Enhancing Web Accessibility: Automated Detection of Issues with Generative AI

ZIYAO HE, University of California, Irvine, USA
SYED FATIUL HUQ, University of California, Irvine, USA
SAM MALEK, University of California, Irvine, USA

Websites are integral to people's daily lives, with billions in use today. However, due to limited awareness of accessibility and its guidelines, developers often release web apps that are inaccessible to people with disabilities, who make up around 16% of the global population. To ensure a baseline of accessibility, software engineers rely on automated checkers that assess a webpage's compliance based on predefined rules. Unfortunately, these tools typically cover only a small subset of accessibility guidelines and often overlook violations that require a semantic understanding of the webpage. The advent of generative AI, known for its ability to comprehend textual and visual content, has created new possibilities for detecting accessibility violations. We began by studying the most widely used guideline, WCAG, to determine the testable success criteria that generative AI could address. This led to the development of an automated tool called GenA11y, which extracts elements from a page related to each success criterion and inputs them into an LLM prompted to detect accessibility issues on the web. Evaluations of GenA11y showed its effectiveness, with a precision of 94.5% and a recall of 87.61%. Additionally, when tested on real websites, GenA11y identified an average of eight more types of accessibility violations than the combination of existing tools.

CCS Concepts: • **Software and its engineering → Software defect analysis**; • **Human-centered computing → Accessibility design and evaluation methods**.

Additional Key Words and Phrases: Accessibility, WCAG, Generative AI, LLM

ACM Reference Format:
Ziyao He, Syed Fatiul Huq, and Sam Malek. 2025. Enhancing Web Accessibility: Automated Detection of Issues with Generative AI. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE101 (July 2025), 24 pages. https://doi.org/10.1145/3729371

1 Introduction

In today's world, websites play a crucial role in people's daily lives. There are approximately 1.09 billion websites in existence [Haan 2024], representing a 2,290% increase compared to twenty years ago [Gupta 2024]. However, one crucial aspect often overlooked by web developers in the implementation of web apps is their accessibility. A recent WebAIM study analyzing the top 1,000,000 home pages revealed that the pages contain an average of 56.8 accessibility errors [WebAIM 2024b]. Such errors hinder people with disabilities, who constitute approximately 16% of the global population according to WHO [WHO 2024], from using the internet. For instance, the WebAIM study observed that 81% of the pages have low-contrast text, making it challenging for people with low vision to read effectively, and 54.5% lack alternative text, leaving blind users without access to visual content. These accessibility issues are significant, affecting a large number of web users. In the U.S. alone, 20 million Americans have visual impairments [Institute 2019], and

Authors' Contact Information: Ziyao He, University of California, Irvine, Irvine, USA, ziyaoh5@uci.edu; Syed Fatiul Huq, University of California, Irvine, Irvine, USA, fsyedhuq@uci.edu; Sam Malek, University of California, Irvine, Irvine, USA, malek@uci.edu.

ACM 2994-970X/2025/7-ARTFSE101
https://doi.org/10.1145/3729371

globally, 2.2 billion people suffer from vision impairment [WHO 2023]. Therefore, it is imperative to design web applications that are accessible to all users, regardless of their abilities or disabilities.

Several studies have investigated the prevalence of accessibility issues and concluded that the primary reasons are a lack of knowledge and awareness of web accessibility guidelines, as well as the insufficiency of existing analysis tools for effectively addressing these issues [Abu-Doush et al. 2013; Bi et al. 2022; Doush and Alhami 2022; Huq et al. 2023; Inal et al. 2019; Patel et al. 2020]. While manual accessibility testing remains the most reliable method for evaluating web app accessibility, it is costly to hire experts to thoroughly assess each page for compliance with accessibility guidelines, and even human experts can make errors [Huq et al. 2023]. In addition to hiring accessibility experts, software organizations can engage people with disabilities to test their products and report any accessibility issues they encounter. However, obtaining feedback from people with disabilities can be challenging [Bi et al. 2022], particularly for small software companies with limited resources, as finding users with various types of disabilities is difficult.

This leaves software teams to rely on automated checkers and testing tools that ensure a baseline level of accessibility. The popularity of these tools stems from their speedy report generation, and easy integration into the testing process, for instance, through browser extensions. Organizations also prefer the tools' prioritization on compliance, in an effort to avoid situations like Domino's Pizza [Feingold 2021], where the company lost a lawsuit due to an inaccessible website, compromising its reputation. However, current automation tools contain a number of flaws, leading to being less effective in detecting violations compared to manual evaluation [Alsaeedi 2020; Mucha 2018].

Existing tools typically rely on accessibility guidelines like Web Content Accessibility Guidelines (WCAG) [W3C 2023c] to assess whether a web page is accessible. However, these tools are often ineffective because they only cover a limited number of WCAG Success Criteria, which consist of testable rules designed to ensure web accessibility for all users. For instance, WAVE, one of the most popular accessibility checkers [Alsaeedi 2020], can detect violations of only 13 out of the 86 success criteria without requiring human intervention, according to its documentation [WAI 2024a,b]. Even for the criteria it does cover, WAVE only partially addresses some of them. For example, success criterion *"SC 1.1.1:Non-text Content"* requires visual content conveying important information to not only have a text alternative but also that the text alternative is descriptive and appropriate. WAVE and other existing tools can address the former requirement but fail to assess the latter, leading to the oversight of many important accessibility issues.

The existing tools' inability to detect many issues, such as the descriptiveness of image alt text, stems from their lack of semantic understanding. However, the emergence of generative AI models, which excel at understanding the relationships between texts and visual elements [Zhao et al. 2023], offers promising advancements in web accessibility. Several researchers have already begun applying these models to improve web accessibility. Most efforts so far have focused on *fixing* accessibility issues using large language models (LLMs), relying on existing tools like WAVE for the detection [Huang et al. 2024; Othman et al. 2023]. However, as mentioned earlier, tools like WAVE cover only a limited number of WCAG Success Criteria, making them insufficient as the sole input for the subsequent fixes. This leads to the research question our paper aims to answer: *How effective and efficient is LLM in detecting accessibility issues as per WCAG guidelines?* While one previous study explored using GPT models to automate the detection of accessibility criteria that typically require manual checks, it only covered three criteria [López-Gil and Pereira 2024]. This paper differs by aiming to cover a broader range of success criteria and by examining the strengths and limitations of using LLMs compared to existing rule-based checkers.

Building on these insights, we developed an automated accessibility checker named GenA11y. GenA11y covers 37 WCAG success criteria. It begins by extracting relevant elements from a webpage for each criterion. The extraction process helps the LLM narrow its focus to the most

pertinent elements. These extracted elements are then analyzed by the LLM, which is prompted to identify accessibility violations. Evaluations of GenA11y using two existing datasets from previous studies, as well as 36 real websites, demonstrated its effectiveness and efficiency. GenA11y achieved a precision of 94.5%, a recall of 87.61%, and on average detected eight more types of violations than all existing tools combined per webpage.

Overall, the paper makes the following contributions:

- A study of WCAG success criteria, including a classification of methods for detecting violations of each criterion.
- The first comprehensive investigation into the application of LLMs for detecting accessibility issues on the web.
- The development and public release of an automated tool named GenA11y [He et al. 2024], specifically designed to detect accessibility issues in web apps.
- Evaluations using three data sources, including 36 real websites, and comparisons with five existing tools, demonstrating the effectiveness and efficiency of GenA11y.

The remainder of this paper is organized as follows. Section 2 presents a motivating example. Section 3 explores the WCAG success criteria, listing all the criteria addressed by GenA11y and the corresponding rationales. Section 4 details the implementation of GenA11y. The evaluation results and comparisons with existing tools are presented in Section 5. The paper concludes with related work, a discussion of threats to validity, and future directions.

2 Motivating Example

In this section, we aim to motivate our study by illustrating the state of the art in automated accessibility checking, and the limitations therein.

The example below demonstrates the use of the WAVE accessibility checker to evaluate the accessibility of the OpenAI homepage [OpenAI 2024c]. The checker identifies one accessibility error, highlighted by the red box in Figure 1: a `Missing form label`, where a `Form Element` on the page lacks an associated label to describe its purpose. However, the OpenAI homepage has more than just this one accessibility issue. In the "Research" section, as shown in the right part of Figure 1, there are four cards, each presenting a new research idea or model from OpenAI. Each card includes both text and an image. The image in each card has an `alt` attribute that is identical to the visible text, such as the first image from the left, which has the `alt text`, *"GPT-4o System Card."* The problem is that the image is unrelated to the text in its `alt` attribute and serves merely as decoration. To prevent confusion for blind users who rely on screen readers — where the same textual content would be announced twice, once for the image and once for the visible text — the image should either use an empty `alt` attribute (`alt=""`) or be included as a background image, which instruct screen readers to ignore the image during navigation.

In addition to missing certain accessibility issues, the tool also generates 79 alerts, highlighted by the yellow box in Figure 1, which developers must manually verify to determine if they are indeed accessibility violations. These alerts include tasks such as verifying whether the alternative text for images is appropriate, whether it is too long, whether adjacent links lead to the same destination, and whether video or audio content has alternative formats, such as transcripts.

However, this can be challenging for developers, who, as discussed in Section 1, commonly lack awareness of web accessibility. Their inexperience with WCAG makes manually verifying the compliance of these elements both tedious and error-prone.

To avoid making incorrect judgments about the validity of the alerts, developers often choose to disregard them, trusting the automated checker's identification of a single error as an indicator of overall website accessibility. However, as illustrated earlier, the website clearly has additional

issues that are not being detected and addressed, leading to an inaccessible application for people with disability.

The example above demonstrates that existing tools often fail to detect accessibility violations requiring semantic understanding. However, the advent of Large Language Models, which are adept at complex tasks like processing and understanding both text and visual elements [Zhao et al. 2023], opens new avenues for enhancing web accessibility. This development inspires us to propose a technique that uses LLMs to automate aspects of accessibility analysis that currently necessitate manual review. Additionally, we aim to evaluate whether this AI-driven approach can match or surpass the performance of existing tools in detecting violations of the established success criteria.

