AccessGuru: Leveraging LLMs to Detect and Correct Web Accessibility Violations in HTML Code

NADEEN FATHALLAH, University of Stuttgart, Artificial Intelligence, Germany
DANIEL HERNÁNDEZ, Artificial Intelligence, University of Stuttgart, Germany
STEFFEN STAAB, University of Stuttgart, Germany and University of Southampton, United Kingdom

The vast majority of Web pages fail to comply with established Web accessibility guidelines, excluding a range of users with diverse abilities from interacting with their content. Making Web pages accessible to all users requires dedicated expertise and additional manual efforts from Web page providers. To lower their efforts and, thus, promote inclusiveness, we aim to automatically detect and correct Web accessibility violations in HTML code. While previous work has made progress in detecting certain types of accessibility violations, the problem of automatically detecting and correcting accessibility violations remains an open challenge that we address.

We introduce a novel taxonomy classifying Web accessibility violations into three key categories— Syntactic, Semantic, and Layout. This taxonomy provides a structured foundation for developing our detection and correction method and selecting and redefining evaluation metrics. We propose our novel method, *AccessGuru*, which combines existing accessibility testing tools and Large Language Models (LLMs) to detect accessibility violations of Web accessibility guidelines and taxonomy-driven prompting strategies of LLMs to correct all three accessibility violation categories.

To evaluate these capabilities, we have developed a novel benchmark encompassing Web accessibility violations from real-world Web pages. Our benchmark quantifies syntactic and layout compliance and judges semantic accuracy through a comparative analysis against human expert corrections. Evaluation against our benchmark demonstrates that our method achieves up to 84% average violation score decrease on our benchmark dataset, significantly outperforming existing methods, which achieve at most 50% average violation score decrease.

CCS Concepts: • **Computing methodologies** → **Natural language processing**; • **Human-centered computing** → **Accessibility technologies**.

Additional Key Words and Phrases: Web accessibility, Web accessibility violations, HTML correction, Large Language Models, Prompt Engineering

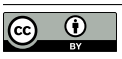
ACM Reference Format:
Nadeen Fathallah, Daniel Hernández, and Steffen Staab. 2025. AccessGuru: Leveraging LLMs to Detect and Correct Web Accessibility Violations in HTML Code. In *The 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25), October 26–29, 2025, Denver, CO, USA.* ACM, New York, NY, USA, 25 pages. https://doi.org/10.1145/3663547.3746360

1 Introduction

Web accessibility is the ability of websites to be accessed by all people, including people with visual, auditory, motor, and cognitive impairments such that they can perceive, understand, navigate, interact with, and contribute to the Web effectively [43, 57, 58]. Organizations like Web Accessibility in Mind (WebAIM) and the World Wide Web Consortium (W3C) have established guidelines, such as the Web Content Accessibility Guidelines (WCAG) [10], which advise developers to create accessible Web content. However, an overwhelming majority of Web pages do not comply with Web accessibility guidelines; for instance, a 2025 study by WebAIM revealed that 94.8% of the top one million most visited Web pages fail to meet accessibility guidelines [55]. This widespread non-compliance is likely due to a lack of expertise in creating accessible content in the first place and the costly and labor-intensive effort of correcting accessibility issues in the HTML code [6, 9].

Users with impairments rely on various assistive technologies, such as screen readers for visually impaired users, screen magnification tools for individuals with low vision, closed captioning for those with hearing impairments, and voice recognition software for users with motor disabilities. These technologies function effectively when accessibility-related information is embedded in HTML. Based on the distinct types of HTML constructs and information required by assistive technologies—as

Authors' Contact Information: Nadeen Fathallah, nadeen.fathallah@ki.uni-stuttgart.de, University of Stuttgart, Artificial Intelligence, Stuttgart, Germany; Daniel Hernández, daniel.hernandez@ki.uni-stuttgart.de, Artificial Intelligence, University of Stuttgart, Stuttgart, Baden-Württemberg, Germany; Steffen Staab, steffen.staab@uni-stuttgart.de, University of Stuttgart, Stuttgart, Germany and University of Southampton, Southampton, United Kingdom.



Manuscript submitted to ACM

specified by the WCAG—we introduce a taxonomy of accessibility violations that classifies them into three dimensions: Syntactic, Semantic, and Layout. Syntactic accessibility violations involve missing or malformed accessibility-enhancing HTML elements and attributes (e.g., missing alt text); Semantic accessibility violations concern whether the provided accessibility-enhancing HTML elements and attributes are meaningful (e.g., alt text that fails to describe the image content); and Layout accessibility violations refer to visual or structural barriers that impede interaction (e.g., insufficient color contrast). A detailed explanation of this taxonomy, its construction, and representative examples can be found in Section 3.

Research has led to the creation of efficient and accurate tools to automatically detect syntactic and layout accessibility violations in HTML, like WAVE, Axe, Google Lighthouse, and AChecker. Current tools fail to identify semantic accessibility violations such as Violation 5 (line 25) in Listing 1. This highlights a significant gap in current accessibility detection methods.

Automating the correction of syntactic, layout, and semantic accessibility violations remains an open challenge. Web accessibility guidelines provide general recommendations rather than detailed solutions. For example, they recommend making interactive elements keyboard-operable for users without mouse access, but don't specify how to manage keyboard focus for dynamically loaded content.

This paper addresses the challenge of improving Web accessibility by introducing a method for automatically detecting and correcting HTML accessibility violations. Our taxonomy provides a systematic understanding of Web accessibility violations and informs our development of detection and correction strategies and the choice of evaluation metrics. *AccessGuru* relies on an automatic Web accessibility evaluation tool to detect syntactic and layout accessibility violations and LLMs to detect semantic accessibility violations. *AccessGuru* transforms Web pages with accessibility violations into guideline-compliant versions by leveraging a pre-trained LLM to generate corrections that minimize a violation score, reflecting their impact on user interaction. In this paper, we make the following contributions:

- We introduce a novel taxonomy that classifies Web accessibility violations into three key categories: Syntactic, Semantic, and Layout. This taxonomy provides a structured foundation for understanding accessibility violations, guiding detection and correction strategies, and informing evaluation metrics.
- We present a dataset of 3,500 real-world Web accessibility violations spanning over 112 distinct types across syntactic, semantic, and layout categories, offering a diverse and representative basis for training and evaluating correction methods. To our knowledge, this is the first publicly available dataset of this scale that includes all three types of accessibility violations, sourced entirely from real-world Web data. The dataset will be published upon acceptance of this paper.
- We develop a novel pipeline that leverages accessibility testing tools and LLMs to automatically detect and correct Web accessibility violations. Building on recent ideas of using LLMs for coding [3, 4, 7, 13, 29, 34, 50].
- We evaluate the correction effectiveness of *AccessGuru* against three baseline methods [15, 23, 37] using a subset of our dataset sampled to reflect real-world violation distributions as indicated by [55], and an additional dataset from [23]. Our method achieves up to 84% average violation score decrease, outperforming baselines capped at 50%. We compare the corrections generated by *AccessGuru* to those generated by human developers. To this end, we conducted a human developer correction study on 55 Web accessibility violations from our dataset, achieving an average similarity of 73%.

To ensure reproducibility and transparency, all source codes, prompts, datasets, and results are available at https://github.com/NadeenAhmad/AccessGuruLLM. The structure of the paper is as follows: Section 2 provides an overview of the existing literature and background information, followed by our proposed taxonomy in Section 3. Section 4 details our approach, Section 5 describes the evaluation process, and the results from our experiments are presented in Section 6. Section 7 discusses the limitations of our method, and Section 8 concludes the paper with directions for future research.

2 Related Work

2.1 WCAG Guidelines

The WCAG [10, 24] are a widely adopted set of guidelines aimed at making Web content accessible to individuals with diverse abilities, including visual, auditory, motor, and cognitive impairments. WCAG are organized around four fundamental principles, referred to by the acronym POUR: (i) Perceivable, ensuring information and interface elements are presented in ways users can perceive, like providing text alternatives for images; (ii) Operable, requiring Website functionality to be accessible through different input methods, such as keyboard navigation or voice commands; (iii) Understandable, ensuring content is clear and easy to interact

with; and (iv) Robust, which demands that content is compatible with current and future technologies. Each of these principles is broken down into specific, testable success criteria, grouped into three conformance levels: A (minimum), AA (mid-range), and AAA (highest).

2.2 Web Accessibility Violation Detection

Early efforts in accessibility violation detection relied on manual testing and expert assessment, which were time-consuming and lacked scalability. This motivated the development of efficient tools like WAVE, Axe, Google Lighthouse, AChecker, and Tenon, which translate WCAG accessibility guidelines into rule-based checks and apply them to individual HTML elements to detect syntax and layout accessibility violations [59]. These reports provide developers with information on each violation, including its type, description, and impact. In our work, we utilize the Axe-Playwright tool, an accessibility detection engine. However, such tools fail to detect semantic accessibility violations [30]. For instance, [33] can confirm the presence of alt text but cannot evaluate whether the description effectively conveys the image's content.

2.3 Web Accessibility Violation Correction

Rule-based Corrections of Accessibility Violations. Early Web accessibility correction methods defined rules to automate corrections [5, 11, 19, 48]. For example, [19] defines a fixed rule that inserts a skip link before a navigation bar to jump to the `<main>` tag. However, such rules assume consistent page structure and fail to generalize across diverse Web layouts.

Computer Vision Correcting Accessibility Violations. Computer vision techniques have been used to correct web accessibility violations, particularly for generating alt text. Facebook's automatic alt text feature applied object detection to describe images for visually impaired users [57], leveraging neural image captioning models such as [51]. Computer vision has been applied to accessibility in graphical user interfaces (GUIs) by using deep learning models to predict natural-language labels for HTML elements, enabling better navigation for visually impaired users [12, 27].

Semi-automatic methods for Correcting Accessibility Violations suggest an initial correction to a human expert for refinement. [31, 38, 45] proposed using image recognition techniques to provide a first draft of the alt text, followed by human validation.

LLM Prompting for Correcting Accessibility Violations. LLMs have been shown to effectively generate code and detect errors [3, 4, 7, 13, 29, 34, 50]. Recent studies have also shown that LLMs can generate accessibility-conformant code [4, 16]. Othman et al. [37] used contextual prompting, where ChatGPT was provided with a non-conformant accessibility code and WCAG 2.1 guidelines to generate corrections. Delnevo et al. [15] explored zero-shot prompting to correct accessibility violations by instructing ChatGPT to determine whether HTML elements are accessibility complaints. Huang et al. [23] investigated three prompting techniques: Reasoning + Acting (ReAct), Few-Shot, and Chain of Thought (CoT). ReAct, which combines reasoning steps with interactive responses, achieved the best performance on their dataset. We adopt these three prompting techniques—contextual [37], zero-shot [15], and ReAct [23]—as baselines for comparison against our proposed method. The prompt templates used for each baseline are directly adapted from the original works and are provided in Table 12. While these methods were effective in correcting some syntax and layout accessibility violations, they don't consider correcting semantic accessibility violations. Additionally, the lack of comparison with human-generated corrections limits a comprehensive evaluation of how well these models handle more complex, real-world semantic accessibility challenges.

LLMs as Assistive Agents for Web Accessibility. LLMs can act as interactive assistive agents, helping users navigate and operate inaccessible web pages. Kodandaram et al. [25] introduced a system that uses LLMs to interpret screen elements and execute spoken user commands such as clicking buttons or filling out forms. Similarly, Mehendale and Walishetti [32] proposed the use of one LLM to translate voice input into web interactions and the use of a second LLM to validate the outcome, thus improving the success rates of task execution. While such systems offer compelling advantages, the translation from content to accessible representation must be performed repeatedly for every user interaction, incurring high computational costs. Furthermore, because the transformations occur in real-time and do not persist in the web content itself, they are difficult to monitor and correct. Their lack of transparency hinders reproducibility and quality assurance.

