ReFLAIR: Detecting Responsive Layout Reflow Issues using Multimodal Generative AI

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

With over 60 percent of global Internet traffic originating from mobile devices, Responsive Web Design (RWD) has become essential for ensuring seamless user experiences across diverse screen sizes and resolutions. The Web Content Accessibility Guidelines require that both information and functionality remain accessible when reflow occurs. However, existing checkers or research tools have limitations: they either employ static analysis that fails to capture how content is actually displayed to users or focus on only one accessibility aspect, such as keyboard operation. Consequently, no prior study has comprehensively addressed both information and functionality loss during reflow for Graphical User Interfaces (GUIs).

This paper introduces ReFLAIR (**Re**flow **F**ault **L**ocalization using **AI**-based **R**esponsive analysis), a multimodal generative AI–driven approach for dynamically detecting reflow issues that cause loss of information or functionality in the GUI. ReFLAIR systematically extracts informative and actionable widgets, compares their presence and behavior across original and reflowed layouts, and employs both scrolling and large language model–guided expansion to uncover hidden interface widgets. We evaluate ReFLAIR on a dataset of 24 diverse webpages drawn from popular sites, prior benchmarks, and newly released webpages. Results show that ReFLAIR outperforms five state-of-the-art techniques, achieving precision improvements of at least 20.49% and recall improvements of at least 55.40%, while maintaining reasonable computational and runtime cost. An ablation study confirms that dynamic exploration (i.e., scrolling and expansion) is essential for high accuracy. We evaluated scalability and generalizability by extending the dataset to 36 webpages, covering 28 domains and higher complexity, and experimenting with alternative models and viewports. The results reinforce ReFLAIR's consistency across diverse subjects and configurations. In summary, our approach contributes to accessibility testing by providing an effective, scalable, and cost-efficient solution for identifying reflow issues in RWD.

CCS Concepts: • **Software and its engineering → Software testing and debugging**.

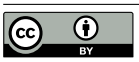
Additional Key Words and Phrases: Responsive Web Design, Web Testing, Multimodal AI

ACM Reference Format:
Yirui He, Ziyao He, Syed Fatiul Huq, and Sam Malek. 2026. ReFLAIR: Detecting Responsive Layout Reflow Issues using Multimodal Generative AI. *Proc. ACM Softw. Eng.* 3, FSE, Article FSE129 (July 2026), 23 pages. https://doi.org/10.1145/3808136

1 Introduction

Over the past decades, the rapid development and innovation of web technologies have made webpages essential components of daily life for billions of people [6]. Modern webpages have evolved to become as powerful and convenient as traditional desktop applications, while eliminating the need for complex installation procedures [46].

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



ACM 2994-970X/2026/7-ARTFSE129
https://doi.org/10.1145/3808136

Responsive Web Design [15] (RWD) has become important for user experience and web accessibility as the modern digital landscape encompasses a vast ecosystem of web-enabled devices with diverse screen sizes and resolutions. With over 60 percent of global Internet traffic originating from mobile devices, webpages need to seamlessly adapt across different viewports for smartphones, tablets, and desktops to ensure optimal user experience [48]. A previous study involving 181 industry professionals revealed that developers primarily adopt RWD to improve user experience and increase accessibility [32]. In addition, RWD offers benefits that extend beyond accessibility. Since Google's mobile-first indexing prioritizes mobile-friendly pages [11], RWD also helps businesses maintain traffic and conversions [58]. Moreover, maintaining a single adaptable codebase offers significant cost-efficiency by eliminating the need for separate mobile and desktop sites, thereby reducing development, maintenance, and testing overhead [33]. As web technologies continue evolving with new screen formats and devices, RWD ensures future-proof, accessible, and high-performing digital experiences.

RWD has attracted significant attention from both industry and academia. The Web Content Accessibility Guidelines (WCAG)[69] enhance web accessibility by providing standards for creating inclusive digital environments. Specifically, WCAG requires that when reflow occurs, both *information* and *functionality* must remain accessible [20]. However, most tools that check WCAG compliance are static, ignoring the dynamic nature of webpages and how content is actually displayed to users. While prior research has studied responsive web design from various perspectives [43, 49, 70, 71], none have comprehensively detected both *information* and *functionality* loss from the Graphical User Interface (GUI) during reflow from a dynamic analysis perspective.