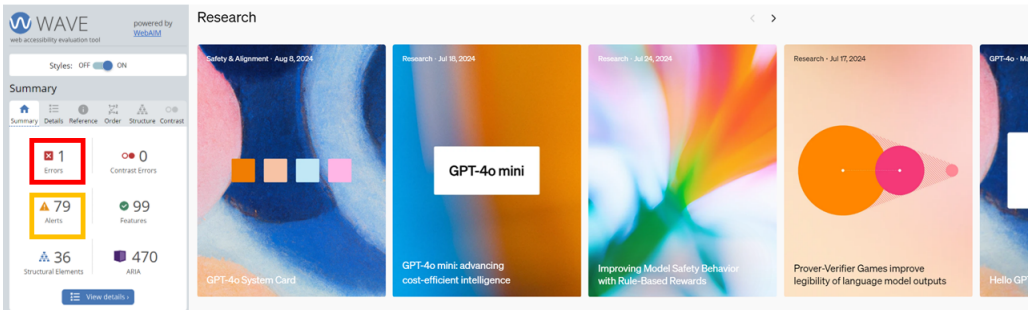

Fig. 1. The analysis result from WAVE accessibility checker.

3 Study of WCAG Success Criteria

The Web Content Accessibility Guidelines (WCAG) is the universally recognized guideline for digital accessibility that informs existing accessibility checkers, as well as GenA11y. Therefore, we present how WCAG is structured, what content is provided, and to what extent GenA11y utilizes it.

3.1 WCAG Structure

WCAG, established in May 1999, aims to enhance web accessibility by providing guidelines that help create more inclusive digital environments for people with disabilities. The latest version, WCAG 2.2, published in October 2023, introduces nine new success criteria compared to version 2.1 and aims to address a broader range of aspects in web accessibility [W3C 2023c]. The guideline is organized into a hierarchical structure that can be summarized as follows:

- **Principles** - WCAG is organized around four key principles that ensure all users can access and interact with web content.
  - **Perceivable:** The content and interface must be presented in ways that users can detect and perceive.
  - **Operable:** Users must be able to interact with and control the interface.
  - **Understandable:** The information and operation of the interface should be clear and comprehensible to all users.
  - **Robust:** Content should remain accessible and functional across different technologies and as those technologies evolve.
- **Guidelines** - WCAG includes 13 guidelines, each organized under one of the four principles. These guidelines outline the essential objectives for web developers to create content that is accessible to people with various disabilities. For instance, the `Perceivable` principle encompasses four guidelines: *Text Alternatives, Time-based Media, Adaptable, Distinguishable.*

- **Success Criteria (SC)** - WCAG 2.2 consists of 86 success criteria. Each guideline is supported by one or more testable success criteria that target specific aspects of accessibility. These criteria help developers assess whether their web applications meet accessibility standards. For example, developers can use success criterion *"SC 1.1.1 Non-text Content"* to ensure that visual elements conveying important information are accessible to blind users. To assist developers in prioritizing these criteria, WCAG categorizes them into three levels: Level A (Minimum), Level AA, and Level AAA. The higher the level, the more stringent the requirements, and the more likely it is that the website will be accessible to a broader range of users. Typically, existing tools cover Level A and some Level AA criteria. Each success criterion includes the following items:
  - **Description** - The description outlines the motivation, purpose, and essential background for each criterion.
  - **Sufficient and Advisory Techniques** - Sufficient techniques are those that ensure conformance to the criterion. Advisory techniques go beyond basic conformance and can further enhance the accessibility of the website.
  - **Common Failures** - This part identifies common scenarios where websites fail to meet the criterion, listing the specific HTML elements involved and explaining why these scenarios result in non-compliance.
  - **Test Rules** - Test rules guide developers in verifying whether their websites comply with the criterion. This includes necessary test steps, relevant elements to be considered, and the expected versus unexpected test outcomes.

Since success criteria are positioned at the bottom of the WCAG hierarchy and are the only components with testable rules and specific scenarios for verifying web accessibility, they become the primary unit for issue detection. This focus on success criteria shapes the design of existing accessibility tools, including GenA11y.

3.2 Classifying WCAG Criteria

In accordance to current practices for detecting accessibility issues, we used WCAG as the primary reference for building GenA11y. We classified the WCAG 2.2's success criteria based on how their violations can be detected. First, we defined two fundamental categories for detecting accessibility issues: static analysis, which only examines HTML, CSS, and JavaScript code, and dynamic analysis, which requires interactive application testing. We then methodically mapped each WCAG success criterion to static or dynamic categories. However, since certain accessibility issues cannot be tested, as acknowledged by WCAG (e.g., *"SC 3.1.5 Reading Level"* [W3C 2023b]), we also categorized certain accessibility issues as "Not Testable". To validate our work, we then compared our categorization of the subset of the success criteria that are also covered by prior tools with their documentation [IBM 2024; WebAIM 2024a] and found no inconsistencies. Additional details, along with screenshot illustrations, are available on our companion website [He et al. 2024].

(1) **Static Analysis**. These criteria can be addressed through static analysis, such as examining `HTML`, `CSS`, and `JavaScript` elements without requiring further interaction, like pointer input or keyboard use. For example, *"SC 2.4.2 Page Titled"* can be evaluated by determining if a page title is present and descriptive based on its alignment with portions of the web page text. Similarly, *"SC 1.4.5 Images of Text"* can be assessed by analyzing the image itself to determine whether visible text is present within it; if so, it would constitute a violation of this criterion. A total of 37 criteria fall under this category.
(2) **Dynamic Analysis**. These criteria require dynamic analysis to generate accurate results. While static analysis can be applied to some of these criteria, it is not as precise as dynamic

analysis. For instance, *"SC 2.4.3 Focus Order"* can be partially addressed through static analysis by examining the `tabindex` attribute to determine the focus order of each element and assess whether it is appropriate. However, since the `tabindex` attribute can be dynamically modified at runtime through JavaScript, the most reliable way to determine the focus order is by using the keyboard to navigate the entire page and evaluate whether the focus order is correct. Additionally, some criteria can only be addressed through dynamic analysis, such as *"SC 2.1.2 No Keyboard Trap"* and *"SC 1.2.2 Captions"*. A common cause of keyboard traps is the continuous insertion of elements into a scrollable list [Salehnamadi et al. 2021], a behavior that can only be detected at runtime. Similarly, the accuracy of captions in a video can only be assessed during runtime. A total of 45 criteria fall under this category.

(3) **Not Testable**. The remaining criteria are not testable, either explicitly stated or implied in their descriptions. For example, in *"SC 3.1.5 Reading Level"* [W3C 2023b], the WCAG Working Group specifically acknowledges that while using the clearest and simplest language is highly desirable, they were unable to develop a method to test for its achievement. A total of four criteria fall under this category.

Although in principle violations in both categories 1 and 2 can be detected by an LLM-based approach, we scope our study to detection of only category 1, i.e., those that can be addressed through static analysis. This allows us to fairly compare GenA11y with existing tools, as most current tools rely on static analysis to detect violations. Furthermore, violations from category 2 require completely different dynamic analysis techniques, such as integrating web crawlers with the ability to create various inputs like mouse or keyboard events. These areas are valuable directions for exploration in future research.

The list of criteria addressed by GenA11y can be found in Table 1. For the rest of the criteria, it can be found in our companion website [He et al. 2024].

4 Approach

Based on the study results from Section 3, we devised a tool called GenA11y that detects violations of 37 success criteria from WCAG. Figure 2 provides an overview of GenA11y, which operates in two main phases. In the first phase, a web scraper extracts relevant elements from the `Document Object Model (DOM)` after receiving the URL of a website. These elements are then fed into individual accessibility analyzers (LLMs), each corresponding to a specific WCAG success criterion, to identify any violations. Each analyzer is equipped with prompts that direct its focus on a specific criterion. After all analyzers have completed their tasks, an accessibility report is generated, detailing the violated elements, issue descriptions, and recommendations for developers.

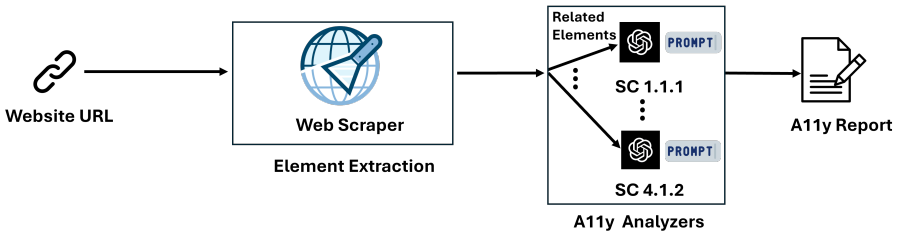

Fig. 2. The overview of GenA11y.

Table 1. The list of WCAG success criteria addressed by GenA11y.

Principle	Guideline	Success Criterion
Perceivable	Text Alternatives	1.1.1 Non-text Content
	Adaptable	1.3.1 Info and Relationships
		1.3.2 Meaningful Sequence
		1.3.3 Sensory Characteristics
		1.3.4 Orientation
		1.3.5 Identify Input Purpose
	Distinguishable	1.4.1 Use of Color
		1.4.2 Audio Control
		1.4.3 Contrast (Minimum)
		1.4.4 Resize Text
		1.4.5 Images of Text
		1.4.6 Contrast (Enhanced)
		1.4.8 Visual Presentation
		1.4.9 Images of Text (No Exception)
		1.4.10 Reflow
		1.4.11 Non-text Contrast
		1.4.12 Text Spacing
Operable	Enough Time	2.2.1 Timing Adjustable
		2.2.2 Pause, Stop, Hide
	Navigable	2.4.1 Bypass Blocks
		2.4.2 Page Titled
		2.4.4 Link Purpose (In Context)
		2.4.5 Multiple Ways
		2.4.6 Headings and Labels
		2.4.8 Location
		2.4.9 Link Purpose (Link Only)
		2.4.10 Section Headings
	Input Modalities	2.5.3 Label in Name
		2.5.5 Target Size (Enhanced)
		2.5.8 Target Size (Minimum)
Understandable	Readable	3.1.1 Language of Page
		3.1.2 Language of Parts
		3.1.4 Abbreviations
	Predictable	3.2.2 On Input
		3.2.5 Change on Request
	Input Assistance	3.3.2 Labels or Instructions
Robust	Compatible	4.1.2 Name, Role, Value

4.1 Element Extraction

During our preliminary investigation with five university home pages, we identified several technical challenges when feeding the LLM directly with the HTML page source, where the details of our preliminary study can be found in our companion website [He et al. 2024]. First, the page source often contains millions of tokens, leading to significant overhead and requiring the creation of multiple chunks to fit within the token limit of current LLMs. Second, critical attributes such as background image URLs, font sizes, and colors may not be present in the HTML page source if they are defined in external CSS files or through JavaScript, preventing the LLM from accurately assessing related violations. Additionally, certain elements, like image URLs, must be directly provided for the LLM to evaluate the descriptiveness of alt texts; without access to the actual image content, this assessment is incomplete. Third, we observed frequent hallucinations—a known issue in LLMs [Perković et al. 2024]—where the model reported many non-violated elements. These hallucinations likely occurred because the LLM lacked access to the full context needed to accurately determine violations, especially when critical visual and structural elements were missing from the input data. Lastly, the LLM's reported violations disproportionately focused on specific success criteria, such as *"SC 1.3.1 Info and Relationships"* and *"SC 4.1.2 Name, Role, Value"*, possibly because these criteria have numerous defined test rules and failure conditions that direct the LLM's attention toward them.