2.4 Prompt Engineering Techniques

LLMs are highly sensitive to prompt structure, producing more accurate and contextually relevant responses when guided by well-structured prompts. This observation has given rise to the field of prompt engineering, which involves designing instructions that generative AI models can effectively interpret [44, 56]. We survey techniques that we have adopted in the following subsections.

2.4.1 Role-play prompting directs LLMs to adopt specific personas, characters, professions, or roles (e.g., doctor, teacher, or historical figure), priming them to provide responses that align with the expert knowledge of that role. This approach can help guide the model's output to be more precise and factually accurate without altering the underlying capabilities of the LLM, as evidenced by studies [18, 26, 35, 46, 52].

2.4.2 Contextual prompting enriches prompts with task-specific information to guide the model's response [22, 41]. Dynamic contextual prompting adapts the context to each task, improving accuracy and reducing hallucinations. Prior work [22, 41] shows that task-specific context yields more relevant and higher-quality outputs than zero-shot.

2.4.3 Metacognitive Prompting, metacognition refers to the ability to self-reflect and critically evaluate one's own cognitive processes [20, 21]. Metacognitive prompting [53, 60] mimics human cognitive steps through the following stages: **(a) Self-understanding:** the model assesses its understanding of the prompt by identifying and interpreting relevant information, **(b) Reflection:** the model undergoes preliminary judgment, forming an initial response, followed by inference evaluation, where it reflects on and refines its initial interpretation, and **(c) Self-regulation:** the model finalizes its response and performs a confidence assessment to evaluate the reliability of its decision.

2.4.4 Corrective Re-prompting involves re-querying the model with error-related feedback when a generated action or response fails to meet certain conditions [40]. The feedback typically includes information about why the action failed (such as unmet preconditions), allowing the model to adjust and generate a more appropriate response.

3 Our Taxonomy of Web Accessibility Violations

We introduce a structured taxonomy of Web accessibility violations that categorizes them into three distinct dimensions: *Syntactic*, *Semantic*, and *Layout*. To construct the taxonomy, we adopted a methodology from prior work on taxonomy standardization and classification [47, 49]. In the first stage, we conducted a literature review to identify recurring Web accessibility violations, including studies incorporating user feedback.

In the second stage, we analyzed WCAG 2.1 alongside rule definitions from automated testing tools (Axe-Playwright [33], WAVE [54], AChecker [1]), extracting and consolidating violation types. We adopt naming conventions from Axe-Playwright to ensure consistency and interpretability (e.g., `button-name`, `html-has-lang`, `color-contrast`). Each type was defined by its associated WCAG and required correction logic. For instance, although both `form-label-mismatch` and `button-label-mismatch` relate to label mismatch, we identified them as distinct semantic accessibility violations: the former refers to form elements whose labels fail to describe their input purpose, while the latter involves buttons with labels that do not reflect their action. Violations exhibiting functional overlap were merged or redefined to ensure mutually exclusive boundaries.

In the third stage, we validated the taxonomy's relevance and generalizability by evaluating its alignment with outputs from automated testing tools (Axe-Playwright [33], WAVE [54], AChecker [1]) and cross-referencing the prevalence of accessibility violations against large-scale reports from the WebAIM 2025 study [55]. For semantic accessibility violations— absent from automated tools—we conducted manual audits of Web pages from [55]. Violations were retained only if observed repeatedly across Web pages.

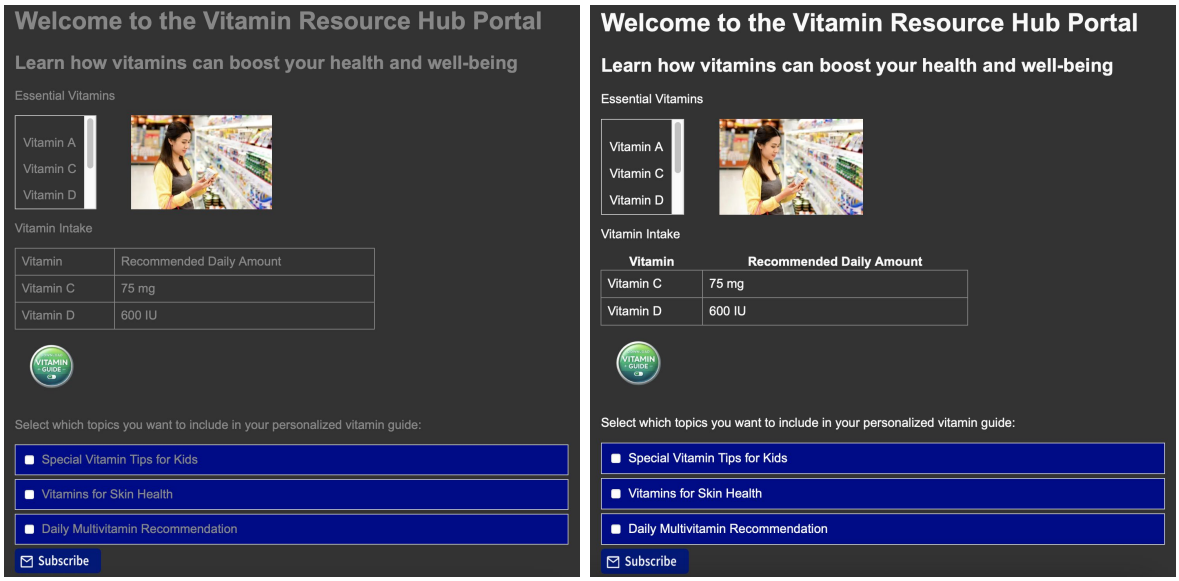
```html
<!DOCTYPE html>
<!-- Violation 1: Missing Language Attribute (Syntax)-->
<html>
<head>
<!--Violation 2: Viewport prevents scaling, zooming (Layout)-->
    <meta name="viewport" content="width=device-width,minimum-scale=1,maximum-scale=1,user-scalable=no">
    <title>Health Information Portal</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body style="font-family: Arial, sans-serif; background-color: #fafafa; color: #888888;">
<!-- Violation 3: Contrast between foreground and background doesn't meet WCAG 2 AA minimum contrast ratio thresholds (Layout)-->
    <div style="background-color: #333; padding: 15px;">
    <h1 style="color: #888888;">Welcome to the Vitamin Resource Hub Portal</h1>
    <div class="content">
    <h2>Learn how vitamins can boost your health and well-being</h2>
    <p>Essential Vitamins</p>
<!-- Violation 4: Scrollable region not keyboard accessible (Syntax)-->
    <div class="scrollable-content">
        <p>Vitamin A </p>
        <p>Vitamin C</p>
        <p>Vitamin D</p>
        <p>Vitamin E</p>
        <p>Vitamin K</p>
    </div>
<!-- Violation 5: Image alt text not descriptive (Semantic)-->
    <img src="https://img.Webmd.com/woman.jpg" alt="image">
    <!-- Violation 6: Table structure missing accessibility attributes (Syntax)-->
    <table>
        <tr>
            <td>Vitamin</td>
            <td>Recommended Daily Amount</td>
        </tr>
        <tr>
            <td>Vitamin C</td>
            <td>75 mg</td>
        </tr>
        <tr>
            <td>Vitamin D</td>
            <td>600 IU</td>
        </tr>
    </table>
<!-- Violation 7: Link without discernible text (Syntax)-->
    <a href="vitamin-guide.pdf"><img src="https://vitaminguide.png" height="95" width="95"></a>
    <p>Select which topics you want to include in your personalized vitamin guide:</p>
<!-- Violation 8: Nested interactive elements (Syntax)-->
    <div>
        <label class="vitamin-checkbox" role="checkbox" aria-checked="false">
            <input type="checkbox"> Special Vitamin Tips for Kids
        </label>
        <label class="vitamin-checkbox" role="checkbox" aria-checked="false">
            <input type="checkbox"> Vitamins for Skin Health
        </label>
        <label class="vitamin-checkbox" role="checkbox" aria-checked="false">
            <input type="checkbox"> Daily Multivitamin Recommendation
        </label>
        </div>
<!-- Violation 9: Button without discernible text (Syntax)-->
    <button class="subscribe-button" type="button">
    <img src="https://ScreenShot.png">
    </button> </div> </div> </body></html>
```

Listing 1. Synthesized HTML web page from our dataset containing accessibility violations. The listing shows the accessibility violations detected by *AccessGuru*.

```html
<!DOCTYPE html>
<!-- Fixed Violation 1: Added Language Attribute -->
<html lang="en">
<head>
<!-- Fixed Violation 2: Viewport allow scaling, zooming-->
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Health Information Portal</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body style="font-family: Arial, sans-serif; background-color: #fafafa; color: #fafafa;">
<!-- Fixed Violation 3: Changed contrast between foreground and background to meet WCAG 2 AA standards -->
    <div style="background-color: #333; padding: 15px;">
    <h1 style="color: #fafafa;">Welcome to the Vitamin Resource Hub Portal</h1>
    <div class="content">
    <h2>Learn how vitamins can boost your health and well-being</h2>
    <p>Essential Vitamins</p>
<!-- Fixed Violation 4: Scrollable region accessible via keyboard (added tabindex attribute) -->
    <div class="scrollable-content" tabindex="0" >
        <p>Vitamin A </p>
        <p>Vitamin C</p>
        <p>Vitamin D</p>
        <p>Vitamin E</p>
        <p>Vitamin K</p>
    </div>
<!-- Fixed Violation 5: Replaced nondescriptive alt text--> <img src="https://img.Webmd.com/woman.jpg" alt=" Woman reading vitamin box label while standing in a pharmacy aisle.">
<!-- Fixed Violation 6: Table structure & accessibility attributes (<th>) -->
    <table>
    <thead>
        <tr>
            <th scope="col">Vitamin</th>
            <th scope="col">Recommended Daily Amount</th>
        </tr> </thead> <tbody>
        <tr>
            <td>Vitamin C</td>
            <td>75 mg</td>
        </tr>
        <tr>
            <td>Vitamin D</td>
            <td>600 IU</td>
        </tr> </tbody> </table>
<!-- Fixed Violation 7: Added accessible text for the link-->
    <a href="vitamin-guide.pdf"><img src="https://vitaminguide.png" alt="Link to download  PDF guide on vitamins"></a>
    <p>Select which topics you want to include in your personalized vitamin guide:</p>
<!-- Fixed Violation 8: Removed improper nesting of interactive elements -->
    <div>
        <label class="vitamin-checkbox">
            <input type="checkbox" aria-label="Special Vitamin Tips for Kids"> Special Vitamin Tips for Kids
        </label>
        <label class="vitamin-checkbox">
            <input type="checkbox" aria-label="Vitamins for Skin Health"> Vitamins for Skin Health
        </label>
        <label class="vitamin-checkbox">
            <input type="checkbox" aria-label="Daily Multivitamin Recommendation"> Daily Multivitamin Recommendation
        </label>
    </div>
<!-- Fixed Violation 9: Added discernible text for button -->
    <button class="subscribe" type="button" aria-label="Subscribe to Vitamin Newsletter">
    <img src="https://ScreenShot.png" alt="Subscribe Button">
    </button> </div> </div> </body></html>
```

Listing 2. *AccessGuru* delivers this accessibility-compliant HTML code when provided with Listing 1.

Fig. 1. Interface of the Web page before (a) and after (b) correction with *AccessGuru*, corresponding to the code in Listing 1 and Listing 2. This example includes corrections of all three accessibility violation types: (1) *Syntactic*—added missing table headers and ARIA attributes, (2) *Semantic*—replaced non-descriptive alt text with meaningful descriptions, and (3) *Layout*—adjusted color values to improve contrast. The overall visual appearance remains largely unchanged for typical users.



