it frequently failed to recognize when a previously reported hallucination was invalid. Its balanced performance in detecting redundant issues ($\text{F1}_{R} = 0.37$) was moderate, but not particularly strong.

Table 8. Performance metrics for GPT 4-o mini's classification of previous output violations (OK: correct issue; A: hallucination; R: redundant recommendation).

Class	Precision	Recall	F1-Score	Support	Accuracy
A	0.8	0.2	0.31	358	0.34
OK	0.23	0.83	0.36	103	
R	0.37	0.37	0.37	51	

The relative confusion matrix, depicted in Figure 2, reveals how the GPT-4o mini model performed in classifying violations as Hallucinations, OK, or Redundant. It shows the counts of correct and incorrect predictions for each category. Notably, the model struggled most with accurately identifying Hallucinations, often misclassifying them as OK. The OK category also saw some misclassifications, while the Redundant category had a relatively lower number of correct predictions compared to its errors. Overall, the matrix

highlights the specific types of classification mistakes the model made across the different violation categories.

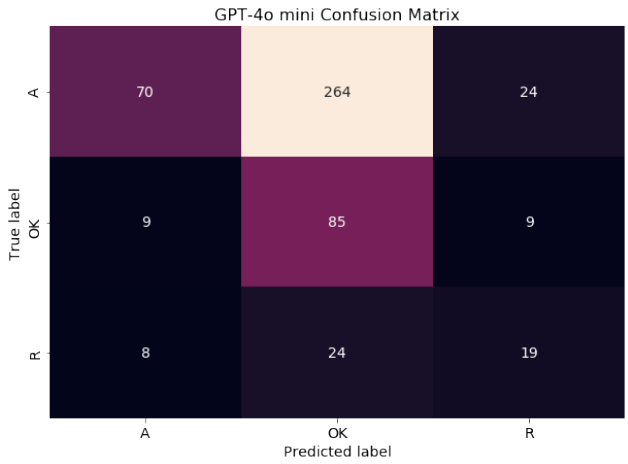

Figure 2. Confusion Matrix of GPT 4-o mini's classification of previous output violations (OK: correct issue; A: hallucination; R: redundant recommendation).

Gemini yielded an even lower overall accuracy of 30%, and its classification reliability was markedly inconsistent across classes. It achieved the highest recall for correct issues (Recall_{OK} = 0.88), suggesting strong sensitivity to real violations. However, the precision for this class was modest (0.29), again reflecting a substantial number of false positives. The model performed especially poorly in identifying redundant outputs (Recall_{R} = 0.02), indicating it was largely unable to recognize when its own suggestions were unnecessary or irrelevant.

Table 9. Performance metrics for Gemini's classification of previous output violations (OK: correct issue; A: hallucination; R: redundant recommendation).

Class	Precision	Recall	F1-Score	Support	Accuracy
A	0.32	0.14	0.2	77	0.3
OK	0.29	0.88	0.44	113	
R	0.4	0.02	0.04	193	

The confusion matrix (Figure 3) for Gemini illustrates its performance in classifying violations into Hallucinations (A), OK, and Redundant (R) categories. Looking at the matrix, we can see how many instances of each true category were correctly and incorrectly predicted by the model. For Hallucinations, while some were correctly identified, a larger number were misclassified as OK. The OK category shows a high number of correct classifications, but also some instances incorrectly labeled as Hallucinations and Redundant. Finally, for the Redundant category, there were a few correct predictions, but a significant number were wrongly classified as OK. This matrix provides a clear picture of the specific types of errors Gemini made across the different violation types.

Finally, we conducted a further test using a more powerful model, O3-mini, which analyzed the violations raised by GPT 4o-mini. Results are reported in Table 10. In contrast to the previous models, O3-mini exhibited significantly more consistent and reliable behavior, achieving an overall accuracy of 79%. It demonstrated strong performance across all classes, particularly in detecting hallucinations (F1_{A} = 0.85) and correctly identifying redundant suggestions (F1_{R} = 0.61). Unlike the other two models, O3-mini balanced both precision and recall effectively, resulting in a respectable F1-score of 0.67 for actual accessibility violations.