These technical challenges highlighted the need for a more efficient approach. Previous studies have shown that breaking complex tasks into smaller, more manageable components can significantly enhance LLM performance [Khot et al. 2022]. In our study, analyzing WCAG success criteria

violations can similarly benefit from evaluating each success criterion individually. Since not all elements on a web page are relevant to every criterion, extracting only the pertinent elements for the LLM allows it to focus on the most critical aspects and reduce unnecessary overhead. Additionally, the extraction component directly addresses the technical challenges we encountered, ensuring that the LLM has access to the necessary context and data. These dual motivations—technical challenges and the performance gains from task decomposition—form the basis for the development of our Element Extraction component.

To identify the related elements for each success criterion, we reviewed the *sufficient and advisory techniques*, *common failures*, and *test rules* discussed in Section 3. For example, the test rule *"object element rendering non-text content has non-empty accessible name"* indicates that the `<object>` tag is a related element that must be extracted. In cases where the description of elements to be extracted is more abstract, such as in the technique *"ARIA15: Using aria-describedby to provide descriptions of images"*, where images can be represented in various ways on a web page, we consulted additional resources like the HTML Living Standard [WHATWG 2024], CSS specifications [W3C 2023a], and HTML accessibility API mappings [W3C 2024]. Images might be represented through the `<img>` tag, `<picture>` tag for providing multiple images that vary the image content, the CSS `background-image` attribute, or any elements with an ARIA role of `image` or `img`. Consulting these three resources helped us compile a comprehensive list of related elements, even when the WCAG criterion descriptions were abstract.

Some criteria not only require attributes that are defined inline, such as the `alt` attribute, but also those defined elsewhere. For example, to evaluate *"SC 1.4.3 Contrast (Minimum)"*, it is necessary to obtain the foreground color, background color and font size, as large text and normal text have different contrast ratio requirements. These attributes can be defined through inline styles, internal CSS sheets, external CSS sheets, or JavaScript. Therefore, the method for extracting these elements should not be constrained by how those attributes are defined. We utilized the DOM structure along with the built-in Javascript API `window.getComputedStyle()` [MDN 2024] to retrieve these attributes, regardless of their definition location.

Some other criteria also require visual evidence, such as screenshots. For instance, to assess whether text becomes clipped or truncated after zooming to 200%, as required by *"SC 1.4.4 Resize Text"*, full-page screenshots are needed both before and after zooming. Web scrapers offer built-in APIs for capturing static screenshots, and with their scrolling capabilities, full-page captures can be obtained. This visual evidence assists in the subsequent analysis.

Given the related elements of each success criterion, we utilize a web scraper to extract them after receiving the website URL. The extraction process begins after the page has fully loaded, indicated by events such as `document.readyState` and `window.performance.timing.loadEventEnd`. For each criterion, the scraper extracts the related elements, inline attributes, attributes defined elsewhere, and visual evidence screenshots. After the extraction process is completed for all criteria, the related components are sent to the Accessibility Analyzer.

4.2 Accessibility Analyzer

Given that GenA11y focuses on 37 success criteria, there are 37 corresponding accessibility analyzers, each designed to target a specific WCAG criterion. The overall prompting process for GenA11y can be illustrated by focusing on the four main components of prompting in LLMs [Zhao et al. 2023]: *Instruction*, *Contextual Information*, *Input*, and *Output*. Another important component of prompting in LLMs is *demonstration*. However, we chose not to utilize it in our approach because each criterion can be violated in multiple ways, and checking WCAG violations on websites is a specification-heavy task, where a previous study showed that demonstrations often fall short [Peng et al. 2023]. Our preliminary study [He et al. 2024] on five university websites reaffirmed

prior findings. We evaluated two demonstration-based prompting strategies: single failure example per criterion and comprehensive failure sets. Results indicated that comprehensive examples improved context but increased token usage, distracted from task goals, and reduced accuracy. Single examples were more efficient but led to narrow analyses focused on similar failures.

4.2.1 Instruction. An instruction typically summarizes the task that an LLM should perform. We utilized expert prompting [Zhao et al. 2023], assigning GenA11y the role of an accessibility expert to draw out more specialized and in-depth knowledge. The prompt text for GenA11y is as follows:

> You are an Accessibility Expert (WCAG Specialist) responsible for detecting WCAG 2.2 violations on websites. Your expertise is crucial in making the web more accessible for everyone. Please analyze the provided, related HTML, CSS, Javascript and additional visual cue elements for compliance with the specified WCAG success criterion. Be confident in your expertise. Do not limit your findings to the violations mentioned in common failures or test rules; explore beyond these areas for potential issues. Do not omit any issue. After analyzing all elements, do not provide individual element-by-element analysis first. Instead, summarize the overall result.

4.2.2 Contextual Information. Contextual information provides essential background for the LLM to accurately frame the task and produce precise results. GenA11y is given four types of contextual information:

- A short description of the received related elements, outlining the elements that need to be assessed and how they are formatted.
- A brief explanation of the focused criterion, helping the LLM recall relevant information from its memory.
- Common failures associated with the criterion, as discussed in Section 3.
- Test rules for identifying violations of the criterion, as discussed in Section 3.

Below is an example of contextual information for *"SC 2.4.4 Link Purpose (In Context)"*, where this criterion requires providing descriptive names or context for all links. A link element is an anchor `<a>` with an `href` attribute or any element with the `ARIA-Role` specified as "link." Since this criterion allows using context to understand a link's purpose, we include ancestor and sibling elements where present. In this case (not shown in the example below), we traverse the DOM to identify meaningful contextual ancestors, following WCAG's techniques such as enclosing sentences (G53) [Initiative 2025], up to a depth of 5.

> - **Short description of the received, related elements:**
> You will be provided with link elements, along with their ancestors and siblings (if any), from a webpage to assess for any violations of WCAG SC 2.4.4. Each link element will be presented on a new line, starting with '————-'.
> - **Short description of the focused criterion:**
> Unlike SC 2.4.9 Link Purpose (Link Only), this criterion allows relying on contextual information (ancestors and siblings) to determine if a link is descriptive. A link should clearly indicate its purpose without requiring the user to click on it.
> - **Common Failures:**
> F63: Failure of Success Criterion 2.4.4 due to providing link context only in content that is not related to the link...
> - **Test Rules:**
> 1. The link must have a non-empty accessible name ...

4.2.3 Input. Input provides LLMs with the essential data needed to complete a task. In our study, the inputs consist of HTML tags, relevant CSS attributes, and corresponding JavaScript functions. In certain cases, a screenshot of an element or a webpage is also required to accurately analyze potential violations of success criteria.

Below is an example of the input provided for *"SC 2.4.4 Link Purpose (In Context)"*:

> The information you need to assess starts after the dashed line below. Pay attention to all the elements listed.
> ——————
> Link Element 1:
> `<a href="https://en.wikipedia.org/wiki/Polar_bear">Read more</a>`
> ——————
> .....

4.2.4 Output. This component outlines how LLMs should structure their output. For GenA11y, the output is specified in the following JSON format:

```
Your output should be structured as a JSON object with the following format:
{
    "overall_violation": "Yes or No",
    "violated_elements_and_reasons": [
        {
            "element": "outerHTML of the element",
            "reason": "Explanation of why it violates the criterion",
            "recommendation": "Recommendation to fix the
                               violation for this specific element"
        }
    ]
}
If there are no violations, the response should be:
{
            "overall_violation": "No",
            "violated_elements_and_reasons": []
}
```

4.2.5 GPT. For this work, we selected GPT-4o as our LLM. We chose this model because it is the latest release from OpenAI and has demonstrated superior performance in text evaluation and vision understanding, outperforming existing models on established benchmarks [OpenAI 2024a]. Additionally, it supports an input token limit of up to 128K, which is essential for analyzing numerous elements on a web page for accessibility evaluation. In the following section 5.7, we demonstrate that analyzing a real website with GenA11y requires an average of 1.912 million tokens. Given that there are 37 analyzers, this amounts to an average of 51,675 tokens per analyzer. The use of GPT-4o is sufficient for this purpose. Another notable feature of GPT-4o is its ability to produce structured output, allowing the model to generate results in the required format with 100% accuracy [OpenAI 2024d]. We also set the temperature parameter to 0 to enhance its determinism.

5 Evaluation

Our overarching objective is to understand how effective and efficient LLMs are in detecting accessibility issues per WCAG guidelines. To that end, we investigate six research questions:

- **RQ1. Recall:** What is the likelihood of GenA11y detecting accessibility issues that are latent in a web page?
- **RQ2. Coverage:** How many of the accessibility violation types derived from WCAG can be detected by GenA11y?
- **RQ3. Precision:** What is the likelihood of a detected issue being a real issue?
- **RQ4. Variability:** How consistent are GenA11y's detection results across multiple runs?
- **RQ5. Ablation Study:** How does each component of GenA11y contribute to its effectiveness?
- **RQ6. Performance and Cost:** What is the average time and cost required to run GenA11y on real websites?

5.1 Experiment Setup

We used two existing datasets to evaluate GenA11y. The first dataset is called Accessibility Tool Audit (D_a) [Duran 2017], where an accessibility team affiliated with the UK government manually constructed 142 web pages with associated accessibility issues. The second dataset is collected from Ma11y (D_m) [Tafreshipour et al. 2024], where accessibility issues were programmatically injected into real web pages.