In this work, we present ReFLAIR (**Re**flow **F**ault **L**ocalization using **AI**-based **R**esponsive analysis), an approach for automatically detecting reflow issues in GUIs with dynamic click interactions. Reflow issues occur when the webpages' layouts change, and informative or functional widgets disappear on the reflow page. To detect the reflow issue, any information or functionality that appears on the original page but is absent from the reflowed page is counted as missing, indicating a reflow issue. We implement ReFLAIR and evaluate it on a dataset containing three sources: popular webpages, webpages from prior studies, and newly released webpages that postdate the LLM used in our implementation. Our evaluation demonstrates that ReFLAIR achieves 80.49% precision and 95.31% recall, outperforming the second-best baselines by 20.49% in precision and 55.40% in recall while maintaining reasonable cost.

To summarize, this paper makes the following contributions:

- We introduced ReFLAIR, a dynamic approach for detecting reflow issues in GUIs.
- We evaluated ReFLAIR's effectiveness by comparing it against seven baselines. The results demonstrate that ReFLAIR outperforms existing methods in both precision and recall metrics with reasonable cost.
- We further validated ReFLAIR 's generalizability and scalability on an extended dataset comprising 36 webpages across 28 distinct domains [45], and demonstrated ReFLAIR 's adaptability to an alternative LLM backend and an additional tablet viewport configuration.
- We made the implementation of ReFLAIR available publicly [14] .

The remainder of the paper is organized as follows: Section 2 explains the background and provides a motivating example; Section 3 describes ReFLAIR in more detail; Section 4 details our evaluation and analysis of the obtained results; Section 5 includes discussion and describes threats to validity; Section 6 outlines related work and Section 7 concludes the paper.

2 Background and Motivation

2.1 Responsive Web Design and WCAG

Modern websites and applications commonly employ RWD to adapt webpage layouts to different viewport sizes across various end-user devices [10]. To maintain equivalent user experiences across devices, webpages may need to adjust or relocate content sections accordingly. Such adaptations need to preserve both *information* and *functionality*, ensuring users retain access to all content. RWD benefits not only mobile device users but also individuals who require interface resizing or zooming on larger devices, which is a key objective of WCAG Success Criterion (SC) 1.4.10 [20].

WCAG is an internationally recognized standard developed by the World Wide Web Consortium (W3C) to make web content more accessible [57]. WCAG was first published in May 1999, aiming to enhance accessibility by providing guidelines that help create inclusive digital environments. The latest version, WCAG 2.2, was published in October 2023 [69]. In the guideline, SC 1.4.10 specifies that webpage content should be presented without loss of information or functionality, and without requiring two-dimensional scrolling except in specific cases. For English webpages, which typically require vertical scrolling for reading, content should be accessible through vertical scrolling alone. However, horizontal scrolling is acceptable for content that inherently requires two-dimensional layout, including essential images (e.g., maps and diagrams), video, games, presentations, data tables, and interfaces where toolbars must remain visible during content manipulation.

2.2 Motivating Example

(a) Full-Sized Page of Mailinator (1280 CSS Pixel)

(b) Reflowed Page of Mailinator (320 CSS Pixel)

Fig. 1. A Reflow Issue in Mailinator [52]

RWD is a widely adopted approach to ensure that webpages remain functional and visually coherent across a broad range of screen sizes. However, the requirements of adaptability across diverse devices can introduce content accessibility issues that are difficult to detect through traditional static analysis.