- *Syntactic accessibility violations* arise when the required accessibility-enhancing HTML elements and attributes are missing or malformed. These include elements like alt text for images or ARIA (Accessible Rich Internet Applications) attributes, which help define the behavior and purpose of interactive components. Syntactic accessibility compliance is fulfilled if all required and useful syntactic constructs for indicating accessibility information are present. For instance, in Listing 1 Violation 6 (line 27), a table presenting vitamin data lacks accessibility syntax elements like `<th>` and scope attributes. This omission hinders screen readers from correctly identifying table headers, making it difficult for visually impaired users to understand the relationship between vitamins and their recommended daily amounts.
- *Layout accessibility violations* occur when the visual and spatial arrangement of content fails to meet accessibility guidelines. This includes sufficient color contrast between text and background to ensure readability. It also includes ensuring that users can adjust content presentation based on their needs, such as enabling text scaling and zooming.
- *Semantic accessibility violations* occur when HTML accessibility elements are present but fail to convey meaningful content. In Listing 1 Violation 5 (line 25), an image includes an alt attribute set to the generic string `"image"`—a common issue caused by auto-generated text. The corrected version is shown in Listing 2, line 25.

Our taxonomy comprises over 112 violation types. Each violation is annotated with associated WCAG guidelines, impact level, and required supplementary information (e.g., `image-alt-not-descriptive` requires the image alongside the HTML to detect and correct the violation). For syntactic and layout accessibility violations, impact levels are derived from existing tools [33], which judge the severity of each violation based on how much it hinders user interaction with the Web content. For semantic accessibility violations, we manually assign impact levels using analogous criteria (e.g., `button-label-mismatch` inherits the "Critical" level from its syntactic counterpart `button-name`). A representative subset is shown in Table 13, and the complete taxonomy can be found in the supplementary material of this paper.

4 Methodology

AccessGuru addresses Web accessibility violations in two stages: first, by automatically detecting accessibility violations, and second, by automatically generating corrections for the detected accessibility violations.

4.1 *AccessGuruDetect*: Web Accessibility Violation Detection

AccessGuruDetect uses an Axe-Playwright-based detector to detect syntactic and layout accessibility violations following methodologies from similar studies [2, 23] and an LLM to detect semantic accessibility violations (See Figure 2).

Given a target HTML document, *AccessGuruDetect* executes the following steps:

(1) The Axe-Playwright tool is executed on the target HTML document; the tool generates a detailed violation report, which includes for each violation: **(a)** violation name, **(b)** affected HTML elements, **(c)** violation description, and **(d)** impact level. We enrich the violation with **(e)** numerical violation scores ranging from 1 (lowest) to 5 (highest) by mapping the qualitative impact levels reported by Axe-Playwright ("cosmetic", "minor", "moderate", "serious", and "critical") to corresponding numeric values.
(2) For each violation, we use the violation name to query our predefined taxonomy, which specifies whether supplementary information beyond the HTML is required for correction (see Table 13). If additional data is needed—such as color values for contrast violations—we extract it as **(f)** supplementary information from the rendered HTML document (e.g., computed CSS styles for background and foreground colors).

To detect semantic accessibility violations, *AccessGuru* uses an LLM-based semantic detector that runs in parallel to the Axe-Playwright-based detector. Given the raw HTML document and a screenshot of its rendered view. *AccessGuruDetect* executes the following steps:

(1) The HTML document is rendered in a headless browser that captures a long screenshot, resulting in a full-page image to handle scrolling. The image is captured at a resolution of 1920×1080 pixels.
(2) The LLM is prompted with the HTML document, screenshot, and our semantic violation taxonomy. We show the prompt template in Table 8. The prompt instructs the LLM to identify all semantic accessibility violations and enclose violation **(a)** names and **(b)** affected HTML elements within string markers (e.g., `[START]`, `[END]`). To support reasoning over non-textual content (Web page screenshot), the LLM must exhibit multimodal capabilities.
(3) Each marked segment is matched against the original HTML code to ensure it refers to an existing element.
(4) If any of the returned HTML segments do not match elements in the original HTML, they are discarded as hallucinations and excluded from the final set of detected violations.
(5) For each violation, we use the violation name returned by the LLM to look up our taxonomy to assign **(c)** violation description, **(d)** impact level ("cosmetic", "minor", "moderate", "serious", and "critical"). We enrich the violation with **(e)** numerical violation scores ranging from 1 (lowest) to 5 (highest).
(6) For each violation, we use the violation name to look up our taxonomy to determine if supplementary information to the HTML is required (e.g., images for `image-alt-not-descriptive`, videos for `video-caption-alt-not-descriptive`). If so, we extract **(f)** supplementary information crucial for correcting accessibility violations.

For illustration purposes, Listing 1 includes annotated comments to indicate the locations of accessibility violations; these comments are added manually for reader clarity and are not provided as input to *AccessGuruDetect*. When the unannotated HTML in Listing 1 is processed, *AccessGuruDetect* identifies all the annotated accessibility violations. Each detector runs independently, and the aggregated output is stored as a single JSON file containing a set of violation entries. Samples from the output are shown in Tables 1 and 2.

AccessGuruCorrect operates on the output of *AccessGuruDetect*—a set of detected Web accessibility violations—taking as input $V = \{v_1, v_2, \ldots, v_n\}$, where each v_i represents an individual violation (see Figure 2).

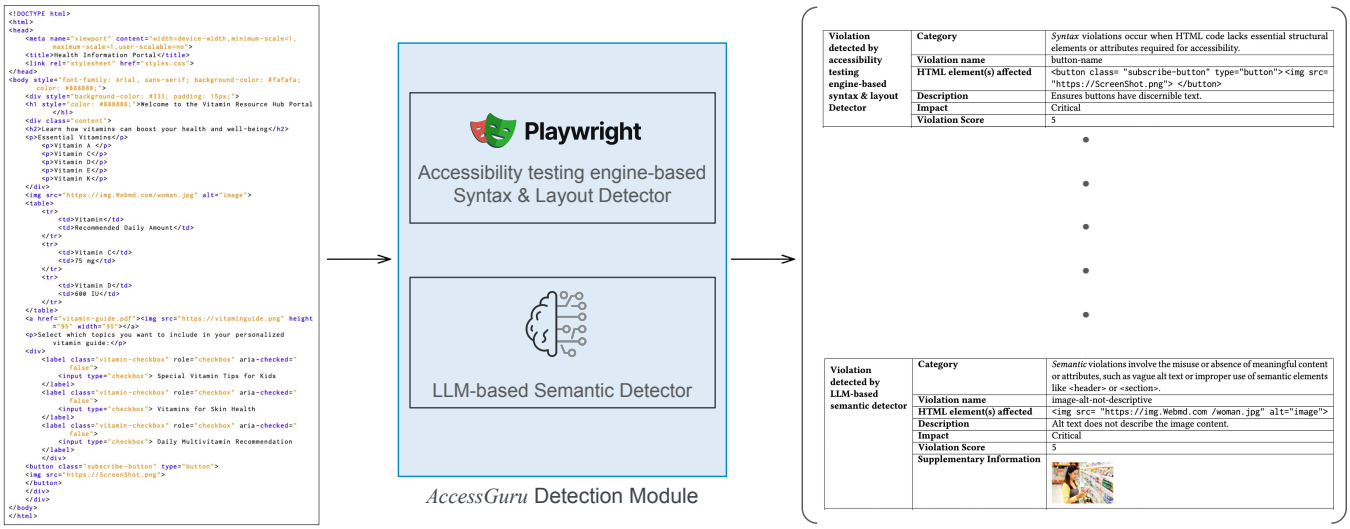
4.2 *AccessGuruCorrect*: Web Accessibility Violation Correction

AccessGuruCorrect operates on the output of the *AccessGuruDetect*—a set of detected Web accessibility violations—taking as input $V = \{v_1, v_2, \ldots, v_n\}$, where each v_i represents an individual We accessibility violation (see Figure 2).

Each violation includes the affected HTML element(s), violation metadata (See Tables 1 and 2 for sample inputs). The goal is to generate correct HTML for each violation that minimizes the total violation score, as shown in Figure 3 and detailed in Algorithm 1.

For each violation v, detected by *AccessGuruDetect* and enriched with violation-specific data such as supplementary information when required (e.g., images for `image-alt-not-descriptive`, videos for `video-caption-alt-not-descriptive`), we construct

Fig. 2. Overview of the *AccessGuruDetect*. Given a raw HTML document (left), it applies two detectors: (1) a syntax and layout detector based on the Axe-Playwright accessibility testing engine and (2) an LLM-based semantic detector. The output set of detected accessibility violations (right).



an *initial prompt* (Algorithm 1: line 3) and submit it to a pre-trained LLM (Algorithm 1: line 4). For semantic violations, the LLM must possess multimodal capabilities to reason over non-textual content (e.g., images for `image-alt-not-descriptive`, videos for `video-caption-alt-not-descriptive`).

The *initial prompt* integrates three techniques: (1) **role-play prompting**, (2) **contextual prompting**, and (3) **metacognitive prompting**. The role-play technique embeds a Web accessibility expert persona (See Table 9) to guide the LLM toward expert-like behavior. Contextual prompting enriches the prompt with violation-specific data— Web page URL, domain, WCAG guidelines, and a violation category description. Driven by our taxonomy, which defines the categories (*Syntactic*, *Semantic*, *Layout*) and their associated characteristics, we include the description of the violation category in each prompt. For example, layout violations may require CSS and responsive design awareness while preserving visual integrity for all users. These components are integrated using metacognitive prompting; we show the prompt template Table 10.

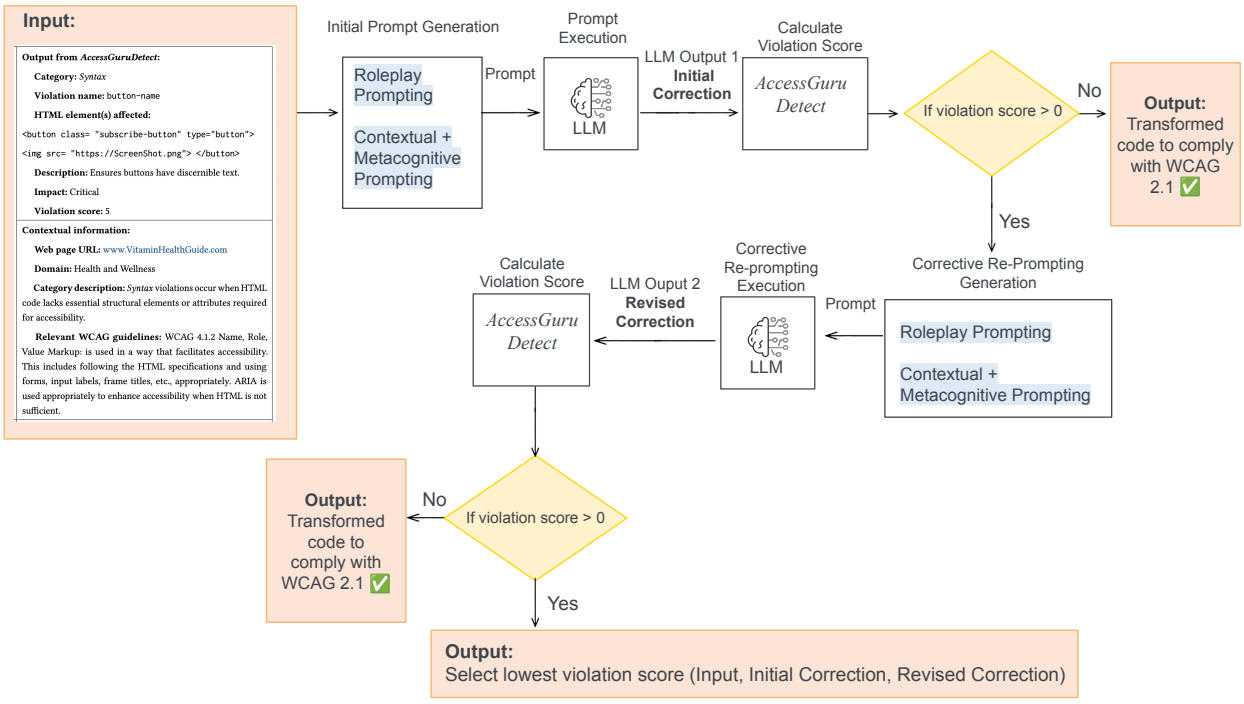
The LLM returns a correction (*LLMOutput*1), enclosed between string markers (e.g., ###START### and ###END###), as specified in the prompt. We extract the HTML snippet using regular expressions and evaluate it with *AccessGuruDetect* to assign a violation score (Algorithm 1 line 5). If the score is zero, the correction is accepted (Algorithm 1: line 6). Otherwise, we apply corrective re-prompting by adding feedback and resubmitting the prompt (Algorithm 1: lines 8–10), yielding a revised correction (*LLMOutput*2).