Since the scope of our study are the 37 WCAG success criteria that can be detected statically, we mapped each test case in the two datasets to the corresponding success criterion, aiming to thoroughly evaluate the tool's ability to detect violations across all criteria. Collectively, the two datasets covered 29 out of the 37 criteria. During this process, we found that some subject websites in D_a did not align with any WCAG criteria, and some were outdated—such as those related to *"SC 4.1.1 Parsing"*, which was removed in WCAG 2.2. Consequently, 21 subject websites were eliminated, leaving 121 valid ones. Since eight criteria lacked web pages with the corresponding issues for GenA11y to evaluate, we followed the original methodology from D_a, using the sufficient techniques, failures, and test rules specified in WCAG to construct new subject websites. These constructed subject websites were then added to D_a, resulting in a total of 144.

Since both D_a and D_m contain manually constructed or injected accessibility issues, these are ideal for calculating the recall of GenA11y, and were used for answering RQ1 and RQ5. However, we did not rely on these datasets to measure GenA11y's precision. Additional issues, not specified by the datasets but detected by our tool, does not necessarily indicate false positives. These could be accessibility issues unintentionally introduced by the creators of D_a or existing issues in the original state of D_m, before any issues were injected.

To calculate precision (RQ3), we constructed a third dataset (D_r) with real webpages. We used the 30 websites from a prior study [Tafreshipour 2023; Tafreshipour et al. 2024], all in their original state, before any accessibility issues were injected. These 30 websites are the most popular websites in the world according to Semrush [Semrush 2024] and come from 28 different categories (e.g., banking, e-commerce, fashion). We also used D_r to gather performance and cost data in RQ6.

To compare GenA11y with existing tools in terms of precision and recall, we selected five out of six accessibility checkers, identified as the most popular tools according to a prior study [Tafreshipour et al. 2024]: IBM [IBM 2024], QualWeb [QualWeb 2024], Axe-Core [Systems 2024], A11yWatch [A11yWatch 2024], and WAVE [WebAIM 2024a]. We could not include Access Continuum [Access 2024] as we were unable to obtain the required API key from the support team.

For this experiment, we used Python Selenium [Muthukadan 2024] as the web scraper to extract related elements. To improve efficiency, we implemented multi-processing to run multiple accessibility checkers concurrently and assess different success criteria. Multi-threading was also used to prevent idle time while awaiting OpenAI API responses.

All experiments were run on an Asus ROG with an AMD Ryzen 7 processor and 32 GB of memory.

5.2 RQ1 - Recall

Table 2. Recall of GenA11y and existing tools.

WCAG Criteria	Total Issues		Detected											
			GenA11y		IBM		QualWeb		Axe-Core		A11yWatch		WAVE	
	D_a	D_m	D_a	D_m	D_a	D_m	D_a	D_m	D_a	D_m	D_a	D_m	D_a	D_m
SC 1.1.1 Non-text Content	14	62	**14**	**62**	5	34	5	13	4	16	4	11	3	16
SC 1.2.1 Audio-only and Video-only	2	-	0	-	0	-	0	-	0	-	0	-	0	-
SC 1.3.1 Info and Relationships	31	52	**31**	**47**	15	24	8	0	5	0	10	0	1	10
SC 1.3.2 Meaningful Sequence	1	29	**1**	**29**	0	15	0	0	0	0	0	0	0	0
SC 1.3.3 Sensory Characteristics	2	-	**2**	-	0	-	0	-	0	-	0	-	0	-
SC 1.3.4 Orientation	2	-	**2**	-	1	-	1	-	0	-	0	-	0	-
SC 1.3.5 Identify Input Purpose	2	-	**2**	-	**2**	-	**2**	-	0	-	0	-	0	-
SC 1.4.1 Use of Color	2	17	**2**	**17**	0	0	0	0	0	2	0	0	0	0
SC 1.4.2 Audio Control	2	-	**2**	-	0	-	**2**	-	0	-	0	-	0	-
SC 1.4.3 Contrast (Minimum)	3	23	**3**	**23**	**3**	11	**3**	11	**3**	0	1	0	**3**	16
SC 1.4.4 Resize Text	2	27	**2**	**27**	0	0	0	5	0	0	0	0	0	0
SC 1.4.5 Images of Text	5	-	**5**	-	0	-	0	-	0	-	0	-	0	-
SC 1.4.6 Contrast (Enhanced)	2	23	**2**	**23**	**2**	11	**2**	11	**2**	0	0	0	**2**	16
SC 1.4.8 Visual Presentation	2	-	**2**	-	0	-	0	-	0	-	0	-	0	-
SC 1.4.9 Images of Text (No Exception)	5	-	**5**	-	0	-	0	-	0	-	0	-	0	-
SC 1.4.10 Reflow	1	-	**1**	-	0	-	0	-	0	-	0	-	0	-
SC 1.4.11 Non-text Contrast	-	25	-	**24**	-	20	-	0	-	0	-	0	-	0
SC 1.4.12 Text Spacing	2	-	**2**	-	0	-	0	-	0	-	0	-	0	-
SC 2.1.1 Keyboard	8	5	0	0	0	0	0	0	0	0	0	**2**	0	0
SC 2.1.2 No Keyboard Trap	1	-	0	-	0	-	0	-	0	-	0	-	0	-
SC 2.2.1 Timing Adjustable	2	-	**2**	-	**2**	-	**2**	-	0	-	0	-	**2**	-
SC 2.2.2 Pause, Stop, Hide	2	30	**2**	**30**	**2**	0	1	0	**2**	0	1	0	**2**	0
SC 2.2.3 No Timing	1	-	**1**	-	0	-	0	-	0	-	0	-	0	-
SC 2.4.1 Bypass Blocks	1	-	**1**	-	0	-	0	-	0	-	0	-	0	-
SC 2.4.2 Page Titled	3	29	**3**	**29**	2	0	2	0	2	0	2	0	2	0
SC 2.4.3 Focus Order	4	20	0	0	0	0	0	0	0	0	0	0	0	0
SC 2.4.4 Link Purpose (In Context)	2	13	**2**	**13**	0	11	1	10	1	12	1	9	1	0
SC 2.4.5 Multiple Ways	2	-	**2**	-	0	-	0	-	0	-	0	-	0	-
SC 2.4.6 Headings and Labels	1	-	**1**	-	**1**	-	0	-	0	-	**1**	-	**1**	-
SC 2.4.7 Focus Visible	2	24	0	0	0	**11**	0	8	0	0	0	0	0	0
SC 2.4.8 Location	2	-	**2**	-	0	-	0	-	0	-	0	-	0	-
SC 2.4.9 Link Purpose (Link Only)	4	13	**4**	**13**	0	11	0	10	0	12	0	9	0	0
SC 2.4.10 Section Headings	2	-	**2**	-	0	-	0	-	0	-	0	-	0	-
SC 2.5.3 Label in Name	-	6	-	**5**	-	2	-	1	-	0	-	1	-	0
SC 2.5.5 Target Size (Enhanced)	1	-	**1**	-	0	-	0	-	0	-	0	-	0	-
SC 2.5.8 Target Size (Minimum)	1	-	**1**	-	0	-	0	-	0	-	0	-	0	-
SC 3.1.1 Language of Page	4	-	**4**	-	3	-	3	-	3	-	2	-	3	-
SC 3.1.2 Language of Parts	5	-	**5**	-	1	-	1	-	1	-	0	-	1	-
SC 3.1.4 Abbreviations	1	-	**1**	-	0	-	**1**	-	0	-	0	-	0	-
SC 3.2.2 On Input	1	10	**1**	**10**	0	4	0	0	0	0	0	0	0	0
SC 3.2.5 Change on Request	1	35	**1**	**35**	**1**	19	0	20	0	0	0	0	0	23
SC 3.3.2 Labels or Instructions	1	-	**1**	-	**1**	-	0	-	**1**	-	**1**	-	**1**	-
SC 4.1.2 Name, Role, Value	12	2	**12**	**2**	9	**2**	7	**2**	7	**2**	6	**2**	7	1
Total	**144**	**445**	**127**	**389**	**50**	**175**	**41**	**91**	**31**	**44**	**29**	**34**	**29**	**82**
Recall			**87.61%**		**38.20%**		**22.41%**		**12.74%**		**10.70%**		**18.85%**	

This research question seeks to answer how many of the injected accessibility issues can be detected by GenA11y and other existing tools. An issue is considered successfully reported by GenA11y if in the LLM output discussed in Section 4.2.4 the `overall_violation` is marked as *Yes* and the violated elements and their reasons match the known accessibility issues in the dataset. Table 2 presents the results across two datasets, with the highest value for each criterion in each dataset highlighted in bold and shaded.

GenA11y detected 516 out of 589 accessibility issues, achieving an impressive 87.61% recall, and more than twice the number of issues detected by the next best solution. Among the five existing tools, IBM detected the highest number of errors, accounting for 38.20% of the total with 225 violations, followed by QualWeb and WAVE. Axe-Core and A11yWatch detected the fewest.

Most of the issues detected by existing tools are based on the analysis of the HTML syntax. This accounts for all the issues reported in D_a and 80% of those in D_m. Issues that require only syntactical understanding align perfectly with how existing tools are designed, as they are rule-based and each criterion is checked against a set of predefined rules. For example, a common failure under *"SC 1.3.1 Info and Relationships"* is that the `DT` or `DD` elements, which represent definition term and definition description, are not placed within a `DL` (description list) element - a failure that can be easily verified by these tools. Similarly, violations of *"SC 4.1.2 Name, Role, Value"* can be detected by checking if an `ARIA role name` is valid, as ARIA roles consist of a fixed set of values. If the defined value on the webpage does not match one of these predefined ARIA role values, it is a violation.

GenA11y identified 127 out of 144 issues in D_a, and 389 out of 445 issues in D_m. Since the tool does not support issues that require dynamic analysis, it missed 17 and 49 issues from D_a and D_m, respectively. Additionally, GenA11y missed seven issues from D_m as part of a recurring pattern of false negatives. A common violation of *"SC 1.3.1 Info and Relationships"* involves using styling to make text appear as headings rather than using actual heading elements. GenA11y is prompted to analyze screenshots, identify potential headings, and compare them to the actual headings on the page. Any mismatch constitutes a violation. However, when styling is used to mimic lower-level headings, such as `H5` or `H6`, which can look similar to bolded normal text, GenA11y struggles to recognize them as headings and thus misses these violations. This accounted for five instances of false negatives.