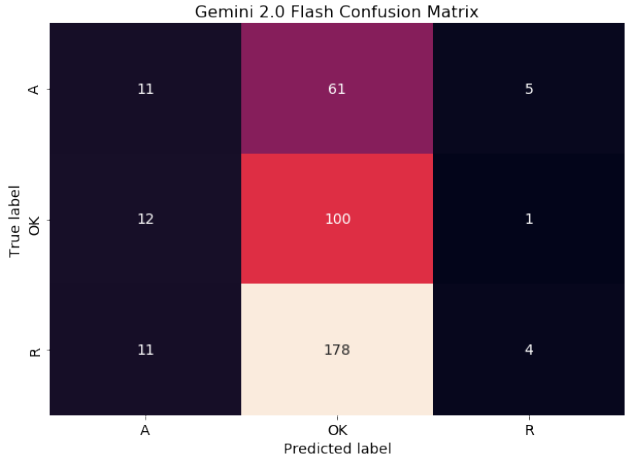

Figure 3. Confusion matrix of Gemini's classification of previous output violations (OK: correct issue; A: hallucination; R: redundant recommendation).

Table 10. Performance metrics for O3-mini's classification of previous output violations (OK: correct issue; A: hallucination; R: redundant recommendation) identified by GPT-4o mini.

Class	Precision	Recall	F1-Score	Support	Accuracy
A	0.86	0.84	0.85	358	0.79
OK	0.66	0.67	0.67	103	
R	0.56	0.67	0.61	51	

The confusion matrix for O3-mini, depicted in Figure 4, which analyzed violations identified by GPT 4o-mini, reveals a strong performance in classifying these violations. The model correctly identified a large number of Hallucinations (A), with fewer being misclassified as Ok or Redundant. For truly OK violations, O3-mini showed a good level of accuracy, with a smaller number being incorrectly labeled as Hallucinations. The Redundant (R) category also saw a reasonable number of correct classifications, although some were misidentified as Hallucinations. Overall, the matrix indicates that O3-mini demonstrates a better ability to distinguish between these violation types compared to the previous models, aligning with its reported higher accuracy and balanced precision–recall.

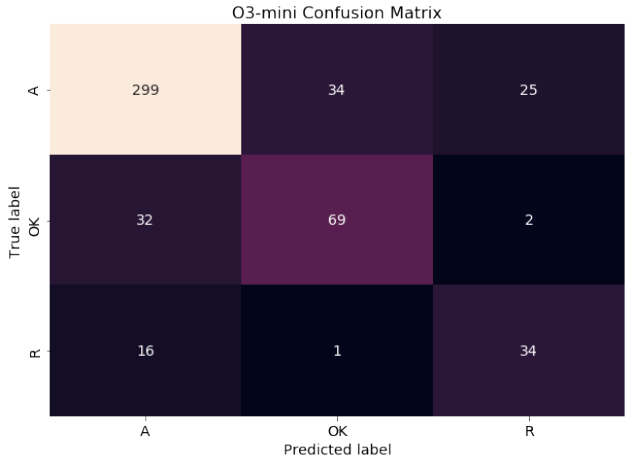

Figure 4. Confusion matrix of O3-mini's classification of previous output violations (OK: correct issue; A: hallucination; R: redundant recommendation) identified by GPT-4o mini.

4.3. Consistency and Reliability of LLMs Evaluations

These findings underscore that while GPT-4o mini and Gemini are capable of generating plausible accessibility reports even suffering from hallucinations and redundant violations, they suffer from low self-verification reliability, particularly in distinguishing between valid, redundant, and hallucinated issues. Their limited ability to reassess their own outputs poses risks for over-reporting or misguiding developers in real-world auditing scenarios. By contrast, O3-mini appears to offer a more dependable approach to systematic accessibility evaluation, especially when applied in multi-stage workflows where validation and refinement are essential.