If the revised output still contains accessibility violations (Algorithm 1: line 11), we compare the scores of v, *LLMOutput*1, and *LLMOutput*2 (Algorithm 1: line 12). The version with the lowest score is selected as the final correction (Algorithm 1: line 13), with ties resolved by preferring the most recent output.

Our scoring mechanism is robust to new LLM-induced accessibility violations by treating all accessibility violations equally in the total violation score. For instance, if *LLMOutput*1 yields a violation score of 4, while *LLMOutput*2 introduces two accessibility violations scoring 1 and 2 (total 3), we prefer *LLMOutput*2. Conversely, if both LLM outputs score worse than the original input (e.g., *LLMOutput*1 violations score = 4 and *LLMOutput*2 violations score = 3 vs. v violations score = 2), we select the original code as the output.

The output is the correction generated by *AccessGuruCorrect*, which is appended to its corresponding JSON entry in the set, as shown in Tables 1 and 2. Resulting in a final JSON file that contains all detected accessibility violations along with their suggested corrections.

Fig. 3. Overview of *AccessGuruCorrect* in: For each Web accessibility violation detected by *AccessGuruDetect* (examples of the input accessibility violations are shown in Tables 1 and 2), the LLM is prompted to generate the corrected code. The generated code is assigned a violation score; if the violation score remains above zero, corrective re-prompting is applied to improve the response further.



Algorithm 1 *AccessGuruCorrect*

input: Set of detected accessibility violations $V = \{v_1, v_2, \ldots, v_n\}$
ouput: Set of corrected HTML segments $\{c_1, c_2, \ldots, c_n\}$
for each violation $v \in V$ **do**
    Construct *Initial Prompt*(v)
    Submit prompt to LLM to obtain $LLMOutput1$
    if $ViolationScore(LLMOutput1) = 0$ **then**
        $c \leftarrow LLMOutput1$
    else
        Construct *Corrective Prompt*(v)
        Submit prompt to LLM to obtain $LLMOutput2$
        if $ViolationScore(LLMOutput2) = 0$ **then**
            $c \leftarrow LLMOutput2$
        else
            $c \leftarrow SelectBest(v, LLMOutput1, LLMOutput2)$
        end if
    end if
end for
return $\{c_1, c_2, \ldots, c_n\}$

4.2.1 Correction Independence and Overwrites: AccessGuruCorrect applies corrections sequentially, generating a separate correction for each violation. Our taxonomy ensures that violations are non-overlapping, so each correction typically modifies a different attribute of the same HTML node. Consider the following example, where a single HTML node contains two violations:

```
<a href="subscribe.html" style="color:#767676;background:#303030;">
<img src="sub.png"> </a>
```

AccessGuru generates two independent corrections:

- **Violation A: `link-name`**

```
- <a href="subscribe.html" ...>
+ <a href="subscribe.html" aria-label="Subscribe" ...>
```

- **Violation B: `color-contrast`**

```
- style="color:#767676;background:#303030;"
+ style="color:#FFFFFF;background:#1A1A1A;"
```

Each correction targets a distinct part of the node and can be applied independently. However, rare edge cases may result in overlapping edits. For example:

```
<html lang="eng" xml:lang="en-GB">
```

AccessGuru suggests:

- **Violation A: `html-lang-valid`**

```
- lang="eng"
+ lang="en"
```

- **Violation B: `html-xml-lang-mismatch`**

```
- lang="eng"
+ lang="en-GB"
```

The second correction may overwrite the first; any overwrite, therefore, further refines the correction.

Table 1. Example JSON entry from *AccessGuru*'s output for a syntax violation from the HTML document in Listing 1 (Violation 9, Line 57). It consists of: (1) the violation v detected by *AccessGuru Detect*, including metadata; (2) contextual information; and (3) the correction generated by *AccessGuru Correct*.

Input to *AccessGuru Correct*	**Output from *AccessGuruDetect*:** **Category:** *Syntax* **Violation name:** `button-name` **HTML element(s) affected:** `<button class="subscribe-button" type="button"> <img src="https://ScreenShot.png"> </button>` **Description:** Ensures buttons have discernible text. **Impact:** Critical **Violation score:** 5
	Contextual information: **Web page URL:** www.VitaminHealthGuide.com **Domain:** Health and Wellness **Category description:** *Syntax* violations occur when HTML code lacks essential structural elements or attributes required for accessibility. **Relevant WCAG guidelines:** WCAG 4.1.2 Name, Role, Value Markup: is used in a way that facilitates accessibility. This includes following the HTML specifications and using forms, input labels, frame titles, etc., appropriately. ARIA is used appropriately to enhance accessibility when HTML is not sufficient.
Output from *AccessGuru Correct*	`<button class="subscribe" type="button" aria-label="Subscribe to Vitamin Newsletter"> <img src="https://ScreenShot.png" alt="Subscribe Button Image"> </button>`

Table 2. Example JSON entry from *AccessGuru*'s output for a semantic violation from Listing 1 (Violation 5, Line 25). It consists of: (1) the violation v output from *AccessGuru Detect*; (2) input to *AccessGuruCorrect*, which is the output from *AccessGuru Detect* along with the contextual information; and (3) the output correction generated by *AccessGuru Correct*.

Input to *AccessGuru Correct*	**Output from *AccessGuruDetect*:** **Category:** *Semantic* **Violation name:** `image-alt-not-descriptive` **HTML element(s) affected:** `<img src="https://img.Webmd.com/woman.jpg" alt="image">` **Description:** alt text does not describe the image content. **Impact:** Critical **Violation score:** 5 **Supplementary information:** 
	Contextual information: **Web page URL:** www.VitaminHealthGuide.com **Domain:** Health and Wellness **Category description:** *Semantic* violations involve the misuse or absence of meaningful content or attributes, such as vague alt text or improper use of semantic elements like `<header>` or `<section>`. **Relevant WCAG guidelines:** WCAG 1.1.1 Non-text Content. All meaningful visual elements (e.g., images, image buttons, image maps) must have descriptive alternative text. Form controls, inputs, multimedia, and frames must include accessible names, labels, or titles to ensure clarity for assistive technologies.
Output from *AccessGuru Correct*	`<img src="https://img.Webmd.com/woman.jpg" alt="Woman reading the label on a vitamin box while standing in a pharmacy aisle.">`

5 Evaluation

We evaluate the effectiveness of *AccessGuru* in detecting and correcting Web accessibility violations across the three categories defined in our taxonomy: *syntactic*, *semantic*, and *layout*. Specifically, we ask:

- Detection Evaluation: **RQ 1.** To what extent can our detection method identify accessibility violations across all three categories?
- Syntactic and Layout Correction Evaluation: **RQ 2.** To what extent can the LLM generate HTML code that satisfies syntactic and layout accessibility compliance, effectively addressing Web accessibility violations?
- Semantic Correction Evaluation: **RQ 3.** To what extent are the LLM-generated attributes semantically meaningful, as evaluated by human experts and in comparison to corrections made by human developers?

5.1 Dataset of Web Accessibility Violations from [23]

We use the dataset from [23] to evaluate whether *AccessGuruDetect* can successfully identify the reported violations and assess whether *AccessGuruCorrect* can transform violations into accessibility-compliant HTML. The dataset consists of Web accessibility violations collected from 25 URLs, including popular Websites such as Google Calendar, Slack, and BBC. The dataset contains 171 rows of accessibility violations, which were identified using the Axe-Playwright and span 40 distinct accessibility violation types. Each violation is categorized by severity (ranging from cosmetic to critical) and includes details such as the Web page URL, violation name, description, and HTML elements.

The dataset introduced by Huang et al. [23] represents a valuable step toward evaluating automated accessibility corrections. However, it remains limited in several important ways. The dataset is relatively small in scale and does not offer broad coverage of known Web accessibility violation types- 40 accessibility violation types. Moreover, upon manual inspection, we observed that the dataset provides only HTML for each violation, which is insufficient for certain accessibility violations. For example, addressing contrast violations requires access to the color values of foreground and background elements, which are often defined in external CSS and not reflected in the raw HTML. This omission restricts the ability to fully detect or correct such violations.

5.2 Our novel dataset for benchmarking corrections of syntactic, layout, and semantic accessibility violations

We introduce a new dataset that comprehensively covers syntactic, semantic, and layout accessibility violations, reflecting real-world accessibility violations. Our dataset was collected through a structured process guided by the WebAIM 2025 study [55], which identifies commonly visited websites with accessibility violations. To account for variation in accessibility violations across domains—e.g., government sites often use data tables, while e-commerce relies heavily on forms and images—we used GPT-4 [36] to identify a diverse set of popular Web domains such as health, education, government, news, technology, and e-commerce, including multilingual websites. We then asked GPT-4 to suggest representative URLs from each category based on sites listed in the WebAIM 2025 study [55]. This process yielded 448 URLs.

We crawled each URL using Playwright (see §5.4) and retained only those pages where `document.readyState === "complete"`. We applied *AccessGuruDetect* (Figure 2) to each URL to identify Web accessibility violations. This yielded 3,500 Web accessibility violations. The dataset spans over 112 distinct violation types across all three categories. To our knowledge, it is the most comprehensive publicly available dataset of real-world Web accessibility violations to date.

To ensure our evaluation reflects real-world Web accessibility violations, we sampled a representative subset of 305 violations from our full dataset of 3,500 instances. Sampling aligns the distribution of violation types in the subset with real-world frequencies reported in the WebAIM 2025 study [55]. This method avoids biases caused by overrepresented violation types in large-scale crawled data but does not reflect their actual prevalence across the Web. This subset size was chosen to enable controlled and consistent comparison across LLMs and baselines. Table 3 compares the distribution of violation categories in our subset with that of the dataset from [23].

Table 3. Distribution of accessibility violations in our dataset and dataset from [23]

Web Accessibility Violation Category	**# Violation in Our Sampled Dataset**	**# Violation in [23] Dataset**
Syntax	195	151
Layout	55	20
Semantic	55	0
Total	**305**	**171**
# Violation Types	**112**	**40**

5.3 Prompt-based baseline methods for accessibility violation correction

We compare *AccessGuru* to three other baselines, that use the following prompt engineering techinues to correct Web accessibility violations: contextual prompting [37], Re-Act prompting [23], and zero-shot prompting [15], using our sampled dataset and the dataset from Huang et al. [23] (See section 2.3).

5.4 Implementation

We implement the *AccessGuruDetect* using `Axe-Playwright-1.51.0` for syntax and layout accessibility violations and `GPT-4o` for semantic accessibility violations requiring multimodal reasoning capabilities to reason over Web page screenshot (See Section 4.1). For *AccessGuruCorrect*, we evaluate three LLMs on syntax and layout accessibility accessibility violations: `GPT-4-0125-preview`, `Mistral-7B-v0.1`, and `Qwen2.5-Coder`—a model optimized for coding tasks. These models vary in size, architecture, and training methods, enabling comparison across capabilities. To correct semantic accessibility violations requiring multimodal reasoning capabilities to reason over images (See Section 4.2 and Table 2), we use: `GPT-4`, `Pixtral-12B`, and `Qwen-VL`.