GenA11y detects significantly more issues than existing accessibility checkers. It does so due to LLM's ability to reason about violations both semantically and syntactically. For instance, detecting a violation of *"SC 1.4.5 Images of Text"* requires a semantic understanding of images to determine if they convey information through visible text. If they do, it is considered a violation. Similarly, assessing whether a link is descriptive under *"SC 2.4.4 Link Purpose (In Context)"* involves first checking if there is text within the `<a>` tag (a syntactic check), and then determining whether the text accurately conveys the link's purpose or is misleading (a semantic check). The LLM's ability to comprehend both textual and visual content enables GenA11y to detect accessibility violations that other tools might miss.

Existing tools can detect two criteria, *"SC 2.1.1 Keyboard"* and *"SC 2.4.7 Focus Visible"*, that GenA11y cannot identify. While these criteria can be partially detected statically, dynamic analysis is necessary for accurate results. For example, as shown in Table 2, under *"SC 2.4.7"*, QualWeb detected eight violations out of 24. *"SC 2.4.7"* mandates that each item receiving focus must have a visible indicator. These partial detections are enabled by the CSS attribute `focus-visible`, which applies styles when an element is focused, along with `outline-color`, which defines the outline color. However, other attributes like *border-color* and *box-shadow* also influence the appearance of focus indicators. When multiple attributes are defined, the one that takes effect depends on runtime conditions, user interactions, and browser rendering. Additionally, these attributes can be dynamically modified through JavaScript, and the focus indicator can be removed entirely via JavaScript. Therefore, the most reliable way to detect these violations is by manually navigating through each focusable element using the keyboard and checking whether a visible focus indicator appears when the element is focused.

Table 3. Magnitude of detected issues by GenA11y and existing tools on 30 real websites

Website	Detected Violation Types By Existing Tools		Additional Violation Types
	Total	By GenA11y	
Adp	11	11	7
Agoda	6	6	9
Caliente	6	6	11
Capital One	7	7	4
Craigslist	8	8	11
Cricbuzz	9	7	6
Discord.com	4	4	9
Doordash.com	4	3	7
Doubleclick	6	5	7
Ebay.com	9	8	6
Fragrantica	7	7	7
Genius	9	8	11
Google	4	3	9
Live.com	5	4	9
Makemytrip	5	5	10
NIH	7	5	7
OpenAI	6	4	9
Progressive	5	5	7
Qualtrics	4	4	7
Samsung	8	6	6
ScienceDirect	4	4	6
Shein	11	9	7
Stackoverflow	10	10	9
Steam	10	9	11
USPS	9	8	7
Walmart	6	6	5
Yahoo	10	10	7
Youtube	9	9	9
Zerodha	10	10	9
Zoro	7	7	8
Total	216	198	237

5.3 RQ2 - Coverage

We ran GenA11y alongside five existing tools on D_r dataset to determine if GenA11y could achieve similar coverage in detecting violation types as existing tools and identify any additional violation types it could detect. A violation type refers to a success criterion with a violation. To identify these, we first performed a union of all success criteria where violations were detected by existing tools. We then conducted a validation process to include only valid violation types. A success criterion was deemed valid if at least one reported issue was accurate; otherwise, it was excluded from the union set. During this validation process, we observed a common pattern in the invalid violations detected by existing tools. Specifically, when tools attempted to detect violations that are most effectively detected through a dynamic analysis—such as *"SC 2.1.1 Keyboard"*, where the tool reported that certain features could only be accessed via pointer input (like a mouse) instead of a keyboard, or *"SC 2.4.7 Focus Visible"*, where the focus indicator was flagged as not visible—runtime evaluation by us showed these violations were invalid, highlighting the need for dynamic analysis.

We applied the same validation process to GenA11y to determine how many violation types it could detect. Table 3 demonstrates the coverage of all six tools across 30 websites.

GenA11y successfully detected 198 out of 216 violation types reported by existing tools, achieving a similarity ratio of 91.67%. In addition, GenA11y was able to detect 237 additional violation types beyond those found by the combination of all existing tools, averaging eight new violation types per website. Detection of these additional violation types required either semantic understanding or a combination of syntactic and semantic analysis, as previously illustrated in section 5.2.

We identified a few areas where GenA11y was less effective compared to the existing tools, particularly in detecting violations related to *"SC 1.3.1 Info and Relationships"* and *"SC 1.4.3 Color Contrast"*. These accounted for 15 missed cases by GenA11y. For instance, one missed violation involved the `for` attribute in a `<label>` element not pointing to a valid `id` in an `<input>` element.

Existing tools can easily scan the page and verify whether the assigned `id` exists, whereas GenA11y struggled to detect this issue. Additionally, in cases of insufficient color contrast, existing tools use a predefined formula to flag violations. However, GenA11y tended to overlook these violations when a large number of elements were involved and only a few elements had insufficient contrast.

5.4 RQ3 - Precision

Table 4. Precision of GenA11y on 12 websites.

	Published Before			Published After		
Criterion	**Total**	**Correct**	**Precision**	**Total**	**Correct**	**Precision**
1.1.1 Non-text Content	73	63	86.30%	72	67	93.1%
1.3.1 Info and Relationships	58	52	89.66%	81	75	92.6%
1.3.5 Identify Input Purpose	-	-	-	6	4	66.7%
1.4.1 Use of Color	406	389	95.81%	121	114	94.2%
1.4.3 Contrast (Minimum)	260	242	93.08%	213	201	94.4%
1.4.4 Resize Text	5	5	100%	11	9	81.8%
1.4.5 Images of Text	-	-	-	15	11	73.3%
1.4.6 Contrast (Enhanced)	369	347	94.04%	293	277	94.5%
1.4.8 Visual Presentation	14	7	50.00%	3	2	66.7%
1.4.9 Images of Text (No Exception)	-	-	-	15	11	73.3%
1.4.10 Reflow	5	5	100%	7	5	71.4%
1.4.11 Non-text Contrast	-	-	-	40	34	85.0%
1.4.12 Text Spacing	57	57	100%	9	9	100.0%
2.2.2 Pause, Stop, Hide	3	3	100%	8	7	87.5%
2.4.1 Bypass Blocks	89	89	100%	50	47	94.0%
2.4.2 Page Titled	3	0	0%	2	1	50.0%
2.4.4 Link Purpose (In Context)	12	11	91.67%	15	13	86.7%
2.4.5 Multiple Ways	-	-	-	7	7	100.0%
2.4.6 Headings and Labels	14	8	57.14%	20	14	70.0%
2.4.8 Location	5	5	100%	4	4	100.0%
2.4.9 Link Purpose (Link Only)	20	19	95.00%	33	33	100.0%
2.4.10 Section Headings	9	8	88.89%	4	3	75.0%
2.5.3 Label in Name	79	76	96.20%	79	78	98.7%
2.5.5 Target Size	489	489	100%	137	137	100.0%
2.5.8 Target Size (Minimum)	1	1	100%	12	12	100.0%
3.1.1 Language of Page	1	1	100%	1	1	100.0%
3.1.4 Abbreviations	4	4	100%	4	4	100.0%
3.2.5 Change on Request	1	0	0%	-	-	-
3.3.2 Labels or Instructions	12	12	100%	18	17	94.4%
4.1.2 Name, Role, Value	18	18	100%	15	14	93.3%
Overall	**2007**	**1911**	**95.2%**	**1295**	**1211**	**93.5%**

Table 5. The overall precision of the tools on 12 websites.

Tool	**Total**	**Correct**	**Precision**
Axe-Core	33	33	100%
WAVE	603	592	98.2%
IBM	807	787	97.5%
A11yWatch	311	298	95.8%
GenA11y	3302	3122	94.5%
Qualweb	68	49	72.1%

Due to the large volume of accessibility issues detected by GenA11y and other tools, manually validating every reported issue was infeasible. Therefore, we focused on a subset of websites for precision analysis. We began by sorting the 30 subject websites in D_r based on two factors: the number of violated success criteria and the total number of violations. Both factors were max-normalized, each contributing 50% to the overall score for each website. From this ranking, we randomly selected two websites from each tercile: *Steam* and *Ebay* from the top 10, *Craigslist* and *Qualtrics* from the middle 10, and *OpenAI* and *Google* from the bottom 10. These six websites, referred to as **Published Before**, are well-known and have likely been included in the training data of existing LLMs. Therefore, we additionally selected six websites published after the release date

of our model, *gpt-4o-2024-08-06* [OpenAI 2024b]. Three of these were sourced from Web Designer Depot [Depot 2024a,b], and three from the newly registered domain name community list [Shreshta 2025]. We verified their publication dates using the WayBack Machine [Archive 2025], confirming that the earliest archived date for each was after the model's release. These six websites, referred to as **Published After**, are listed on our companion website [He et al. 2024]. For these 12 websites, two authors independently reviewed each detected violation. An issue was considered successfully reported by GenA11y if, in the LLM output discussed in Section 4.2.4, the `overall_violation` field was marked as *Yes*, the violated elements and their reasons matched the corresponding WCAG success criterion, and the elements were visible on the screen. Any disagreements between the authors were resolved through discussion.

The precision of GenA11y on these twelve websites is shown in Table 4. GenA11y reported a total of 3,302 accessibility violations, of which 3,122 were found to be valid, resulting in a precision of 94.5%. We observed no significant difference in precision between websites published before and after the model's release date. GenA11y detected one violation of a criterion that was only found in the **Published Before** dataset and five violations of criteria that were only found in the **Published After** dataset. There were six success criteria that had no false positives across either datasets, such as *"2.5.5 & 2.5.8 Target Size"*, *"2.4.8 Location"*, *"3.1.4 Abbreviation"*. We also identified recurring patterns of false positives in GenA11y's reports, particularly in criteria *"1.1.1 Non-text Content"*, *"1.4.1 Use of Color"*, *"1.4.3 & 1.4.6 Contrast"*, *"1.4.8 Visual Presentation"*, *"2.4.2 Page Titled"*, and *"2.4.6 Headings and Labels"*. For instance, a common issue with GenA11y is its tendency to be overly strict when assessing the appropriateness of page titles, alt texts, or headings. On OpenAI, GenA11y considered the page title "OpenAI" insufficiently descriptive and suggested changing it to something like "Learn About OpenAI: Research, Products, and Opportunities."