Addressing RQ2 of how consistent and reliable LLM evaluations are when systematically assessing accessibility across multiple versions of table components, the results of our second evaluation phase reveal significant inconsistencies and varying levels of reliability among different LLMs. The stark contrast observed, with O3-mini demonstrating high accuracy while GPT-4o mini and Gemini exhibited relatively poor performance in verifying accessibility violations, underscores the critical need for robust internal validation within LLMs used for such tasks. Although initial outputs from all models appeared plausible, only O3-mini consistently and reliably distinguished between actual issues, hallucinations, and redundant suggestions during the self-verification process. This finding indicates that the initial output quality of an LLM is not a sufficient measure of its practical reliability for accessibility evaluations; a systematic self-verification step is crucial for ensuring robustness. The inconsistency observed, where over-reporting can lead to wasted effort and under-reporting to non-compliance, poses a serious challenge for real-world applications. Consequently, these findings strongly support the necessity of multi-step LLM evaluation pipelines and prompt further investigation into how architectural differences or training objectives influence a model's ability to reason reliably when performing systematic accessibility assessments across table component versions.

5. Discussion and Limitations

This section discusses the empirical findings of our study, situating them within three complementary perspectives: first, we outline the strengths and limitations of LLM-based accessibility evaluation observed in our experiments; second, we compare these results with related work to highlight similarities, differences, and unique contributions; and third, we reflect on the broader capabilities and risks of applying LLMs in accessibility auditing workflows. Finally, the limitations of this study are acknowledged.

5.1. Strengths and Limitations of LLM-Based Accessibility Evaluation

The results of this study demonstrate both the promise and current limitations of using LLMs for automated accessibility evaluations of dynamically generated web content. In addressing RQ1, the evaluation shows that both GPT-4o mini and Gemini are capable of identifying a broad spectrum of accessibility violations across multiple frontend frameworks and backend technologies. These models consistently surfaced key issues such as missing scope attributes, redundant or incorrect use of headers, and absent dynamic feedback mechanisms—violations that are crucial to ensuring WCAG compliance. However, despite their capacity to detect many legitimate issues, both models also produced a substantial number of hallucinated or redundant violations. These inaccuracies suggest a knowledge gap or misalignment with current HTML5 semantics. Additionally, the reliance on deprecated attributes like summary points to a tendency in some models to prioritize outdated best practices, potentially introducing confusion or unnecessary remediation work for developers.

In addressing RQ2, the evaluation revealed significant disparities in model reliability when subjected to a verification phase using the Chain-of-Verification-inspired prompt. Here, both GPT-4o mini and Gemini displayed weak self-consistency, with low overall accuracy (34% and 30%, respectively), high false-positive rates, and limited ability to correctly reassess hallucinated or redundant violations. The performance gaps between detection and verification stages underscore a fundamental limitation in their reasoning stability and factual grounding, which is critical for trustworthy automated evaluations. By contrast, the O3-mini model, when used in a second-stage validation role, demonstrated markedly superior consistency and judgment, achieving a high overall accuracy of 79%. It excelled not only in distinguishing valid issues from hallucinations but also in recognizing whether previously flagged items were either spurious or redundant. O3-mini's superior performance stems from its specialized architecture for logical reasoning [39]. Unlike generalist models that are optimized for a wide range of tasks, O3-mini is built for a deliberate, multi-step "chain of thought" process. This approach allows it to systematically analyze problems, internally verify its own logic, and reduce hallucinations and factual errors. Key to this is its ability to adjust a "reasoning effort" level, enabling a more thorough, internal analysis on complex tasks. This makes it a more reliable "thinker" for precision-critical jobs like code verification, where correctly distinguishing between valid and invalid code is paramount. This finding suggests that a tiered evaluation pipeline—where a high-recall model conducts the initial pass and a more conservative, high-precision model validates those findings—could be a viable and practical strategy for integrating LLMs into real-world accessibility audits.

5.2. Comparison with Related Works

Our findings are consistent with, yet distinct from, prior research on the use of LLMs for accessibility (see Table 1). Like Othman et al. [18] and Delnevo et al. [19], we observed that LLMs can correctly identify and remediate many accessibility issues in HTML code, but often with inconsistent or redundant attribute usage. Similarly, Duarte et al. [12] and López-Gil & Pereira [16] highlighted that different models specialize in different sub-tasks but struggle with certain systematic gaps (e.g., missing headings or nuanced guideline interpretation). Our study extends these observations by showing that such inconsistencies persist when LLMs operate on dynamically generated code across frameworks, and that post hoc verification with a reasoning-specialized model like O3-mini can mitigate these weaknesses.