Consider the example illustrated in Fig. 1, which shows reflow issues in "Mailinator " [52], a publicly available real-world webpage. Mailinator provides users with temporary inboxes for receiving emails without registration. The original webpage *(a)*, rendered at a full-sized screen (1280 CSS pixels), displays seven navigation options in the header (*HOME, PRICING, FAQ, API, EMAIL, LOGIN, SIGNUP*). When reflowed to a narrow viewport (320 CSS pixels), as shown in *(b)*, the layout is adapted to accommodate all existing elements within the constrained space. This example demonstrates common reflow adaptation. The navigation options are collapsed into a hamburger menu, accessible via the menu button in the top-right corner. When clicked, this reveals a vertical navigation menu containing the primary options (*HOME, PRICING, EMAIL, LOGIN, SIGNUP*). However, reflow issues occur when there is a loss of information or functionality after the transformation. In our example, while most navigation options successfully transition to the hamburger menu, the *FAQ* and *API* options are missing from the collapsed menu, preventing users from accessing these important sections. This demonstrates how common UI adaptations—such as collapsing navigation into hamburger menus—can inadvertently hide critical functionality.

As our motivating example illustrates, determining whether certain UI elements remain discoverable after reflow requires user interactions. Static analysis tools [13, 23, 49, 71] struggle with

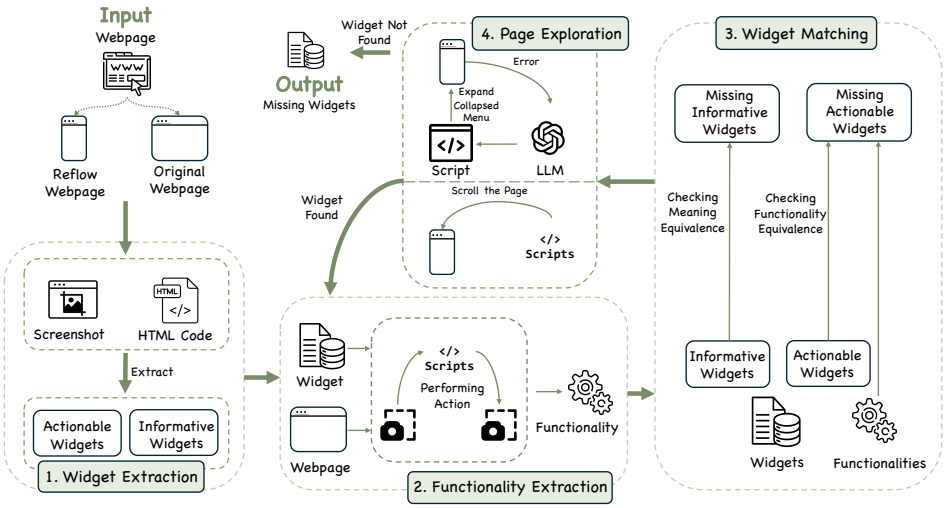


Fig. 2. An Overview of ReFLAIR

detecting this issue because they cannot simulate the necessary interactions to reveal the hidden content. For instance, the collapsed navigation menu appears only when users click the hamburger button, and without simulating this interaction, static tools cannot determine whether all original navigation options remain accessible. When *information* and *functionalities* become inaccessible in the reflowed webpage, we consider them reflow issues that compromise the user experience and violate accessibility guidelines. We propose a dynamic analysis approach that simulates user interactions to systematically identify reflow accessibility issues.

3 Approach

We present an automated approach, named ReFLAIR, for detecting reflow issues in webpages, focusing on *information* and *functionality* loss.

Fig. 2 illustrates the overall workflow of ReFLAIR for identifying reflow issues. The approach identifies widgets in the original webpage within the reflowed version. Widgets that cannot be located in the reflowed page are identified as missing widgets, indicating potential reflow issues.

In **stage 1**, the process begins with an input webpage that undergoes widget extraction to identify actionable and informative widgets from both screenshot and HTML code analysis. In **stage 2**, for each actionable widget, the system proceeds with functionality extraction to determine widget functionalities. In **stage 3**, the widget matching process then determines which widgets from the original page are present in the reflowed page and which are missing. Lastly, in **stage 4**, when widgets are not initially found, the system explores the page by scrolling and using LLM-guided expansion of collapsed menus to uncover hidden widgets, ensuring that any newly discovered widget matches the one from the original page.