Another pattern of false positives occurs with *"1.4.1 Use of Color"*. A common failure is marking links when they are only visually distinguishable by color, such as using blue without underlining or other indicators. In this case, GenA11y emphasizes text-based methods like underlining, bolding, or italicizing, while overlooking shape-based cues. On Ebay, for example, links lack underlining but use visual markers like borders or arrows alongside the anchor element (<a>), yet GenA11y flagged them as violations for not using text-based indicators. This may be because WCAG provides numerous examples of how to use text-based techniques to meet this criterion, leading GPT models to prioritize these methods after training. Refined prompts may help reduce these false positives.

Regarding existing tools, as shown in Table 5, Axe-Core is the most precise checker, achieving a perfect precision of 100%, though it detected the fewest issues. Wave follows, identifying 592 correct issues out of 603, with a precision of 98.2%. IBM and A11yWatch also perform well, while Qualweb has the lowest precision at 72.1%, detecting the second fewest accessibility violations.

5.5 RQ4- Variability

As discussed in Section 4.2.5, we set the temperature parameter of GenA11y to 0 to increase its determinism. However, prior research has shown that even with the temperature parameter set to 0, determinism is not always guaranteed [Ouyang et al. 2023]. To evaluate the consistency of GenA11y's detection results — the recall and precision values obtained in **RQ1** and **RQ3**, we ran GenA11y five times, following the same approach as in the literature [Huq et al. 2024].

5.5.1 How variable is recall across multiple runs? We ran GenA11y on the Accessibility Tool Audit (D_a) and Ma11y (D_m) datasets five times. Table 6 summarizes the number of issues detected in each run for both datasets.

The second and fifth runs detected the same number of issues as when GenA11y was run once in Section 5.2 for **RQ1**. In these runs, the tool missed 17 issues from D_a and 49 issues from D_m

due to its current inability to support issues requiring dynamic analysis. Additionally, GenA11y consistently missed seven issues from D_m, including a recurring pattern of false negatives related to violations of *"SC 1.3.1 Info and Relationships"*. Specifically, these violations involved using styling to make text appear as headings instead of using proper heading elements.

In the first and fourth runs, GenA11y detected the same number of issues in D_a but missed two additional issues in the first run and one additional issue in the fourth run for D_m. These missed issues were also related to *"SC 1.3.1 Info and Relationships"*, consistent with the recurring pattern described earlier. In the third run, GenA11y missed one issue in D_a, which was related to determining whether an element had sufficient color contrast. Overall, the results in Table 6 demonstrate that GenA11y exhibits uniformity in recall across multiple runs.

Table 6. Variability of GenA11y on Recall.

Run	D_a		D_m	
	Detected	Missing	Detected	Missing
1	131	17	387	51
2	131	17	389	49
3	130	18	389	49
4	131	17	388	50
5	131	17	389	49

5.5.2 How variable is precision across multiple runs? We ran GenA11y on the **Published After** dataset five times to evaluate its precision variability. Similar to the process in Section 5.4, two authors independently verified each detected violation, and any disagreements were resolved through discussion. Table 7 summarizes the precision results across each run.

The precision difference between runs was minimal, ranging from 92.9% in the second run to 93.7% in the third run. This demonstrates that GenA11y exhibits uniformity in precision.

We also identified common issues across all five runs by comparing the `outerHTML` of the violated elements generated by GenA11y. The `outerHTML`, which includes attributes and textual descriptions, served as a way to determine whether a reported issue was the same across runs. A common issue in this context refers to a violation detected in all five runs. As shown in Table 7, the majority of issues were consistently detected across all runs, while a small fraction were either unique to a single run or appeared in some but not all runs. This demonstrates that GenA11y is stable in terms of the number of issues detected per run.

Table 7. Variability of GenA11y on Precision.

Run	Common	Total	Correct	Precision
1	1142	1295	1211	93.5%
2	1142	1249	1160	92.9%
3	1142	1241	1163	93.7%
4	1142	1277	1191	93.3%
5	1142	1268	1187	93.6%

5.6 RQ5- Ablation Experiment

We conducted an ablation experiment on D_a dataset with four alternative models with the objective of evaluating the contributions of our design choices in the development of GenA11y:

- **Base Model** utilizes the latest version of GPT-4o and is given the `HTML` of a website to detect accessibility violations.
- **Extraction-Only Model** adds the extraction component introduced in Section 4. The extracted elements are then fed to the GPT-4o model with only basic prompting that instructs it to detect accessibility violations for these elements.

- **Prompting-Only Model** adds detailed prompting as discussed in Section 4, but the input is merely the HTML of a website.
- **GenA11y**. This model combines both element extraction and detailed prompting.

The results of the ablation experiment are presented in Table 8. The base model detected 57 issues. By incorporating either the extraction or prompting methodology, the model identified 23 additional issues compared to the base model alone. For instance, consider the missing title attribute in an `Iframe` element. When our extraction methodology is added, it enables the LLM to narrow its focus specifically on the `Iframe`. Alternatively, when the prompting methodology is introduced, it provides contextual information that includes checking whether the `Iframe` element has a title attribute. This approach also guides the LLM to pay particular attention to the `Iframe`.

The extraction methodology provides the LLM with the necessary information that is otherwise unavailable in the HTML. For example, one subject website has text with a contrast ratio of less than `4.5:1`, which violates *"SC 1.4.3 Contrast (Minimum)"*. We need the extraction component to retrieve the background and foreground colors so the LLM can calculate the contrast ratio. Additionally, we also need the font size because small and large text have different contrast ratio requirements. The addition of the extraction component identifies 13 more unique issues compared to the base model.

The prompting methodology guides the LLM in assessing the page for violations. For instance, *"SC 2.4.5 Multiple Ways"* requires that a website to provide at least two ways to access the same content. The prompt sets this threshold and clarifies what counts as different options. Our prompting methodology enables the model to identify 24 additional unique issues than the base model.

In some cases, detecting violations requires both the extraction and prompting methodologies, accounting for 10 issues that cannot be detected by either the extraction or prompting model alone. For instance, to assess whether an image's `alt text` is appropriate, the image URL must first be extracted and then formatted according to the LLM's requirements to ensure that it can properly access and evaluate the image. Additionally, accessibility guidelines need to be provided as part of the prompt to help LLM determine what constitutes an inappropriate `alt text` for an image.

Table 8. Ablation experiment results: recall achieved by different models.

Model	Accessibility Issues Detected
Base model	57
Extraction model	93
Prompting model	104
GenA11y (Extraction + Prompting)	127

5.7 RQ6 - Performance and Cost

The performance of GenA11y was evaluated by running it on 30 real websites. The average time GenA11y took to evaluate a website was 240 seconds. GenA11y operates in two phases. The first phase, described in Section 4.1, involves extracting related elements. This phase is not significantly affected by the website size, ranging from a minimum of 25 seconds on *Google.com* to a maximum of 65 seconds on *Shein.com*. The second phase, discussed in Section 4.2, analyzes these elements for accessibility violations. This phase is more sensitive to website size, taking as little as 67 seconds to evaluate *Google.com* and up to 402 seconds for *Shein.com*, where a larger number of elements increases the analysis time. Even on content-heavy websites like *Shein.com*, which contain numerous text and image elements, GenA11y's performance remains reasonable. Given the parallel-processing choices made in the implementation of GenA11y, we expect its performance to improve if it were to be deployed on a server with many processors.

The average cost of running GenA11y is 4.78 USD per website, with around 1.912 million tokens consumed. This cost is directly linked to the size of the website, as larger sites contain more elements and therefore use more tokens.

6 Related Work

The Web Content Accessibility Guidelines (WCAG) have inspired the development of most existing accessibility checkers [A11yWatch 2024; Agency 2021; Benavidez 2015; Broccia et al. 2020; Gay and Li 2010; IBM 2024; QualWeb 2024; Systems 2024; WebAIM 2024a]. However, due to their rule-based nature and reliance on static analysis, these tools primarily understand the syntax of web pages and cover only a limited portion of WCAG success criteria. As a result, they often fail to adequately detect accessibility violations on the web.

Several studies have explored detecting accessibility violations through dynamic analysis during interactions with web pages [Durgam et al. 2023; Takagi et al. 2003], and inferring and assigning correct accessibility attributes, like valid ARIA labels to visual elements that were previously missing them [Bajammal and Mesbah 2021; Duarte et al. 2018]. Additionally, dynamic changes, such as pop-up windows introduced by JavaScript during runtime, can create accessibility issues for individuals with disabilities. As a result, some studies have focused on detecting these inaccessible dynamic changes [Fernandes et al. 2012a,b; Sensiate et al. 2020; Watanabe et al. 2017].

Researchers have discovered that certain accessibility violations can only be detected when using assistive technologies, which might otherwise be overlooked. This has led to the incorporation of assistive technology like screen readers during the evaluation to identify issues such as keyboard navigation problems [Chiou et al. 2023a,b, 2021]. A common characteristic of studies employing dynamic analysis techniques is that they typically focus on one or two specific WCAG criteria, making them less comprehensive than static analyzers.

The rise of generative AI in recent years presents new opportunities for web accessibility evaluation. To date, most efforts have centered around automatically fixing issues detected by existing tools [Huang et al. 2024; Othman et al. 2023]. Given that current checkers predominantly identify syntactic errors, an important next step is to develop more advanced tools capable of both syntactic and semantic understanding of web pages. A recent study explored the use of generative AI for detecting accessibility violations but only addressed 3 WCAG criteria and was not fully automated [López-Gil and Pereira 2024]. Additionally, the study focused exclusively on the aspects of these three criteria that require semantic understanding, while overlooking the ability of generative AI to detect syntactic errors. Moreover, the `HTML` was provided directly to the model. In contrast, GenA11y is fully automated, covers 37 success criteria, extracts information not available in the `HTML` source, and was evaluated across three datasets, including 36 real websites.