Other works, such as Huynh et al. [20] and Taeb et al. [22], emphasize context-aware or multimodal accessibility support. By contrast, our contribution lies in demonstrating that LLMs can also make inferences about dynamic runtime behaviors (e.g., asynchronous data loading, error handling), a dimension typically invisible to static validation tools. This expands the scope of AI-assisted accessibility evaluation beyond what has been addressed in prior HTML- or UI-focused approaches. Moreover, while Aljedaani et al. [21,23] and Acosta-Vargas et al. [24] stressed educational value and limitations of generative AI outputs, our study highlights the feasibility of integrating LLMs within developer workflows (e.g., CI/CD pipelines) through multi-stage verification pipelines.

In sum, while the literature consistently points to both the opportunities and risks of relying on LLMs for accessibility tasks, our work focuses on dynamic code-level reasoning, and the benefits of pairing high-recall models with high-precision verifiers. These findings contribute to bridging the gap between prior single-shot evaluations of HTML fragments and a systematic, pipeline-based approach to accessibility validation.

5.3. Unique Capabilities and Challenges

One of the most interesting results obtained is that LLMs can make some considerations about web accessibility based on the dynamic behavior of the elements. In the experiments, LLMs raised accessibility concerns regarding asynchronous data loading and error handling. Such issues can not be detected by standard validation tools. Such a capability has the potential to upend the validation process. Before, when using server-side languages or JavaScript frameworks, the validation process was to be carried out on the generated HTML code for each dynamic page. Taking advantage of LLMs, the validation process can be continuously carried out while developing the code base. The cross-framework consistency of detection also suggests that LLMs are sufficiently abstracted from framework-specific syntax to generalize accessibility reasoning across languages like PHP and JavaScript-based frontend libraries. This opens opportunities for scalable accessibility checks in continuous integration pipelines regardless of the underlying stack.

On the negative side, despite their potential benefits, the experiments conducted illustrate that utilizing LLMs without human oversight poses challenges. The inconsistencies, especially in hallucination management and redundant suggestion identification, underscore the risks of deploying LLMs for automated audits without human oversight. Over-reporting can lead to audit fatigue or unnecessary fixes, while under-reporting critical issues risks poor user experience for people with disabilities. Then, the content generated by LLMs may perpetuate biases and inaccuracies inherent in their training data, as highlighted in previous studies [43]. This raises concerns about over-reliance on LLM output, leading to potential neglect of manual review and correction [44]. Furthermore, in many jurisdictions, compliance with accessibility standards is a legal obligation, and web developers ultimately bear the responsibility for ensuring adherence to these standards [45]. The results advocate for future research focused on refining LLM behavior through prompt engineering, feedback-loop training, or architectural improvements to promote better fact-grounding and domain alignment.

Ultimately, while LLMs are well-positioned to assist in accessibility evaluation tasks, their integration must be framed within a structured, multi-step validation pipeline, ideally guided by expert reviews and aligned with modern web standards. Further investigation into model behavior on domain-specific knowledge, semantic parsing of complex HTML structures, and evolution-aware guideline interpretation (e.g., accounting for HTML5 updates) will be critical to advancing the dependability of LLMs in this space.

5.4. Limitations

It is important to acknowledge several limitations inherent in our current study. First, it is important to note that while the implemented components in our experiments exhibited significant diversity in their internal structure, they were limited in scope, primarily generating tables rather than fully functional websites. It is worth considering, however, that validating table generation, as performed by the components in our dataset, represents one of the more complex aspects of accessibility validation. We chose this focus deliberately, as it allowed us to isolate and study a critical yet difficult accessibility task. However, extending the evaluation to complete websites was beyond the scope of this work due to the scale and variability involved, and remains a promising avenue for future research.

In addition, the validation of individual components does not guarantee the accessibility of the overall HTML page. Indeed, various accessibility issues can emerge when integrating these components into a complete webpage. This limitation stems from the fact that accessibility issues often emerge from interactions between components when integrated into a larger system. While our component-level approach enables a controlled and systematic analysis, it inevitably overlooks emergent accessibility barriers at the page

level. Overcoming this limitation would require a comprehensive end-to-end evaluation framework that integrates both component and system perspectives, which is an important direction for future extensions of this work.