3.1 Informative Widget and Actionable Widget

Following WCAG guidelines, ReFLAIR examines information accessibility and functional accessibility separately, checking whether any loss occurs during reflow. To assess ***information*** accessibility, ReFLAIR focuses on two types of ***informative widgets***: *visible text* and *icons/images accompanied by non-empty alt attributes*. These widgets are critical to web accessibility, as they represent the primary means by which users consume and interpret content—especially those relying on assistive

technologies. Text is the most fundamental widget for conveying meaning, and ensuring its readability and correct positioning under reflow is essential for both sighted users and visual impairments. Similarly, images and icons often encode functional or semantic information, and the HTML `alt` attribute serves as a textual alternative that allows assistive technologies to convey information to users with visual impairments and enables sighted users to obtain content when images are not visible [19, 24]. Using the criterion of filtering out images without `alt` attributes, ReFLAIR also excludes decorative images and icons [19, 66]. By focusing on these two types of widgets, our analysis captures the information units that are both perceptually and functionally critical under reflow, aligning with accessibility standards and best practices. To assess ***functional*** accessibility, ReFLAIR focuses on ***actionable widgets***. More specifically, ReFLAIR extracts interactive widgets, such as buttons, links, and any widgets with JavaScript event listeners attached [8].

The *informative widget* set and *actionable widget* set for a webpage are not disjoint sets. Rather, they may contain overlapping widgets. A text-based hyperlink exemplifies this overlap, as it simultaneously functions as both an informative widget (i.e., providing textual content) and an actionable widget (i.e., enabling navigation). In contrast, some widgets belong to only one category: a non-interactable image with `alt` text serves solely as an informative widget, while an interactive icon without `alt` text functions only as an actionable widget.

3.2 ReFLAIR

The approach consists of four stages: *Widget Extraction*, *Functionality Extraction*, *Widget Matching*, and *Page Exploration*. ReFLAIR systematically processes both original and reflowed versions of webpages during the *Widget Extraction* and *Functionality Extraction* stages. Subsequently, ReFLAIR examines whether widgets from the original page directly appear in the reflowed page during *Widget Matching*. Widgets that cannot be located proceed to *Page Exploration* for further investigation. If widgets remain unfound after *Page Exploration*, they are classified as missing.

1. Widget Extraction. The widget extraction stage performs static analysis of the target webpages' Document Object Model (DOM) and screenshots.

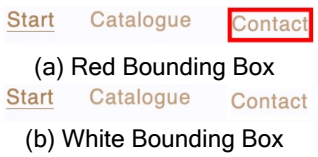


Fig. 3. Bounding Box to Ensure "*Contact*" Widget is Displayed

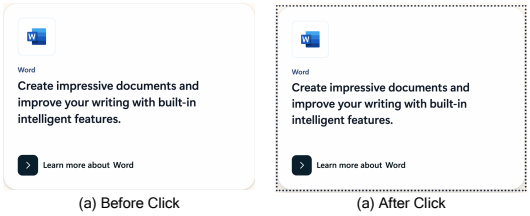
ReFLAIR identifies all visible widgets containing meaningful textual content by traversing the rendered DOM and selecting nodes with non-empty normalized text values. The normalization process removes leading and trailing whitespace and collapses internal whitespace sequences, excluding purely decorative or empty widgets. To extract informative image-based widgets, ReFLAIR focuses on DOM queries that cover multiple rendering mechanisms commonly used in modern web development [67, 68, 74, 75]. These queries target standard `<img>` tags, widgets annotated with `role="img"` (i.e., an ARIA attribute signaling image interpretation by assistive technologies), and `<svg>` widgets representing scalable vector graphics.

To identify actionable widgets from a given interface, ReFLAIR followed established best practices [43, 49] by extracting all interactive UI widgets, including (1) native control widgets (e.g., `button`) and (2) non-native control widgets that have been customized with interactivity (e.g., customized buttons implemented using `<div>` widgets) and are attached to mouse-related event listeners (e.g., `onclick`).