Accessibility has also been studied outside the domain of web. For instance, several prior works have proposed rule-based mobile accessibility analysis tools that leverage guidelines from Apple and Google [Android 2023, 2024a,b; Chen et al. 2021; da Silva et al. 2022; Eler et al. 2018; Hao et al. 2014; KIF 2023; Li et al. 2023]. Several accessibility issues have been determined to be only detectable through interactions that involve assistive technologies, and led to studies incorporating such technology in the analysis process [Alotaibi et al. 2022; Alshayban and Malek 2022; Mehralian et al. 2022; Salehnamadi et al. 2021, 2023, 2022; Taeb et al. 2023]. To date, no study has combined platform guidelines with generative AI for broader accessibility detection.

7 Threats to Validity

External Validity. GenA11y leverages GPT-4o in its implementation. However, we believe our approach is generalizable to other commercial LLMs, such as Google Gemini [Google 2025], which also process text and images, and can understand web pages.

To evaluate GenA11y's precision and performance, we conducted an assessment using websites from 28 distinct categories. We selected these sites based on their popularity rankings within their respective categories according to Semrush [Semrush 2024]. By including websites of varying sizes and complexities, we have strived to strengthen the generalizability of our results.

To address potential bias from LLMs being trained on websites in our evaluation dataset, we tested GenA11y on 6 additional websites published after the LLM's release date. As a result, we found no significant differences in precision when running GenA11y on the two datasets.

To ensure reliable precision analysis across the six tools, two authors independently reviewed each detected violation. The review of 5,124 issues yielded 122 disagreements, with a kappa value of 0.745, indicating substantial inter-rater agreement. Evaluating GenA11y's precision variability over 1,643 issues from five runs revealed 57 disagreements, with a kappa of 0.721.

To provide a comprehensive comparison, we evaluated the precision and recall of GenA11y against existing tools. For fairness, we selected five popular accessibility checkers from a previous study [Tafreshipour et al. 2024], including WAVE, one of the most widely used tools in industry.

To address concerns about the non-deterministic nature of LLMs, we conducted five separate runs of GenA11y for both recall and precision measurements. The analysis showed no significant differences across runs, demonstrating that the evaluation results from Section 5.2 (Recall) and Section 5.4 (Precision), obtained from a single run, align closely with the results from multiple runs.

Internal Validity. GenA11y integrates several libraries, including Selenium and OpenAI API, which introduces potential risks of defects. Additionally, there may be defects in our tool's implementation. To mitigate these risks, we used up-to-date third-party tools, conducted thorough GitHub code reviews, and tested the tool on websites outside the final evaluation.

8 Conclusion

Existing accessibility tools detect only a limited number of accessibility violations due to their lack of semantic understanding of web content. However, with the advent of generative AI, new possibilities have emerged, as these models excel at comprehending both textual and visual elements. In this work, we introduced GenA11y, an automated accessibility checker that leverages generative AI. GenA11y first extracts relevant elements from the web page, which are then processed by LLM-based accessibility analyzers. These analyzers are provided with customized prompted to identify potential accessibility violations. Our evaluation of GenA11y on two existing datasets and popular websites demonstrated its efficiency and effectiveness. GenA11y detected 291 more issues in two existing datasets, achieving a recall of 87.61%, which is 49.41% higher than the best existing tool. Additionally, when applied on real-world websites, on average GenA11y identified eight more violations types per page than all the existing tools combined, with a precision of 94.5%.

Future directions for this work include employing generative AI to detect accessibility violations that require dynamic analysis. Additionally, during our experiments, we observed that GenA11y not only detects violations but also provides clear descriptions and actionable suggestions. This is especially beneficial for software developers who may not be familiar with accessibility guidelines. An interesting direction for future research is to conduct a developer study to evaluate GenA11y's effectiveness in helping developers understand and resolve accessibility violations, as well as to compare developers' preferences for GenA11y's results with those of existing tools.

Our research artifacts are publicly available [He et al. 2024].

Acknowledgments

This work has been supported, in part, by award numbers 2211790 and 2106306 from the National Science Foundation. We thank the anonymous reviewers for their detailed feedback, which helped us improve the work.

References

A11yWatch. 2024. *A11yWatch: Web accessibility evaluation tool.* A11yWatch. Retrieved August 23, 2024 from https://a11ywatch.com/

Iyad Abu-Doush, Ashraf Bany-Mohammed, Emad Ali, and Mohammed Azmi Al-Betar. 2013. Towards a more accessible e-government in Jordan: an evaluation study of visually impaired users and Web developers. *Behaviour & Information Technology* 32, 3 (2013), 273–293.

Level Access. 2024. *Access Continuum overview – Level Access Help Center.* Level Access. Retrieved August 23, 2024 from https://client.levelaccess.com/hc/en-us/articles/13844879206807-Access-Continuum-overview

Administrative Modernization Agency. 2021. *Access Monitor Plus.* Administrative Modernization Agency. Retrieved August 23, 2024 from https://accessmonitor.acessibilidade.gov.pt/

Ali S Alotaibi, Paul T Chiou, and William GJ Halfond. 2022. Automated detection of talkback interactive accessibility failures in android applications. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST).* IEEE, 232–243.

Abdullah Alsaeedi. 2020. Comparing web accessibility evaluation tools and evaluating the accessibility of webpages: proposed frameworks. *Information* 11, 1 (2020), 40.

Abdulaziz Alshayban and Sam Malek. 2022. AccessiText: automated detection of text accessibility issues in Android apps. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 984–995.

Android. 2023. *Accessibility Scanner - Apps on Google Play.* Google. Retrieved August 23, 2024 from https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor&hl=en_US

Android. 2024a. *Espresso : Android Developers.* Google. Retrieved August 23, 2024 from https://developer.android.com/training/testing/espresso

Android. 2024b. *Improve your code with lint checks.* Google. Retrieved August 23, 2024 from https://developer.android.com/studio/write/lint?hl=en

Internet Archive. 2025. *Wayback Machine.* Retrieved Feburary 7, 2025 from https://web.archive.org/

Mohammad Bajammal and Ali Mesbah. 2021. Semantic Web Accessibility Testing via Hierarchical Visual Analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE).* IEEE, 1610–1621.

Carlos Benavidez. 2015. *Examinator.* Retrieved August 23, 2024 from http://examinator.net/

Tingting Bi, Xin Xia, David Lo, John Grundy, Thomas Zimmermann, and Denae Ford. 2022. Accessibility in software practice: A practitioner's perspective. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 4 (2022), 1–26.

Giovanna Broccia, Marco Manca, Fabio Paternò, and Francesca Pulina. 2020. Flexible automatic support for web accessibility validation. *Proceedings of the ACM on Human-Computer Interaction* 4, EICS (2020), 1–24.

Sen Chen, Chunyang Chen, Lingling Fan, Mingming Fan, Xian Zhan, and Yang Liu. 2021. Accessible or Not An Empirical Investigation of Android App Accessibility. *IEEE Transactions on Software Engineering* 48 (2021), 3954–3968.

Paul T Chiou, Ali S Alotaibi, and William GJ Halfond. 2023a. BAGEL: An Approach to Automatically Detect Navigation-Based Web Accessibility Barriers for Keyboard Users. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems.* 1–17.

Paul T Chiou, Ali S Alotaibi, and William GJ Halfond. 2023b. Detecting Dialog-Related Keyboard Navigation Failures in Web Applications. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 1368–1380.

Paul T. Chiou, Ali S. Alotaibi, and William G. J. Halfond. 2021. Detecting and Localizing Keyboard Accessibility Failures in Web Applications *(ESEC/FSE 2021).* Association for Computing Machinery, New York, NY, USA, 855–867. https://doi.org/10.1145/3468264.3468581

Henrique Neves da Silva, Silvia Regina Vergilio, and André Takeshi Endo. 2022. Accessibility Mutation Testing of Android Applications. *Journal of Software Engineering Research and Development* 10 (2022), 8–1.

Web Designer Depot. 2024a. *20 Best New Websites, October 2024.* Retrieved Feburary 7, 2025 from https://webdesignerdepot.com/20-best-new-websites-october-2024/

Web Designer Depot. 2024b. *40 Best New Websites, 2024.* Retrieved Feburary 7, 2025 from https://webdesignerdepot.com/40-best-new-websites-2024/

Iyad Abu Doush and Ikdam Alhami. 2022. Evaluating the accessibility of computer laboratories, libraries, and websites in Jordanian Universities and Colleges. In *Research anthology on physical and intellectual disabilities in an inclusive society.* IGI Global, 1968–1985.

Carlos Duarte, Ana Salvado, M. Elgin Akpinar, Yeliz Yeşilada, and Luís Carriço. 2018. Automatic Role Detection of Visual Elements of Web Pages for Automatic Accessibility Evaluation *(W4A '18).* Association for Computing Machinery, New York, NY, USA, Article 21, 4 pages. doi:10.1145/3192714.3196827

Mehmet Duran. 2017. *Accessibility tools audit results - Overview - GDS accessibility team.* alphagov. Retrieved August 23, 2024 from https://alphagov.github.io/accessibility-tool-audit/index.html

Fernando Durgam, Julián Grigera, and Alejandra Garrido. 2023. Dynamic detection of accessibility smells. *Universal Access in the Information Society* (2023), 1–12.

Marcelo Medeiros Eler, José Miguel Rojas, Yan Ge, and Gordon Fraser. 2018. Automated accessibility testing of mobile apps. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation*. ICST, Västerås, Sweden, 116–126.

Lainey Feingold. 2021. *Another Big Win in the Domino's Pizza Accessibility Saga.* Retrieved August 23, 2024 from https://www.lflegal.com/2021/06/dominos-june-2021/

Nádia Fernandes, Daniel Costa, Carlos Duarte, and Luís Carriço. 2012a. Evaluating the accessibility of web applications. *Procedia Computer Science* 14 (2012), 28–35.

Nádia Fernandes, Daniel Costa, Sergio Neves, Carlos Duarte, and Luís Carriço. 2012b. Evaluating the accessibility of rich internet applications. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*. 1–4.

Greg Gay and Cindy Qi Li. 2010. AChecker: open, interactive, customizable, web accessibility checking. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*. Association for Computing Machinery, Raleigh, USA, 1–2.