Our systematic investigation focused primarily on components built with Vue. This decision was motivated by the need to constrain experimental variability and ensure methodological rigor. We hypothesized that the behavior observed would not differ significantly across other frameworks (such as React or Angular) or languages (like PHP). However, empirically verifying this across multiple ecosystems would require a significantly larger dataset and infrastructure. Such an undertaking, while important, was beyond the practical scope of this study.

Furthermore, the handling of images warrants separate and more in-depth consideration. The generation of effective alternative text is highly context-dependent, varying based on the specific role an image plays within a webpage. This limitation reflects the broader difficulty of capturing contextual information in automated approaches. Addressing it would require multimodal reasoning that incorporates both visual and textual cues.

Finally, our study did not comprehensively address accessibility issues related to visual design such as accommodating color blindness and ensuring sufficient color contrast. This omission arises because these aspects depend not only on HTML but also on CSS and the final rendered output of the page. A multimodal pipeline capable of analyzing both code and rendered output would be required to overcome this limitation. While beyond the current scope, this remains a crucial step toward holistic accessibility validation.

The limitations identified in this work stem from deliberate methodological choices to ensure tractability, focus, and rigor. They also highlight directions where future work can extend and complement our contributions, particularly through multimodal analysis, broader framework coverage, and system-level validation.

6. Conclusions and Future Works

As the Web continues to evolve into a critical infrastructure for modern society, ensuring its inclusivity, equity, and sustainability becomes not only a technical challenge but also a moral imperative. This paper has explored the emerging role of LLM in supporting web developers toward more accessible and human-centered web design.

Our experiments yielded several key findings. First, in addressing RQ1, we demonstrated that state-of-the-art LLMs such as GPT-4o mini and Gemini can identify a wide range of accessibility violations across multiple frontend and backend technologies. The models consistently flagged issues such as missing scope attributes, redundant headers, and absent feedback mechanisms, revealing their potential as real-time accessibility collaborators during development. Importantly, they also surfaced accessibility concerns tied to dynamic behavior (e.g., asynchronous data loading, error handling), which are not detectable by conventional static validation tools. This highlights a novel capability of LLMs to reason beyond static code and into runtime considerations, paving the way for scalable integration into continuous integration pipelines.

Second, in addressing RQ2, we observed clear disparities between detection and verification performance. While both GPT-4o mini and Gemini showed broad coverage in detecting accessibility barriers, they struggled to maintain factual grounding and self-consistency during a verification phase, achieving relatively low accuracies (34% and 30%, respectively). By contrast, the O3-mini model excelled as a second-stage validator, achieving 79% accuracy in distinguishing genuine issues from hallucinations and redundant suggestions. This result underscores the importance of adopting a tiered pipeline where high-recall models perform initial detection, followed by high-precision models that refine and validate results before integration into developer workflows.

Together, these findings highlight both the promise and limitations of current LLMs. On the one hand, they can democratize accessibility knowledge by lowering the barriers for developers to detect violations early, reasoning across frameworks, and identifying issues missed by automated tools. On the other hand, their tendency to hallucinate violations, propagate outdated practices, and show verification inconsistencies underscores the risks of uncritical adoption. Human oversight, critical reflection, and adherence to evolving accessibility standards remain indispensable to ensure trustworthy results.

This paper paves the way for several future works. More LLMs should be included in the evaluation to have a broader understanding of how such models can be employed for large-scale accessibility testing. Some experiments could be focused on the fine-tuning of LLMs with the perspective of web accessibility to make the model understand some very specific and advanced aspects of it. This would also help in distinguishing valid practices from genuine violations, thereby reducing hallucinations and false positives. Finally, this approach allows the model to be continuously updated with the latest WCAG guidelines and evolving web technologies, ensuring it remains a current and highly effective tool for accessibility validation. Another promising research direction involves leveraging LLMs to generate concise and comprehensible summaries of web content, enhancing accessibility for individuals with cognitive disabilities or those seeking shorter, more digestible information. Furthermore, integrating a conversational agent into the website could further bridge the accessibility gap. Users with disabilities could interact with this agent directly, bypassing the need to navigate the website's interface to locate specific information.