Handling Unreliable LLM Outputs: During the manual inspection of LLM-generated responses, we identified several reliability issues in LLM-generated corrections. Some outputs were incomplete, hallucinated unrelated HTML, especially for long or multilingual pages (e.g., Chinese). In other cases, the model returned only textual advice without any HTML code. To ensure a fair and robust evaluation across all methods, we apply consistency checks to verify that LLM-generated corrections are valid and complete. For our method, we explicitly instruct the LLM to enclose the corrected code between string markers (e.g., ###START### and ###END###), enabling reliable extraction via regular expressions. For baselines that do not follow this format, we extract the first valid HTML snippet from the response heuristically, using pattern matching for common HTML tags.

For all LLM responses in our experiments, we manually verify that the extracted HTML contains a structurally valid and complete correction. If the response is missing required elements, contains malformed markup, or consists solely of textual advice, we flag it as not fixed. In these cases, the original violation score of the violation v is used as the final output score avoid inflating performance metrics.

5.5 Evaluation Metrics

We evaluate the effectiveness of *AccessGuru* in detecting and correcting Web accessibility violations across the three categories. For each category, we employ category-specific metrics to assess whether the LLM-generated corrections resolve the detected accessibility violations, as measured by automated evaluation tools or manual validation.

5.5.1 Detection Evaluation Metrics, We evaluate accessibility violation detection performance by measuring the detected violation count, the total number of Web accessibility violations identified across syntactic, layout, and semantic categories.

5.5.2 Syntactic and Layout Correction Metrics, we evaluate syntactic and layout accessibility compliance based on the reduction in violation scores. To calculate the average violation score of the entire dataset R, we use Equation 1, where i represents the index of each violation and n denotes the total number of entries in the dataset.

$$R = \frac{1}{n} \sum_{i=1}^{n} ViolationScore(v_i) \tag{1}$$

To calculate the percentage improvement I in violation score, we use Equation 2, we compare the average violation scores before $R_{initial}$ and after R_{fix} applying a correction method.

$$I = 1 - \frac{R_{fix}}{R_{initial}} \tag{2}$$

5.5.3 Semantic Correction Metrics, we calculate the similarity between human-generated and LLM-generated attributes using Sentence-BERT cosine similarity to assess the semantic quality of LLM-generated corrections. Sentence-BERT is well-suited for this task as it captures semantic meaning [42], making it more robust than traditional word-overlap metrics such as BLEU [39] or ROUGE [28], which rely on exact n-gram matches. We then computed the average similarity score to evaluate the alignment between human and LLM-generated attributes. For instance, given an LLM-generated alt text attribute: *"A golden retriever playing with a ball in a grassy park."* We compare it to three human-generated variants:

- *"A dog fetching a ball in a green field."* (Similarity: 0.5986)
- *"A golden retriever running in the park with a toy."* (Similarity: 0.8364)
- *"A happy dog playing outdoors with a ball."* (Similarity: 0.6760)

The average similarity score across these responses is 0.7037, indicating a strong semantic alignment.

6 Experiments and Results

6.1 Detecting Accessibility Violations Experiments and Results (RQ1)

To evaluate **RQ1**—, we applied *AccessGuruDetect* to the HTML documents of the 16 URLs from the dataset by Huang et al.[23]. Table 4 compares the number of detected violations by *AccessGuru Detect* against those originally reported in the dataset [23], *AccessGuru Detect* outperforms the original dataset by additionally detecting 104 semantic and more layout violations (15 vs. 8).

Although both methods use Axe-Playwright for detecting syntax and layout violations, *AccessGuruDetect* reports fewer syntax (82 vs. 118) and more layout violations (15 vs. 8). To investigate this discrepancy, we refer to the counts reported in [23], which help explain the decrease in violations detected by automatic tools. Since that dataset was collected in early 2024, some changes in reported violations are expected due to the dynamic nature of Web content. This interpretation is consistent with broader trends: according to the WebAIM Million study [55], Web accessibility has seen incremental improvements over the past year. For example, low color contrast violations decreased from 81% in 2024 to 79.1% in 2025.

Table 4. Comparison of accessibility violation detection coverage on 16 Web pages from the dataset of Huang et al. [23].

Web Accessibility Violation Category	Detected by [23]	Detected by *AccessGuruDetect*
Syntax	118	82
Layout	8	15
Semantic	0	104
Total	**126**	**201**

6.2 Syntactic and Layout Correction Experiments and Results (RQ2)

To evaluate **RQ2**—, we conduct three analyses: (1) a comparison against three correction baselines, (2) a cross-LLM evaluation, and (3) an ablation study to assess the impact of our corrective re-prompting strategy.

Baselines Comparison. We compare *AccessGuru* with three prompting-based baselines: contextual prompting [37], ReAct prompting [23], and zero-shot prompting [15]. As shown in Table 5, *AccessGuru* consistently achieves the highest violation score decrease and number of corrected violations on both our dataset and the Huang et al. dataset [23]. With GPT-4, *AccessGuru* reduces violation scores by 0.84 (204 corrections) on our dataset, significantly outperforming ReAct and contextual prompting (0.50 and 0.46, respectively). On the Huang et al. dataset with GPT-4, *AccessGuru* reduces violation scores by 0.83 (141 corrections), significantly outperforming ReAct and contextual prompting (0.48 and 0.42, respectively).

Cross-LLM Comparison. AccessGuru was tested with GPT-4, Qwen2.5, and Mistral-7B. GPT-4 consistently achieved the best performance across all correction tasks as shown in Table 5.

Ablation Study. To assess the impact of the corrective re-prompting strategy, we executed *AccessGuru* without the re-prompting phase using GPT-4. Performance dropped from 0.84 to 0.72 on our dataset, confirming the added value of this component. Even without it, *AccessGuru* outperforms the best-performing baseline, Re-Act prompting (0.50). This pattern holds consistently across all LLMs evaluated.

Qualitative Analysis. During the manual inspection of the results, we observed that all three baselines often provided incomplete solutions or adopted an "Occam's Razor" approach, where problematic elements were removed rather than properly corrected. This was frequently observed in long HTML snippets; when asked to correct an HTML snippet with sixteen elements, the output would only contain nine. In contrast, our method did not exhibit these issues. By asking the LLM to generate the code between string markers (###START### and ###END###) in our prompts as shown in Table 10, we improved the model's ability to provide complete solutions. We also observed that baseline methods would occasionally resolve color contrast accessibility violations by changing both background and foreground colors to black and white. While this resolves the accessibility violation for users with diverse abilities, it distorts the visual design and layout for normal users. This behavior was not observed in our method's results, as our prompt template—shown in Table 10—was driven by our proposed taxonomy. The taxonomy entails that correcting layout accessibility violations shouldn't distort the visual design and layout for all users. We attribute the notably poor performance of the zero-shot prompting baseline [15] to the nature of its prompt design shown in Table 12. The prompt asked, "`Is the following HTML code accessible?`" without distinguishing between detection and correction tasks. As a result, we observed that the LLM often failed to recognize existing accessibility violations, and in cases where it did, it sometimes responded with only a confirmation that a violation was present, without generating the corrected code. This ambiguity in the prompt limited the effectiveness of the zero-shot approach across both datasets. Additionally, the lack of explicit reference to accessibility guidelines often leads to corrections that don't comply with recognized standards.

6.3 Semantic Correction Experiments and Results (RQ3)

To evaluate **RQ3**—, we conducted two complementary evaluations: (1) human annotation to assess WCAG compliance and (2) a developer correction study to compare LLM corrections with those of human experts.

Human Annotation for WCAG Compliance. Two human annotators with five years of web development and accessibility experience reviewed corrections for 55 semantic violations from our sampled dataset. The human annotators assigned a violation

Table 5. Comparison of violation score decrease and number of corrected accessibility violations for **syntax & layout** violations on our dataset and the dataset from [23].

Method	Model	Our Dataset (Size=250)		Huang et al. Dataset [23] (Size=171)	
		Avg. Violation Score Decrease	# Corrected Violations	Avg. Violation Score Decrease	# Corrected Violations
Contextual prompting [37]	GPT-4	0.46	123	0.42	86
ReAct prompting [23]	GPT-4	0.50	141	0.48	91
Zero-shot prompting [15]	GPT-4	0.19	43	0.12	19
AccessGuru w/o reprompting (Ours)	GPT-4	0.72	184	0.67	119
***AccessGuru*(Ours)**	GPT-4	**0.84**	**204**	**0.83**	**141**
Contextual prompting [37]	Mistral-7B	0.12	44	0.27	62
ReAct prompting [23]	Mistral-7B	0.13	45	0.26	48
Zero-shot prompting [15]	Mistral-7B	0.05	10	0.002	2
AccessGuru w/o reprompting (Ours)	Mistral-7B	0.50	162	0.51	110
***AccessGuru* (Ours)**	Mistral-7B	**0.82**	**200**	**0.79**	**147**
Contextual prompting [37]	Qwen2.5	0.41	121	0.39	77
ReAct prompting [23]	Qwen2.5	0.44	130	0.37	71
Zero-shot prompting [15]	Qwen2.5	0.14	54	0.19	49
AccessGuru w/o reprompting (Ours)	Qwen2.5	0.49	153	0.52	103
***AccessGuru* (Ours)**	Qwen2.5	**0.74**	**183**	**0.75**	**126**

score of 0 if the correction fully resolved the accessibility violation according to WCAG 2.1; otherwise, the original violation score assigned during the detection of the semantic violation was retained. For example, if an image originally had the alt text "image" (violation score 5), and the LLM corrected it to a descriptive alt text, the violation was considered resolved, and the annotator assigned the correction a score of 0. If the correction is still vague (e.g., "image alt text"), the human annotator assigns a correction violation score of 5. We then computed the average violation score decrease across all samples—i.e., how much the violation score was reduced after correction. As shown in Table 6, *AccessGuru* with GPT-4 achieved the highest average violation score decrease (0.96), resolving 53 out of 55 violations. This outperformed both ReAct prompting (0.87) and contextual prompting (0.82), confirming the effectiveness of our approach for semantic correction.

Qualitative Analysis. We also manually reviewed the semantic correction quality across GPT-4, Qwen-VL, and Pixtral, each showing distinct strengths. We found that explicitly instructing the model to consider the screenshot in the prompt—shown in Tables 10, 11,12—was critical for eliciting image-grounded responses across all models. GPT-4 offered the most balanced performance, accurately integrating HTML structure and visual cues with minimal hallucinations. Qwen-VL showed strong image-based reasoning but frequently hallucinated additional HTML or introduced unrelated structures, especially. Pixtral, by contrast, preserved original HTML faithfully and avoided unnecessary changes but often failed to ground its corrections in the provided image. For example, when prompted to generate alt text for the HTML5 logo, it produced generic attributes such as: (1) "an orange and white SVG graphic," rather than a meaningful description of the image's identity—e.g., "the HTML5 logo."

Comparison with Human Developer Corrections. To assess how closely *AccessGuru* aligns with human correction behavior, we conducted a human developer correction study. This human developer correction study builds on findings from prior work [17], which demonstrated that LLMs are capable of generating semantic corrections. In particular, that study showed that GPT-based models could effectively address the `video-caption-not-descriptive` violation with corrections comparable in quality to those written by human annotators. Three full-stack developers independently corrected the same 55 semantic violations. Each developer received the HTML, violation metadata, and WCAG guidance—identical to what *AccessGuru* receives. We measured the similarity between LLM- and human-generated corrections using Sentence-BERT cosine similarity. As shown in Table 7, *AccessGuru* (GPT-4) achieved an average semantic similarity score of 0.77 when compared to human-generated corrections, indicating that the LLM-produced outputs closely matched the phrasing, structure, and meaning of human-written solutions.

Table 6. Comparison of violation score decrease and number of corrected accessibility violations for **semantic** violations on our dataset (Size=55).