Google. 2025. *Gemini - chat to supercharge your ideas.* Retrieved Feburary 7, 2025 from https://gemini.google.com/

Ruchi Gupta. 2024. *How Many Websites Are There? The Web in 2024.* Retrieved August 23, 2024 from https://themeisle.com/blog/how-many-websites-are-there/#gref

Katherine Haan. 2024. *Top Website Statistics For 2024.* Retrieved August 23, 2024 from https://www.forbes.com/advisor/business/software/website-statistics/

Shuai Hao, Bin Liu, Suman Nath, William GJ Halfond, and Ramesh Govindan. 2014. PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM New York, NY, USA, Bretton Woods, New Hampshire, USA, 204–217.

Ziyao He, Syed Fatiul Huq, and Sam Malek. 2024. *GenA11y Companion Website.* SEAL. Retrieved April 18, 2025 from https://github.com/seal-hub/GenA11y.git

Calista Huang, Alyssa Ma, Suchir Vyasamudri, Eugenie Puype, Sayem Kamal, Juan Belza Garcia, Salar Cheema, and Michael Lutz. 2024. ACCESS: Prompt Engineering for Automated Web Accessibility Violation Corrections. *arXiv preprint arXiv:2401.16450* (2024).

Syed Fatiul Huq, Abdulaziz Alshayban, Ziyao He, and Sam Malek. 2023. # A11yDev: Understanding Contemporary Software Accessibility Practices from Twitter Conversations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–18.

Syed Fatiul Huq, Mahan Tafreshipour, Kate Kalcevich, and Sam Malek. 2024. Automated Generation of Accessibility Test Reports from Recorded User Transcripts. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 534–546.

IBM. 2024. *Verify - automated – IBM Accessibility.* IBM. Retrieved August 23, 2024 from https://www.ibm.com/able/toolkit/verify/automated

Yavuz Inal, Kerem Rızvanoğlu, and Yeliz Yesilada. 2019. Web accessibility in Turkey: awareness, understanding and practices of user experience professionals. *Universal Access in the Information Society* 18 (2019), 387–398.

Web Accessibility Initiative. 2025. *G53: Identifying the purpose of a link using link text combined with the text of the enclosing sentence.* Retrieved Feburary 7, 2025 from https://www.w3.org/WAI/WCAG22/Techniques/general/G53

Health Policy Institute. 2019. *Visual Impairments.* Georgetown University. Retrieved August 23, 2024 from https://hpi.georgetown.edu/visual/#:~:text=Almost%2020%20million%20Americans%20%E2%80%94%208,people%20age%2065%20and%20older.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406* (2022).

KIF. 2023. *Keep It Functional - An iOS Functional Testing Framework.* KIF. Retrieved August 23, 2024 from https://github.com/kif-framework/KIF

Linlin Li, Ruifeng Wang, Xian Zhan, Ying Wang, Cuiyun Gao, Sinan Wang, and Yepang Liu. 2023. What You See Is What You Get? It Is Not the Case! Detecting Misleading Icons for Mobile Applications. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 538–550.

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

MDN. 2024. *Window: getComputedStyle() method - Web APIs | MDN.* Mozilla. Retrieved August 23, 2024 from https://developer.mozilla.org/en-US/docs/Web/API/Window/getComputedStyle

Forough Mehralian, Navid Salehnamadi, Syed Fatiul Huq, and Sam Malek. 2022. Too Much Accessibility is Harmful! Automated Detection and Analysis of Overly Accessible Elements in Mobile Apps. In *2022 37th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, ACM New York, NY, USA, Rochester, Michigan, USA, 13 pages.

Justyna Magdalena Mucha. 2018. *Combination of automatic and manual testing for web accessibility.* Master's thesis. Universitetet i Agder; University of Agder.

Baiju Muthukadan. 2024. *Selenium with Python — Selenium Python Bindings 2 documentation.* Retrieved August 23, 2024 from https://selenium-python.readthedocs.io/

OpenAI. 2024a. *Hello GPT-4o.* OpenAI. Retrieved August 23, 2024 from https://openai.com/index/hello-gpt-4o/

OpenAI. 2024b. *Models - OpenAI API.* Retrieved Feburary 7, 2025 from https://platform.openai.com/docs/models#current-model-aliases

OpenAI. 2024c. *OpenAI.* OpenAI. Retrieved August 23, 2024 from https://openai.com/

OpenAI. 2024d. *Structured Outputs.* OpenAI. Retrieved August 23, 2024 from https://platform.openai.com/docs/guides/structured-outputs/introduction

Achraf Othman, Amira Dhouib, and Aljazi Nasser Al Jabor. 2023. Fostering websites accessibility: A case study on the use of the Large Language Models ChatGPT for automatic remediation. In *Proceedings of the 16th International Conference on PErvasive Technologies Related to Assistive Environments*. 707–713.

Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. 2023. LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. *arXiv preprint arXiv:2308.02828* (2023).

Rohan Patel, Pedro Breton, Catherine M Baker, Yasmine N El-Glaly, and Kristen Shinohara. 2020. Why software is not accessible: Technology professionals' perspectives and challenges. In *Extended abstracts of the 2020 CHI conference on human factors in computing systems*. 1–9.

Hao Peng, Xiaozhi Wang, Jianhui Chen, Weikai Li, Yunjia Qi, Zimu Wang, Zhili Wu, Kaisheng Zeng, Bin Xu, Lei Hou, et al. 2023. When does In-context Learning Fall Short and Why? A Study on Specification-Heavy Tasks. *arXiv preprint arXiv:2311.08993* (2023).

Gabrijela Perković, Antun Drobnjak, and Ivica Botički. 2024. Hallucinations in LLMs: Understanding and Addressing Challenges. In *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*. IEEE, 2084–2088.

QualWeb. 2024. *QualWeb.* QualWeb. Retrieved August 23, 2024 from https://qualweb.di.fc.ul.pt/evaluator/

Navid Salehnamadi, Abdulaziz Alshayban, Jun-Wei Lin, Iftekhar Ahmed, Stacy Branham, and Sam Malek. 2021. Latte: Use-case and assistive-service driven automated accessibility testing framework for android. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–11.

Navid Salehnamadi, Ziyao He, and Sam Malek. 2023. Assistive-Technology Aided Manual Accessibility Testing in Mobile Apps, Powered by Record-and-Replay. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–20.

Navid Salehnamadi, Forough Mehralian, and Sam Malek. 2022. GroundHog: An Automated Accessibility Crawler for Mobile Apps. In *2022 37th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, ACM New York, NY, USA, Rochester, Michigan, USA, 13 pages.

Semrush. 2024. *Semrush - Online Marketing Can Be Easy.* Semrush. Retrieved August 23, 2024 from https://www.semrush.com/

Leonardo Sensiate, Humberto Lidio Antonelli, Willian Massami Watanabe, and Renata Pontin de Mattos Fortes. 2020. A Mechanism for Identifying Dynamic Components in Rich Internet Applications. In *Proceedings of the 38th ACM International Conference on Design of Communication* (Denton, TX, USA) *(SIGDOC '20)*. Association for Computing Machinery, New York, NY, USA, Article 42, 8 pages. https://doi.org/10.1145/3380851.3416785

Shreshta. 2025. *Newly Registered domain names community lists.* Retrieved Feburary 7, 2025 from https://shreshtait.com/newly-registered-domains/nrd-1w

Deque Systems. 2024. *axe: Accessibility Testing Tools and Software.* Deque Systems. Retrieved August 23, 2024 from https://www.deque.com/axe/

Maryam Taeb, Amanda Swearngin, Eldon School, Ruijia Cheng, Yue Jiang, and Jeffrey Nichols. 2023. Axnav: Replaying accessibility tests from natural language. *arXiv preprint arXiv:2310.02424* (2023).

Mahan Tafreshipour. 2023. *mahantaf.github.io.* Retrieved August 23, 2024 from https://github.com/mahantaf/mahantaf.github.io/tree/main

Mahan Tafreshipour, Anmol Deshpande, Forough Mehralian, Iftekhar Ahmed, and Sam Malek. 2024. Ma11y: A Mutation Framework for Web Accessibility Testing. In *International Symposium on Software Testing and Analysis*. ACM, 1–12.

Hironobu Takagi, Chieko Asakawa, Kentarou Fukuda, and Junji Maeda. 2003. Accessibility designer: visualizing usability for the blind. *ACM SIGACCESS accessibility and computing* 77-78 (2003), 177–184.

W3C. 2023a. *CSS Snapshot 2023.* W3C. Retrieved August 23, 2024 from https://www.w3.org/TR/css-2023/

W3C. 2023b. *Understanding SC 3.1.5:Reading Level (Level AAA).* W3C. Retrieved August 23, 2024 from https://www.w3.org/WAI/WCAG22/Understanding/reading-level.html

W3C. 2023c. *Web Content Accessibility Guidelines (WCAG) 2.2.* W3C. Retrieved August 23, 2024 from https://www.w3.org/TR/WCAG22/

W3C. 2024. *HTML Accessibility API Mappings 1.0.* W3C. Retrieved August 23, 2024 from https://www.w3.org/TR/html-aam-1.0/#html-element-role-mappings

WAI. 2024a. *WAVE Help.* WAI. Retrieved August 23, 2024 from https://wave.webaim.org/help

WAI. 2024b. *WAVE-WCAG Mappings.* WAI. Retrieved August 23, 2024 from https://docs.google.com/spreadsheets/d/1oSyK4QiyHf1zx-xrc4P9M7efYVv0vohNxVWjeHan8Iw/edit?gid=902071383#gid=902071383

Willian Massami Watanabe, Renata PM Fortes, and Ana Luiza Dias. 2017. Acceptance tests for validating ARIA requirements in widgets. *Universal Access in the Information Society* 16 (2017), 3–27.

WebAIM. 2024a. *WAVE Web Accessibility Evaluation Tool.* WebAIM. Retrieved August 23, 2024 from https://wave.webaim.org/

WebAIM. 2024b. *WebAIM: The WebAIM Million - The 2024 report on the accessibility of the top 1,000,000 home pages.* Retrieved August 23, 2024 from https://webaim.org/projects/million/

WHATWG. 2024. *HTML Standard.* WHATWG. Retrieved August 23, 2024 from https://html.spec.whatwg.org/

WHO. 2023. *Blindness and vision impairment.* WHO. Retrieved August 23, 2024 from https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment

WHO. 2024. *Disability.* Retrieved August 23, 2024 from https://www.who.int/health-topics/disability#tab=tab_2

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

Received 2024-09-12; accepted 2025-04-01