ReFLAIR filters out invisible widgets to ensure the extracted widgets are displayed on the GUI and visible for end-users. It verifies widget visibility through three validation mechanisms. ReFLAIR first uses the Selenium function `is_displayed()`. It then checks the location of the widget to ensure the widget is located within the viewport. Finally, to provide visual confirmation, ReFLAIR captures two screenshots of the same page: one with a red bounding box placed around the target

widget, and another with a white bounding box, as illustrated in Fig. 3. By comparing these two screenshots, ReFLAIR can determine widget visibility: if the target widget is present on the page, the two screenshots will differ from each other due to the color difference of the two bounding boxes. However, if the widget is missing, both screenshots will appear identical since there is no displayed widget to flag with a bounding box. This multi-stage filtering process was implemented because confirmation using bounding boxes is time-consuming. ReFLAIR first uses `is_displayed()` and viewport checks as a preliminary filter to exclude obviously hidden widgets, then applies bounding box verification as a more precise method to confirm widget visibility. This approach reduces the overall computational overhead while maintaining accuracy.

2. Functionality Extraction. For each actionable widget, ReFLAIR extracts its functionality by capturing the page changes introduced through interacting with widgets. Specifically, we define *functionality extraction* as capturing before-and-after screenshots along with any changes to the Uniform Resource Locator (URL). To achieve this, ReFLAIR identifies widgets using their corresponding `XPath` [21] selectors for precise widget targeting, then performs click actions on the widgets and records both URL and GUI changes (through paired screenshots) to capture user-perceivable changes. Additionally, we exclude actionable widgets that exhibit only visual highlighting effects without revealing additional information or providing navigation to other sections or pages (e.g., see Fig. 4), because these widgets do not contribute substantively to the functional completeness assessment. In contrast to prior approaches [43] that characterize functionality through structural tuples extracted from HTML source code (i.e., function, tag type, input attributes, labels, and textual properties), ReFLAIR directly interacts with the GUI with Selenium's `click()` function and captures screenshots both before and after each interaction, thereby preserving visual context for subsequent steps.



(a) Before Click (a) After Click

Fig. 4. Highlight Effect After Click Text "Word"

3. Widget Matching The widget matching stage determines whether widgets from the original webpage are present in its reflowed version. This step requires all extracted *informative widgets* and *actionable widgets* along with their corresponding functionalities.

The matching process follows a categorical approach based on widget types. For informative widgets that lack interactive functionality, ReFLAIR verifies whether the information appears on the reflowed page through information equivalence analysis. The information equivalence verification proceeds as follows: ReFLAIR first determines whether the text extracted from the original version has an exact match within the text set extracted from the reflowed version. If an exact match exists, the system verifies whether the context is the same. When multiple instances of identical text appear across either page version, as illustrated in Fig. 5 with repeated occurrences of "9.6" highlighted with red boxes and "Exceptional" highlighted with green boxes, ReFLAIR provides screenshots as context with the highlighted text to the LLM to determine whether the texts convey equivalent semantic meaning. For instance, those "9.6" from the screenshots convey different semantic information as these values

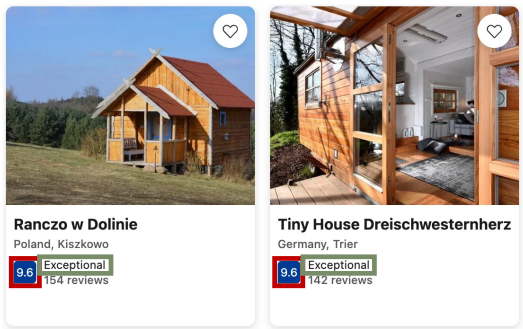


Fig. 5. Duplicated Text with Different Context

represent scores for different houses. Similarly, for images or icons with `alt` text, ReFLAIR applies the same equivalence verification process. If no equivalent widget is identified on the reflowed page following this analysis, the widget is added to the *Missing Informative Widgets* set for further exploration, as shown in Figure 2.