Method	Model	Avg. Violation Score Decrease	# Corrected Violations
Contextual prompting [37]	GPT-4	0.82	44
ReAct prompting [23]	GPT-4	0.87	48
Zero-shot prompting [15]	GPT-4	0.33	18
AccessGuru w/o reprompting (Ours)	GPT-4	0.92	51
***AccessGuru*(Ours)**	GPT-4	**0.96**	**53**
Contextual prompting [37]	Pixtral	0.75	41
ReAct prompting [23]	Pixtral	0.81	44
Zero-shot prompting [15]	Pixtral	0.54	29
AccessGuru w/o reprompting (Ours)	Pixtral	0.83	46
***AccessGuru*(Ours)**	Pixtral	**0.92**	**51**
Contextual prompting [37]	Qwen-VL	0.60	32
ReAct prompting [23]	Qwen-VL	0.37	20
Zero-shot prompting [15]	Qwen-VL	0.18	8
AccessGuru w/o reprompting (Ours)	Qwen-VL	0.69	37
***AccessGuru*(Ours)**	Qwen-VL	**0.75**	**41**

Table 7. Sentence-BERT similarity between *AccessGuru* (GPT-4) and human corrections across semantic violation categories

Violation Category	# Violations	Avg. Similarity
image-alt-not-descriptive	6	0.83
lang-mismatch	18	0.84
link-text-mismatch	11	0.68
form-label-mismatch	5	0.70
ambiguous-heading	4	0.68
page-title-not-descriptive	3	0.86
button-label-mismatch	8	0.83
Average Across 55 Violations		**0.77**

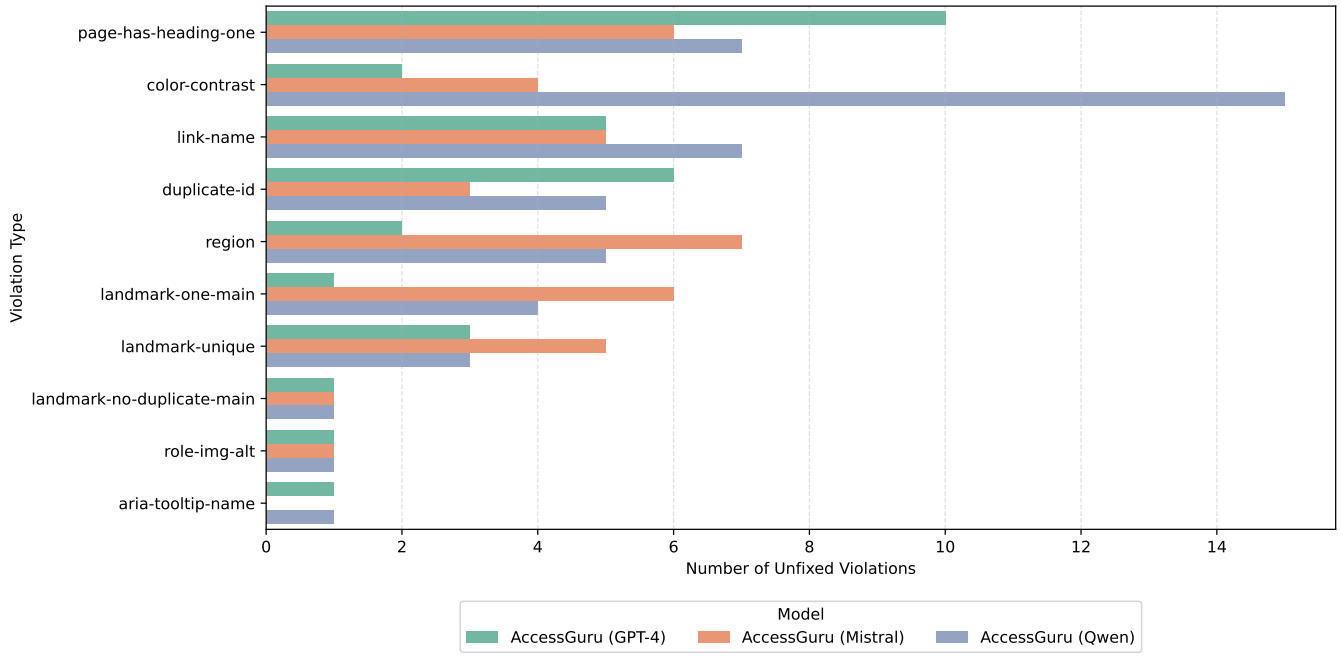
While overall similarity scores indicate strong semantic alignment, certain categories—most notably `link-text-mismatch` and `form-label-mismatch`—had lower scores. For `link-text-mismatch`, this is likely due to the LLM's inability to access the target of hyperlinks, especially when the destination is a downloadable file (e.g., `<a href="rep123.pdf">Click here</a>`). Without knowing the link's content, the model cannot generate a descriptive label like `Download quarterly report`. In contrast, when the link points to another section of the same page (e.g., `#contact`), the LLM performs better by using the surrounding context. In the case of `form-label-mismatch`, the LLM sometimes fails to infer connections between labels and input fields.

A full comparison of all methods across LLMs and violation types is provided in Tables 5 and 6, showing that *AccessGuru* consistently outperforms baselines on both datasets and across syntactic, layout, and semantic accessibility violations.

7 Limitations

We have shown that *AccessGuru* works conceptually and have evaluated it thoroughly. However, several non-trivial engineering efforts have not yet been integrated into our current implementation. These include the reconstruction of a fully corrected HTML document. While our correction module outputs individual corrected segments per violation, reintegrating these into the original document requires resolving potential conflicts, particularly when multiple overlapping corrections affect nested or related elements. For instance, corrections to a list and an image within a list item must be merged carefully to preserve the semantic structure.

Fig. 4. Breakdown of top 10 uncorrected Web accessibility violations by AccessGuru across three LLMs (GPT-4, Mistral, and Qwen).



Another limitation is the reliability of the LLM-based semantic detector; in our observations, the LLM occasionally hallucinated accessibility violations or misidentified affected elements due to a lack of grounding in HTML attributes or structures. Long HTML documents pose challenges; they increase the prompt length, potentially overwhelming the model's context window and leading to incomplete reasoning or fabricated results. The LLM-based semantic detector relies on a static screenshot of the web page, which doesn't capture dynamic content or alternate views such as drop-down menus, pop-ups, or language toggles.

While we manually verified the correctness of detected violations, we did not establish the complete set of ground-truth violations for each Web page. As a result, we do not report recall or precision metrics. This limits our ability to quantify undetected violations and fully assess detection completeness.

Finally, correcting accessibility violations is exceptionally difficult, as it requires a nuanced understanding of user needs, Web design principles, and advanced reasoning capabilities to ensure effective correction [8, 14]. To better understand which violations remained uncorrected, we analyzed the corrections produced by three *AccessGuru* models—GPT-4, Mistral, and Qwen. As shown in Figure 4, certain violations such as page-has-heading-one, color-contrast, and link-name were consistently difficult to correct. A limitation of *AccessGuru* is that it corrects color-contrast accessibility violations by adjusting foreground and background values to meet WCAG thresholds only. However, it does not account for cases where color is used to convey meaning, such as red for errors or green for success. In such scenarios, even after contrast improvements, users with color vision deficiencies may still struggle to understand the intended message. Future work should investigate how visual cues like color can be supplemented with alternative representations (e.g., icons, text labels, or ARIA attributes) to ensure the semantic meaning is preserved for all users.

8 Conclusion and Future work

Our proposed solution, *AccessGuru*, helps to improve Web accessibility by detecting and correcting accessibility violations in HTML documents. *AccessGuru* combines automated evaluation tools with prompting strategies for pre-trained LLMs to generate compliant corrections. *AccessGuru Detect* is evaluated by comparing its accessibility violation coverage against existing accessibility detection datasets. *AccessGuruCorrect* is evaluated on two datasets using three different LLMs, with additional human studies to assess the semantic quality of the corrections compared to human-generated solutions. Our benchmark extends beyond automatic evaluation tools' conformance by addressing semantic accessibility violations. Future work should follow this direction by also

moving beyond conformance checks and incorporating measures like task completion and usability, which automatic evaluation tools alone do not fully capture.

An important future direction is the reconstruction of fully corrected Web pages. While our system outputs individual corrected HTML snippets per violation, these must be re-integrated into the original document to form a coherent and fully accessible Web page. This reconstruction process involves merging overlapping or nested corrections while preserving the semantic structure, layout, and any unaffected content. We intend to address this engineering effort in future iterations of our system.

Additionally, future research should explore more robust grounding techniques, multi-view analysis of dynamic content—including the use of video to capture temporal changes—and strategies to handle long-context scenarios—such as hierarchical analysis or chunked evaluation—to improve the effectiveness of LLMs in detecting semantic accessibility violations. Finally, accessible Web pages available in the wild could be leveraged in future work to help LLMs retrieve compliant examples and apply similar patterns in corrections.

9 Acknowledgements

We acknowledge the support of the Stuttgart Research Focus Interchange Forum for Reflection on Intelligent Systems (IRIS).

References

[1] AChecker: IDI Accessibility. [n. d.]. AChecker: IDI Accessibility. http://www.atutor.ca/achecker/. Retrieved 25-06-2024.

[2] Patricia Acosta-Vargas, Mario González, and Sergio Luján-Mora. 2020. Dataset for evaluating the accessibility of the websites of selected Latin American universities. *Data in brief* 28 (2020), 105013.

[3] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 2655–2668. doi:10.18653/V1/2021.NAACL-MAIN.211

[4] Wajdi Aljedaani, Abdulrahman Habib, Ahmed Aljohani, Marcelo Eler, and Yunhe Feng. 2024. Does ChatGPT Generate Accessible Code? Investigating Accessibility Challenges in LLM-Generated Source Code. In *Proceedings of the 21st International Web for All Conference, W4A 2024, Singapore, May 13-14, 2024*. ACM, 165–176. doi:10.1145/3677846.3677854

[5] Suliman K. Almasoud and Hassan I. Mathkour. 2019. Instant Adaptation Enrichment Technique to Improve Web Accessibility for Blind Users. In *Proceedings of the 3rd International Conference on Information System and Data Mining, ICISDM 2019, Houston, TX, USA, April 6-8, 2019*. ACM, 159–164. doi:10.1145/3325917.3325931

[6] Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek. 2020. Accessibility issues in Android apps: state of affairs, sentiments, and ways forward. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 1323–1334. doi:10.1145/3377811.3380392

[7] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *CoRR* abs/2108.07732 (2021). arXiv:2108.07732 https://arxiv.org/abs/2108.07732

[8] Ana Baptista, José Martins, Ramiro Goncalves, Frederico Branco, and Tania Rocha. 2016. Web accessibility challenges and perspectives: A systematic literature review. In *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 1–6.

[9] Tingting Bi, Xin Xia, David Lo, John C. Grundy, Thomas Zimmermann, and Denae Ford. 2022. Accessibility in Software Practice: A Practitioner's Perspective. *ACM Trans. Softw. Eng. Methodol.* 31, 4 (2022), 66:1–66:26. doi:10.1145/3503508

[10] Ben Caldwell, Michael Cooper, Loretta Guarino Reid, Gregg Vanderheiden, Wendy Chisholm, John Slatin, and Jason White. 2008. Web content accessibility guidelines (WCAG) 2.0. *WWW Consortium (W3C)* 290, 1-34 (2008), 5–12.

[11] Carmine Cesarano, Anna Rita Fasolino, and Porfirio Tramontana. 2007. Improving Usability of Web Pages for Blinds. In *Proceedings of the 9th IEEE International Symposium on Web Systems Evolution, WSE 2009, 5-6 October 2007, Paris, France*, Shihong Huang and Massimiliano Di Penta (Eds.). IEEE Computer Society, 97–104. doi:10.1109/WSE.2007.4380250

[12] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Unblind your apps: predicting natural-language labels for mobile GUI components by deep learning. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 322–334. doi:10.1145/3377811.3380327

[13] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondá de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri swear here Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Cast, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[14] Jenny Craven. 2006. Web accessibility: A review of research and initiatives. (2006).