Author Contributions: Conceptualization, G.D. and P.S.; methodology, B.B.; software, B.B.; validation, M.A., B.B., and G.D.; data curation, M.A.; writing—original draft preparation, M.A. and B.B.; writing—review and editing, G.D. and P.S.; visualization, M.A.; supervision, P.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available in Zenodo at https://doi.org/10.5281/zenodo.17062188 (accessed on 5 September 2025).

Acknowledgments: We thank Giulia Bonifazi for her precious support and all the students who developed a version of the Vue components. During the preparation of this manuscript/study, the authors used ChatGPT (GPT-4o) for the purposes of writing assistance. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LLM	Large Language Model
SDGs	Sustainable Development Goals

References

1. Hall, W.; Tiropanis, T. Web evolution and Web Science. *Comput. Netw.* **2012**, *56*, 3859–3865. [CrossRef]
2. Nallasivam, A.; KN, A.S.I.; Desai, K.; Kautish, S.; Ghoshal, S. Universal Access to Internet and Sustainability: Achieving the UN's Sustainable Development Goals. In *Digital Technologies to Implement the UN Sustainable Development Goals*; Springer Nature: Cham, Switzerland, 2024; pp. 235–255. [CrossRef]
3. Rathor, S.; Zhang, M.; Im, T. Web 3.0 and Sustainability: Challenges and Research Opportunities. *Sustainability* **2023**, *15*, 15126. [CrossRef]