[15] Giovanni Delnevo, Manuel Andruccioli, and Silvia Mirri. 2024. On the Interaction with Large Language Models for Web Accessibility: Implications and Challenges. In *21st IEEE Consumer Communications & Networking Conference, CCNC 2024, Las Vegas, NV, USA, January 6-9, 2024*. IEEE, 1–6. doi:10.1109/CCNC51664.2024.10454680

[16] Iyad Abu Doush and Reem Qasem. 2024. Evaluating AI-Generated Web Code for Accessibility Compliance: A Metric-Driven Approach. In *Proceedings of Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion (DSAI '24)*. ACM.

[17] Nadeen Fathallah, Monika Bhole, and Steffen Staab. 2024. Empowering the Deaf and Hard of Hearing Community: Enhancing Video Captions Using Large Language Models. In *Proceedings of Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion (DSAI '24)*. ACM, New York, USA, 1–9. doi:10.48550/arXiv.2412.00342

[18] Nadeen Fathallah, Arunav Das, Stefano De Giorgis, Andrea Poltronieri, Peter Haase, and Liubov Kovriguina. 2024. NeOn-GPT: A Large Language Model-Powered Pipeline for Ontology Learning. In *The Extended Semantic Web Conference*.

[19] Mexhid Ferati and Lirim Sulejmani. 2016. Automatic Adaptation Techniques to Increase the Web Accessibility for Blind Users. In *HCI International 2016 - Posters' Extended Abstracts - 18th International Conference, HCI International 2016, Toronto, Canada, July 17-22, 2016, Proceedings, Part II (Communications in Computer and Information Science, Vol. 618)*, Constantine Stephanidis (Ed.). Springer, 30–36. doi:10.1007/978-3-319-40542-1_5

[20] Stephen M Fleming and Hakwan C Lau. 2014. How to measure metacognition. *Frontiers in human neuroscience* 8 (2014), 443.

[21] Chris D Frith. 2012. The role of metacognition in human social interactions. *Philosophical Transactions of the Royal Society B: Biological Sciences* 367, 1599 (2012), 2213–2223.

[22] Xiaodong Gu, Kang Min Yoo, and Sang-Woo Lee. 2021. Response Generation with Context-Aware Prompt Learning. *CoRR* abs/2111.02643 (2021). arXiv:2111.02643 https://arxiv.org/abs/2111.02643

[23] 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. *CoRR* abs/2401.16450 (2024). doi:10.48550/ARXIV.2401.16450 arXiv:2401.16450

[24] Andrew Kirkpatrick, Joshue O'Connor, Alastair Campbell, and Michael Cooper. 2023. *Web Content Accessibility Guidelines (WCAG) 2.1.* Technical report. World Wide Web Consortium (W3C). https://www.w3.org/TR/2023/REC-WCAG21-20230921/

[25] Satwik Ram Kodandaram, Utku Uckun, Xiaojun Bi, IV Ramakrishnan, and Vikas Ashok. 2024. Enabling Uniform Computer Interaction Experience for Blind Users through Large Language Models. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility*. 1–14.

[26] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. 2024. 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), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (Eds.). Association for Computational Linguistics, 4099–4113. doi:10.18653/V1/2024.NAACL-LONG.228

[27] Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020. Widget Captioning: Generating Natural Language Description for Mobile User Interface Elements. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 5495–5510. doi:10.18653/V1/2020.EMNLP-MAIN.443

[28] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[29] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html

[30] 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.

[31] Andrea Mangiatordi and Marco Lazzari. 2018. Combined use of artificial intelligence and crowdsourcing to provide alternative content for images on websites. In *15th IEEE Annual Consumer Communications & Networking Conference, CCNC 2018, Las Vegas, NV, USA, January 12-15, 2018*. IEEE, 1–6. doi:10.1109/CCNC.2018.8319312

[32] Shridhar Mehendale and Ankit Walishetti. 2024. DexAssist: A Voice-Enabled Dual-LLM Framework for Accessible Web Navigation. *arXiv preprint arXiv:2411.12214* (2024).

[33] Microsoft. 2024. Playwright API. https://playwright.dev. Accessed: 2024-06-30.

[34] Daye Nam, Andrew Macvean, Vincent J. Hellendoorn, Bogdan Vasilescu, and Brad A. Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 97:1–97:13. doi:10.1145/3597503.3639187

[35] Ahmed Njifenjou, Virgile Sucal, Bassam Jabaian, and Fabrice Lefèvre. 2024. Role-Play Zero-Shot Prompting with Large Language Models for Open-Domain Human-Machine Conversation. *CoRR* abs/2406.18460 (2024). doi:10.48550/ARXIV.2406.18460 arXiv:2406.18460

[36] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). doi:10.48550/ARXIV.2303.08774 arXiv:2303.08774

[37] 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, PETRA 2023, Corfu, Greece, July 5-7, 2023*. ACM, 707–713. doi:10.1145/3594806.3596542

[38] Letícia Seixas Pereira, João Guerreiro, André Rodrigues, Tiago João Guerreiro, and Carlos Duarte. 2024. From Automation to User Empowerment: Investigating the Role of a Semi-automatic Tool in Social Media Accessibility. *ACM Trans. Access. Comput.* 17, 3 (2024), 13:1–13:25. doi:10.1145/3647643

[39] Matt Post. 2018. A call for clarity in reporting BLEU scores. *arXiv preprint arXiv:1804.08771* (2018).

[40] Shreyas Sundara Raman, Vanya Cohen, Eric Rosen, Ifrah Idrees, David Paulius, and Stefanie Tellex. 2022. Planning with Large Language Models via Corrective Re-prompting. *CoRR* abs/2211.09935 (2022). doi:10.48550/ARXIV.2211.09935 arXiv:2211.09935

[41] Yongming Rao, Wenliang Zhao, Guangyi Chen, Yansong Tang, Zheng Zhu, Guan Huang, Jie Zhou, and Jiwen Lu. 2022. DenseCLIP: Language-Guided Dense Prediction with Context-Aware Prompting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 18061–18070. doi:10.1109/CVPR52688.2022.01755

[42] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[43] Richard Rutter, Patrick H Lauke, Cynthia Waddell, Jim Thatcher, Shawn Lawton Henry, Bruce Lawson, Andrew Kirkpatrick, Christian Heilmann, Michael R Burks, Bob Regan, et al. 2007. *Web accessibility: Web standards and regulatory compliance*. Apress.

[44] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *CoRR* abs/2402.07927 (2024). doi:10.48550/ARXIV.2402.07927 arXiv:2402.07927

[45] Elliot Salisbury, Ece Kamar, and Meredith Ringel Morris. 2017. Toward Scalable Social Alt Text: Conversational Crowdsourcing as a Tool for Refining Vision-to-Language Technology for the Blind. In *Proceedings of the Fifth AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2017, 23-26 October 2017, Québec City, Québec, Canada*, Steven Dow and Adam Tauman Kalai (Eds.). AAAI Press, 147–156. doi:10.1609/HCOMP.V5I1.13301

[46] Murray Shanahan, Kyle McDonell, and Laria Reynolds. 2023. Role play with large language models. *Nat.* 623, 7987 (2023), 493–498. doi:10.1038/S41586-023-06647-8

[47] Darja Šmite, Claes Wohlin, Zane Galviņa, and Rafael Prikladnicki. 2014. An empirically based terminology and taxonomy for global software engineering. *Empirical Software Engineering* 19 (2014), 105–153.

[48] Dat Trinh Tuan, Van-Hung Phan, et al. 2012. Checking and correcting the source code of web pages for accessibility. In *2012 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future*. IEEE, 1–4.

[49] Muhammad Usman, Ricardo Britto, Jürgen Börstler, and Emilia Mendes. 2017. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. *Information and Software Technology* 85 (2017), 43–59.

[50] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI '22: CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, 29 April 2022 - 5 May 2022, Extended Abstracts*, Simone D. J. Barbosa, Cliff Lampe, Caroline Appert, and David A. Shamma (Eds.). ACM, 332:1–332:7. doi:10.1145/3491101.3519665

[51] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 3156–3164. doi:10.1109/CVPR.2015.7298935

[52] Noah Wang, Z. y. Peng, Haoran Que, Jiaheng Liu, Wangchunshu Zhou, Yuhan Wu, Hongcheng Guo, Ruitong Gan, Zehao Ni, Jian Yang, Man Zhang, Zhaoxiang Zhang, Wanli Ouyang, Ke Xu, Wenhao Huang, Jie Fu, and Junran Peng. 2024. RoleLLM: Benchmarking, Eliciting, and Enhancing Role-Playing Abilities of Large Language Models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 14743–14777. doi:10.18653/V1/2024.FINDINGS-ACL.878

[53] Yuqing Wang and Yun Zhao. 2023. Metacognitive Prompting Improves Understanding in Large Language Models. *CoRR* abs/2308.05342 (2023). doi:10.48550/ARXIV.2308.05342 arXiv:2308.05342

[54] WebAIM. 2025. WAVE. https://wave.webaim.org/api/. Retrieved 2025-01-11.

[55] WebAIM. 2025. *The WebAIM Million - An Annual Accessibility Analysis of the Top 1,000,000 Home Pages.* Technical Report. WebAIM.org. https://webaim.org/projects/million/

[56] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. *CoRR* abs/2302.11382 (2023). doi:10.48550/ARXIV.2302.11382 arXiv:2302.11382

[57] Shaomei Wu, Jeffrey Wieland, Omid Farivar, and Julie Schiller. 2017. Automatic Alt-text: Computer-generated Image Descriptions for Blind Users on a Social Network Service. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, CSCW 2017, Portland, OR, USA, February 25 - March 1, 2017*, Charlotte P. Lee, Steven E. Poltrock, Louise Barkhuus, Marcos Borges, and Wendy A. Kellogg (Eds.). ACM, 1180–1192. doi:10.1145/2998181.2998364

[58] Yeliz Yesilada, Giorgio Brajnik, Markel Vigo, and Simon Harper. 2012. Understanding web accessibility and its drivers. In *International Cross-Disciplinary Conference on Web Accessibility, W4A '12, Lyon, France, April 16-17, 2012*, Markel Vigo, Julio Abascal, Rui Lopes, and Paola Salomoni (Eds.). ACM, 19. doi:10.1145/2207016.2207027

[59] Yeliz Yesilada and Simon Harper (Eds.). 2019. *Web Accessibility - A Foundation for Research, Second Edition.* Springer. doi:10.1007/978-1-4471-7440-0

[60] Yujia Zhou, Zheng Liu, Jiajie Jin, Jian-Yun Nie, and Zhicheng Dou. 2024. Metacognitive Retrieval-Augmented Large Language Models. In *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, Tat-Seng Chua, Chong-Wah Ngo, Ravi Kumar, Hady W. Lauw, and Roy Ka-Wei Lee (Eds.). ACM, 1453–1463. doi:10.1145/3589334.3645481

A Appendix

This appendix complements our methodology by providing prompting templates and a subset of our proposed taxonomy for categorizing Web accessibility violations.

Table 8. Prompt template used in the LLM-based semantic detector within the *AccessGuruDetect* module. The prompt guides the LLM to **detect semantic Web accessibility violations**. We structure the prompt into fixed and dynamic components. Fixed components remain constant across all HTML documents, while dynamic fields are populated based on the specific web page input.