4. Huda, M. Empowering application strategy in the technology adoption: Insights from professional and ethical engagement. *J. Sci. Technol. Policy Manag.* **2019**, *10*, 172–192. [CrossRef]
5. Bhandari, M.P. What is next for the sustainable development goals, what are the challenges concerning SDG 10—Reduced inequalities? *Sustain. Earth Rev.* **2024**, *7*, 23. [CrossRef]
6. Warschauer, M.; Matuchniak, T. New Technology and Digital Worlds: Analyzing Evidence of Equity in Access, Use, and Outcomes. *Rev. Res. Educ.* **2010**, *34*, 179–225. [CrossRef]
7. Leuthold, S.; Bargas-Avila, J.A.; Opwis, K. Beyond web content accessibility guidelines: Design of enhanced text user interfaces for blind internet users. *Int. J. Hum.-Comput. Stud.* **2008**, *66*, 257–270. [CrossRef]
8. Ara, J.; Sik-Lanyi, C.; Kelemen, A. Accessibility engineering in web evaluation process: A systematic literature review. *Univers. Access Inf. Soc.* **2023**, *23*, 653–686. [CrossRef] [PubMed]
9. Chadli, F.E.; Moumen, A.; Gretete, D. Factors & Theoretical Models Related to Web Accessibility Integration: A Systematic Literature Review. In Proceedings of the 2025 International Conference on Circuit, Systems and Communication (ICCSC), Fez, Morocco, 19–20 June 2025; pp. 1–7. [CrossRef]
10. Campoverde-Molina, M.; Luján-Mora, S. Artificial intelligence in web accessibility: A systematic mapping study. *Comput. Stand. Interfaces* **2025**, *96*, 104055. [CrossRef]
11. Herlina. Integrating Accessibility Features and Usability Testing for Inclusive Web Design. In Proceedings of the 2024 International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA), Bali, Indonesia, 17–19 December 2024; pp. 1284–1289. [CrossRef]
12. Duarte, C.; Costa, M.; Seixas Pereira, L.; Guerreiro, J. Expanding Automated Accessibility Evaluations: Leveraging Large Language Models for Heading-Related Barriers. In Proceedings of the IUI '25: 30th International Conference on Intelligent User Interfaces Companion, Cagliari, Italy, 24–27 March 2025; pp. 39–42. [CrossRef]
13. Zan, D.; Chen, B.; Zhang, F.; Lu, D.; Wu, B.; Guan, B.; Yongji, W.; Lou, J.G. Large Language Models Meet NL2Code: A Survey. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Toronto, ON, Canada, 9–14 July 2023. [CrossRef]
14. Yuan, L.; Xu, J.; Gui, H.; Sun, M.; Zhang, Z.; Liang, L.; Zhou, J. Improving Natural Language Understanding for LLMs via Large-Scale Instruction Synthesis. In Proceedings of the AAAI Conference on Artificial Intelligence, Philadelphia, PA, USA, 25 February–4 March 2025; Volume 39, pp. 25787–25795. [CrossRef]
15. Koga, S. Advancing large language models in nephrology: Bridging the gap in image interpretation. *Clin. Exp. Nephrol.* **2024**, *29*, 128–129. [CrossRef] [PubMed]
16. López-Gil, J.M.; Pereira, J. Turning manual web accessibility success criteria into automatic: An LLM-based approach. *Univers. Access Inf. Soc.* **2024**, *24*, 837–852. [CrossRef]
17. Lin, Z.; Guan, S.; Zhang, W.; Zhang, H.; Li, Y.; Zhang, H. Towards trustworthy LLMs: A review on debiasing and dehallucinating in large language models. *Artif. Intell. Rev.* **2024**, *57*, 243. [CrossRef]
18. Othman, A.; Dhouib, A.; Nasser Al Jabor, A. Fostering websites accessibility: A case study on the use of the Large Language Models ChatGPT for automatic remediation. In Proceedings of the PETRA '23: 16th International Conference on PErvasive Technologies Related to Assistive Environments, Corfu, Greece, 5–7 July 2023; pp. 707–713. [CrossRef]
19. Delnevo, G.; Andruccioli, M.; Mirri, S. On the Interaction with Large Language Models for Web Accessibility: Implications and Challenges. In Proceedings of the 2024 IEEE 21st Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 6–9 January 2024; pp. 1–6. [CrossRef]
20. Huynh, G.K.; Lin, W. SmartCaption AI - Enhancing Web Accessibility with Context-Aware Image Descriptions Using Large Language Models. In Proceedings of the 2024 International Conference on Computer and Applications (ICCA), Cairo, Egypt, 17–19 December 2024; pp. 1–7. [CrossRef]
21. Aljedaani, W.; Eler, M.M.; Parthasarathy, P.D. Enhancing Accessibility in Software Engineering Projects with Large Language Models (LLMs). In Proceedings of the SIGCSE TS 2025: The 56th ACM Technical Symposium on Computer Science Education, Pittsburgh, PA, USA, 26 February–1 March 2025; pp. 25–31. [CrossRef]
22. Taeb, M.; Swearngin, A.; Schoop, E.; Cheng, R.; Jiang, Y.; Nichols, J. AXNav: Replaying Accessibility Tests from Natural Language. In Proceedings of the CHI '24: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, 11–16 May 2024; pp. 1–16. [CrossRef]
23. Aljedaani, W.; Habib, A.; Aljohani, A.; Eler, M.; Feng, Y. Does ChatGPT Generate Accessible Code? Investigating Accessibility Challenges in LLM-Generated Source Code. In Proceedings of the W4A '24: The 21st International Web for All Conference, Singapore, 13–14 May 2024; pp. 165–176. [CrossRef]
24. Acosta-Vargas, P.; Salvador-Acosta, B.; Novillo-Villegas, S.; Sarantis, D.; Salvador-Ullauri, L. Generative Artificial Intelligence and Web Accessibility: Towards an Inclusive and Sustainable Future. *Emerg. Sci. J.* **2024**, *8*, 1602–1621. [CrossRef]

25. Novac, O.C.; Madar, D.E.; Novac, C.M.; Bujdosó, G.; Oproescu, M.; Gal, T. Comparative study of some applications made in the Angular and Vue. js frameworks. In Proceedings of the 2021 16th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, Romania, 10–11 June 2021; pp. 1–4.
26. Geetha, G.; Mittal, M.; Prasad, K.M.; Ponsam, J.G. Interpretation and Analysis of Angular Framework. In Proceedings of the 2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India, 8–9 December 2022; pp. 1–6.
27. Javeed, A. Performance optimization techniques for ReactJS. In Proceedings of the 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 20–22 February 2019; pp. 1–5.
28. Gope, D.; Schlais, D.J.; Lipasti, M.H. Architectural support for server-side PHP processing. In Proceedings of the 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, Canada, 24–28 June 2017; pp. 507–520.
29. Bielak, K.; Borek, B.; Plechawska-Wójcik, M. Web application performance analysis using Angular, React and Vue. js frameworks. *J. Comput. Sci. Inst.* **2022**, *23*, 77–83. [CrossRef]
30. Andruccioli, M.; Bassi, B.; Delnevo, G.; Salomoni, P. The Tabular Accessibility Dataset: A Benchmark for LLM-Based Web Accessibility Auditing. *Data* **2025**, *10*, 149. [CrossRef]
31. Kong, A.; Zhao, S.; Chen, H.; Li, Q.; Qin, Y.; Sun, R.; Zhou, X.; Wang, E.; Dong, X. Better Zero-Shot Reasoning with Role-Play Prompting. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Mexico City, Mexico, 16–21 June 2024. [CrossRef]
32. Dhuliawala, S.; Komeili, M.; Xu, J.; Raileanu, R.; Li, X.; Celikyilmaz, A.; Weston, J.E. Chain-of-Verification Reduces Hallucination in Large Language Models. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2024, Bangkok, Thailand, 11–16 August 2024; pp. 3563–3578.
33. Sejnowski, T.J. Large language models and the reverse turing test. *Neural Comput.* **2023**, *35*, 309–342. [CrossRef] [PubMed]
34. Hu, L.; Liu, Z.; Zhao, Z.; Hou, L.; Nie, L.; Li, J. A survey of knowledge enhanced pre-trained language models. *IEEE Trans. Knowl. Data Eng.* **2023**, *36*, 1413–1430. [CrossRef]
35. Min, B.; Ross, H.; Sulem, E.; Veyseh, A.P.B.; Nguyen, T.H.; Sainz, O.; Agirre, E.; Heintz, I.; Roth, D. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Comput. Surv.* **2023**, *56*, 1–40. [CrossRef]
36. Thirunavukarasu, A.J.; Ting, D.S.J.; Elangovan, K.; Gutierrez, L.; Tan, T.F.; Ting, D.S.W. Large language models in medicine. *Nat. Med.* **2023**, *29*, 1930–1940. [CrossRef] [PubMed]
37. Alqahtani, T.; Badreldin, H.A.; Alrashed, M.; Alshaya, A.I.; Alghamdi, S.S.; bin Saleh, K.; Alowais, S.A.; Alshaya, O.A.; Rahman, I.; Al Yami, M.S.; et al. The emergent role of artificial intelligence, natural learning processing, and large language models in higher education and research. *Res. Soc. Adm. Pharm.* **2023**, *19*, 1236–1242. [CrossRef] [PubMed]
38. OpenAI. GPT-4o. 2024. Available online: https://openai.com/index/hello-gpt-4o/ (accessed on 28 February 2025).
39. OpenAI. OpenAI o3-mini. 2022. Available online: https://openai.com/index/openai-o3-mini/ (accessed on 28 February 2025).
40. Google. Gemini 2.0 Flash. 2024. Available online: https://ai.google.dev/models/gemini (accessed on 28 February 2025).
41. Collins, E.; Ghahramani, Z. LaMDA: The Next Generation of Language Models, 2022. Google AI Blog. Available online: https://blog.google/technology/ai/lamda/ (accessed on 11 September 2025).
42. Narang, S.; Chowdhery, A. Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance, 2022. Google AI Blog. Available online: https://research.google/blog/pathways-language-model-palm-scaling-to-540-billion-parameters-for-breakthrough-performance/ (accessed on 11 September 2025).
43. Nozza, D.; Bianchi, F.; Hovy, D. Pipelines for social bias testing of large language models. In Proceedings of the BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models. Association for Computational Linguistics, Dublin, Ireland, 27 May 2022.
44. Kasneci, E.; Seßler, K.; Küchemann, S.; Bannert, M.; Dementieva, D.; Fischer, F.; Gasser, U.; Groh, G.; Günnemann, S.; Hüllermeier, E.; et al. ChatGPT for good? On opportunities and challenges of large language models for education. *Learn. Individ. Differ.* **2023**, *103*, 102274. [CrossRef]
45. Lazar, J. Due Process and Primary Jurisdiction Doctrine: A Threat to Accessibility Research and Practice? In Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility, Galway, Ireland, 22–24 October 2018; pp. 404–406.