Type	Prompt Template
Fixed	You are a Web accessibility expert. Your task is to detect semantic accessibility violations in the given HTML Web page. These accessibility violations are often not detectable by standard automated tools and require interpretation of the content's meaning and user context.
Fixed	A semantic violation occurs when: - Attributes like alt text, language, or link/button labels are present but do not provide meaningful information. - Visual or multimedia content is not described in a way that conveys its purpose to users with disabilities.
Fixed	Use the information below to guide your analysis. You are operating on: - The domain of the web page:
Dynamic	`{Insert Web page Domain}`
Fixed	- The URL of the web page:
Dynamic	`{Insert Web page URL}`
Fixed	You are provided with: - The HTML code of the web page to analyze. - The full semantic accessibility violation taxonomy. This taxonomy defines specific types of semantic accessibility violations and their descriptions. `[Semantic Accessibility Violation Taxonomy]` - A screenshot of the rendered view of the web page.
Dynamic	`{Insert HTML here}`
Dynamic	`{Insert Web page screenshot}`
Fixed	Now, review the HTML and supplementary data. List all semantic accessibility violations you detect, and for each: 1. Identify the affected HTML element. Enclose the exact HTML snippet using the markers `[START]` and `[END]`. 2. Specify the violation name.

Table 9. **Role-play persona** used in the *AccessGuruCorrect* module to guide the LLM in generating WCAG-compliant HTML corrections.

Persona
You are a Web accessibility expert with a strong proficiency in HTML and a deep commitment to fixing Web accessibility violations. You specialize in analyzing Web pages, identifying accessibility violations, and providing immediate, corrected HTML code solutions that meet WCAG 2.1 standards. Your expertise includes resolving problems like missing or improper alt text, insufficient heading structure, non-semantic elements, inaccessible forms, and color contrast accessibility violations. You are adept at transforming flawed code into compliant, clean HTML that works seamlessly with assistive technologies, ensuring that Websites are fully navigable by keyboard and readable by screen readers. You provide the corrected code necessary for immediate implementation, ensuring that Websites are not only compliant but truly inclusive for users with disabilities. Your mission is to ensure that every website and application is accessible to all users by providing expertly corrected HTML, making the web a more inclusive space.

Table 10. **Initial prompt template** used in the *AccessGuruCorrect* module. The prompt integrates role-play, contextual, and metacognitive prompting strategies and is structured around five metacognitive stages: comprehension clarification, preliminary judgment, critical evaluation, decision confirmation, and confidence assessment. The prompt is structured into fixed and dynamic components. The fixed components remain constant across all samples, while dynamic fields are populated based on the violation instance. (For semantic violations involving visual content, an additional instruction guides the LLM to reason over webpage screenshots—see purple-highlighted rows.)

Metacognitive Stage	Type	Prompt Template
Comprehension Clarification	Fixed	`{Role-play persona}` + Clarify your understanding of the following web accessibility violation:

	Dynamic	`{Category}`
		`{Category description}`
		`{Violation name}`
		`{Violation description}`
		`{URL}`
		`{HTMLElement}`
		`{Impact}`

	Fixed	Impact is a rating determined by the severity of the violation, indicating the extent to which it hinders user interaction with the Web content. The scale is [cosmetic, minor, moderate, serious, critical]
	Fixed	Prioritize the attached screenshot of the Web page (which visually shows the UI element with a possible Web accessibility violation). Your tasks: (1) Interpret the visual content of the attached image. (2) Identify the UI element (e.g., a button or icon) shown in the image. (3) Determine whether the element is accessible (i.e., if an image element has a meaningful alt text) (4) Compare your findings with the corresponding HTML provided and highlight any mismatches. (5) Suggest an accessibility-compliant fix if there's a violation.
	Dynamic	`{Web page screenshot}`
Preliminary Judgment	Fixed	Based on your understanding, provide a preliminary correction for the web accessibility violation based on the following WCAG guideline(s):
	Dynamic	`{Relevant WCAG guideline}`
	Fixed	Make sure your generated code corrects the web accessibility violation without introducing new accessibility violations. Ensure you generate the complete corrected code, not just a snippet.
Critical Evaluation	Fixed	Critically assess your preliminary correction, make sure to correct the initial web accessibility violation without introducing new web accessibility violations. Only make corrections if the previous answer is incorrect. Make sure your generated code corrects the web accessibility violation without introducing new accessibility violations.
Decision Confirmation	Fixed	Confirm your final decision on whether the correction is accurate or not and provide the reasoning for your decision. Only suggest further corrections if the initial response contains errors. Make sure your generated code corrects the web accessibility violation without introducing new accessibility violations. Enclose your corrected HTML code to replace the initial code with accessibility violations between these two marker strings: "###START###" as first line and "###END###" as last line.
Confidence Assessment	Fixed	Evaluate your confidence (0-100%) in your correction, enclose your confidence score between these two marker strings: "###START1###" as first line and "###END1###" as last line. Provide an explanation for this confidence level; enclose your explanation between these two marker strings: "###START2###" as the first line and "###END2###" as last line.

Table 11. **Corrective re-prompting template** used in the *AccessGuruCorrect* module. The prompt is structured into fixed and dynamic components. The fixed components remain constant across all samples, while dynamic fields are populated based on the violation instance. (For semantic violations involving visual content, an additional instruction guides the LLM to reason over webpage screenshots—see purple-highlighted rows.)

Metacognitive Prompting Stage	Type	Prompt Template

Comprehension Clarification	Fixed	`{Role-play persona} + You are analyzing a web accessibility issue using a snippet of Affected HTML Element(s) , Web page screenshot and related metadata. The screenshot reflects exactly what is rendered to users. Follow these strict rules:`
		• `Prioritize visual analysis: list at least three specific details observable in the image (e.g., color, shape, text, or spatial arrangement).`
		• `Analyze only the provided HTML snippet and metadata. Do not infer or invent additional structure, styles, or UI elements beyond what is given.`
		• `Avoid introducing or rewriting content not present in the HTML. Do not add or alter CSS, forms, headers, sections, scripts, or attributes unnecessarily. Modify only the minimal code needed to resolve the violation.`
		• `Return only the modified lines in a fenced code block. Leave all other parts of the HTML unchanged.`
		• `Justify every accessibility concern directly with observable evidence from the HTML.`
		`Clarify your understanding of the following web accessibility violation:`

	Dynamic	`{Category}`
		`{Category description}`
		`{Violation name}`
		`{Violation description}`
		`{URL}`
		`{HTMLElement}`
		`{Impact}`
		`{Web page screenshot}`

	Fixed	Impact is a rating determined by the severity of the violation, indicating the extent to which it hinders user interaction with the Web content. The scale is [cosmetic, minor, moderate, serious, critical]
Preliminary Judgment	Fixed	Based on your understanding, provide a preliminary correction for the web accessibility violation based on the following WCAG guideline(s):
	Dynamic	`{Relevant WCAG guideline}`
	Fixed	Make sure your generated code corrects the web accessibility violation without introducing new accessibility violations. Ensure you generate the complete corrected code, not just a snippet.
Critical Evaluation	Fixed	Critically assess your preliminary correction, make sure to correct the initial web accessibility violation without introducing new web accessibility violations. Only make corrections if the previous answer is incorrect. Make sure your generated code corrects the web accessibility violation without introducing new accessibility violations.
Decision Confirmation	Fixed	Confirm your final decision on whether the correction is accurate or not, and provide the reasoning for your decision. Only suggest further corrections if the initial response contains errors. Make sure your generated code corrects the web accessibility violation without introducing new accessibility violations. Enclose your corrected HTML code to replace the initial code with accessibility violations between these two marker strings: "###START###" as the first line and "###END###" as the last line.
Confidence Assessment	Fixed	Evaluate your confidence (0-100%) in your correction, enclose your confidence score between these two marker strings: "###START1###" as first line and "###END1###" as last line. Provide an explanation for this confidence level; enclose your explanation between these two marker strings: "###START2###" as the first line and "###END2###" as last line.

Table 12. Prompt templates used for **baseline methods**, structured into fixed and dynamic components. Fixed components remain constant across all samples, while dynamic fields are populated based on the specific violation instance. These templates are directly adapted from the original works. (For semantic violations involving visual content, we added an additional instruction that guides the LLM to reason over webpage screenshots—see purple-highlighted rows.)

Baseline Method	Type	Prompt Template
Contextual Prompting (Othman et al.[37])	Fixed	Given the following Web page screenshot and source code, can you fix the accessibility issue related to the success criteria according to WCAG 2.1?
	Dynamic	`{HTML}`, `{WCAG relevant to the violation}` , `{Web page screenshot}`
ReAct Prompting (Huang et al. [23])	Fixed	You are a helpful assistant who will correct accessibility issues of a provided website. Provide your thought before you provide a fixed version of the results. E.g. **Incorrect:** Search **Thought:** because ... I will ... **Correct:** Search You are operating on this website:
	Dynamic	`{Web page URL}`, `{violation name}`, `{violation description}`, `{fix advice}`, `{HTML}`
	Fixed	Given the Web page screenshot:
	Dynamic	`{Web page screenshot}`
Zero-shot Prompting (Delnevo et al. [15])	Fixed	Is the following HTML code accessible?
	Dynamic	`{HTML}`
	Fixed	Given the Web page screenshot:
	Dynamic	`{Web page screenshot}`

Table 13. Subset of our Proposed Taxonomy to Categorize Web Accessibility Violations. The full taxonomy can be found in our supplementary material.

Category	Violation Name	Description	Violated Guidelines	Impact	Supplementary Information
Semantic Accessibility Violations	image-alt-not-descriptive	Inaccurate or misleading alternative text that fails to describe the purpose of the image.	WCAG 1.1.1	Critical	Image
	video-captions-not-descriptive	Inaccurate video captions.	WCAG 1.2.1, 1.2.3	Critical	Video
	lang-mismatch	Page language attribute does not match the actual language of the content.	WCAG 3.1.1	Serious	–
	link-text-mismatch	Links fail to convey their purpose or are ambiguous.	WCAG 2.4.4, 2.4.9	Serious	–
	button-label-mismatch	Button labels are unclear or fail to specify their purpose.	WCAG 4.1.2, 2.5.3	Critical	–
	ambiguous-heading	Headings are vague, repetitive, or fail to describe the content.	WCAG 2.4.6, 2.4.10	Moderate	–
	incorrect-semantic-tag	A non-semantic tag (e.g., div or span) is used instead of a proper semantic element (e.g., header, nav, main).	WCAG 1.3.1	Serious	Document structure (other headings, section context)
	color-only-distinction	Visual information is conveyed using color alone without additional indicators like text, shape, or pattern, making it inaccessible to users with color vision deficiencies.	WCAG 1.4.1	Serious	Web page Screenshot
Layout Accessibility Violations	meta-viewport	Ensure <meta name="viewport"> does not disable text scaling and zooming	WCAG 1.4.4	Critical	–
	color-contrast	Ensure the contrast between foreground and background colors meets WCAG 2 AA minimum contrast ratio thresholds	WCAG 1.4.3	Serious	Color Information (Background and Foreground)
	avoid-inline-spacing	Ensure that text spacing set through style attributes can be adjusted with custom stylesheets	WCAG 1.4.12	Serious	–
	target-size	Ensure touch targets have sufficient size and space	WCAG 2.5.5	Serious	–
Syntax Accessibility Violations	duplicate-id-aria	Ensure every id attribute value used in ARIA and in labels is unique	WCAG 4.1.2	Critical	–
	tabindex	Ensure tabindex attribute values are not greater than 0	WCAG 2.1.1	Serious	–
	duplicate-id-aria	Ensure every id attribute value used in ARIA and in labels is unique	WCAG 4.1.2	Critical	–
	tabindex	Ensure tabindex attribute values are not greater than 0	WCAG 2.1.1	Serious	–
	valid-lang	Ensure lang attributes have valid values	3.1.2	Serious	–
	aria-required-attr	Ensure elements with ARIA roles have all required ARIA attributes	WCAG 4.1.2	Critical	–
	meta-refresh	Ensure <meta http-equiv="refresh"> is not used for delayed refresh	WCAG 2.2.1	Critical	–
	empty-table-header	Ensure table headers have discernible text	WCAG 1.3.1, 2.4.6	Minor	–
	empty-heading	Ensure headings have discernible text	WCAG 1.3.1, 2.4.6	Minor	–