For widgets that fall into the category of actionable widgets but are not informative, ReFLAIR examines whether the action can be performed on the reflowed page. If a similar action cannot be achieved on the reflowed page, the widget is added to the *Missing Actionable Widgets* set for further analysis. ReFLAIR evaluates functionality equivalence through two different approaches based on the nature of widget interactions. For widgets that trigger navigation to other pages, the system employs URL-based comparison. When interaction with a widget from the original page results in navigation to another webpage, ReFLAIR compares the destination URLs after removing query parameters and retaining only the base URL. If the processed URLs are identical, the actions are considered functionally equivalent. For widgets that do not result in page transitions, ReFLAIR utilizes an LLM-based comparison approach. The system provides the LLM with image tuples captured before and after interaction, representing functionalities extracted from both the original and reflowed pages. The LLM is queried to assess whether interactions with corresponding widgets across both page versions produce comparable functional outcomes.

When widgets are both informative and actionable, ReFLAIR examines the reflowed page to determine whether equivalent widgets preserve the original information content and functionality. If the information is not conveyed on the reflow page, the widget is classified as a *Missing Informative Widget* and marked for further exploration. If the equivalent functionality cannot be achieved, the widget is classified as a *Missing Actionable Widget* and similarly marked for investigation.

4. Page Exploration When widgets from the original page are not directly visible on the initial page view, we employ two distinct exploration strategies to reveal hidden widgets: *scrolling* and *expansion*. This step takes *Missing Informative Widgets* and *Missing Actionable Widgets* as input and aims to determine whether these widgets are truly missing or simply hidden from the initial view.

The scrolling functionality takes missing widgets as input (i.e., widgets that appear in the original page but do not appear in the reflow page). For each informative widget with an informative identifier (i.e., text for textual widgets or `alt_text` attributes for image-based widgets), ReFLAIR locates all corresponding widgets on the reflowed page that share the identifier and are accessible through scrolling. ReFLAIR implements this scrolling mechanism using Selenium APIs to locate the target widget and JavaScript's `scrollIntoView()` function to bring the widget into the current viewport. For actionable widgets, ReFLAIR extracts the functionality of these widgets as detailed in the *Functionality Extraction* stage and performs comparative analysis against the original widget extracts described in the *Widget Matching* stage. If the newly explored widget matches the original widget, then the widget is removed from the missing widget set.

The expansion approach operates through a different strategy while maintaining similar objectives. Initially, we extract informative identifiers from each informative widget. These target identifiers are subsequently integrated into a predefined prompt template, accompanied by preprocessed HTML code that is passed to the LLM. The HTML preprocessing involves removing extraneous content, including comments and irrelevant tags (e.g., `<script>` and `<style>` widgets) to optimize the input for analysis. At this stage, the LLM is instructed to generate executable Python code using Selenium APIs or JavaScript code to interact with the browser and reveal target widgets concealed within collapsed menu structures, such as hamburger menus or expandable sections. When code execution fails, we provide both the generated code and the corresponding error information to the LLM as feedback, thereby preventing repetition of unsuccessful actions in subsequent attempts. When a hidden widget is revealed and detected by the LLM, we record the action sequence to reveal the widget and count the widget as revealed. Upon successful revelation

of widgets containing displayed identifiers, ReFLAIR extracts their functionality through the *Functionality Extraction* stage and performs comparative analysis with the original widget extracts (via *Widget Matching*). This process terminates when one of the following criteria is met: (1) no missing widgets remain, (2) the LLM determines that the remaining missing widgets are unreachable, or (3) the query count exceeds the maximum configurable limit (e.g., 3 in our experiments).

For actionable widgets lacking displayed identifiers, we evaluate whether equivalent functionality can be achieved through scrolling and expansion mechanisms during the exploration stage.

If a widget remains unfound after this stage, it is considered missing and added to the output set.

3.3 Prompt Design

In developing all prompts for ReFLAIR, we adhered to established best practices from both general LLM applications [79] and web accessibility research [49]. Our methodology encompasses the five essential components of effective LLM prompting: *instruction*, *contextual information*, *input*, *demonstration*, and *output*. We further enhanced our approach through *expert prompting* techniques [49, 79], which involve assigning specialized domain expertise roles to the model to extract more nuanced and technically accurate responses.

For instance, when evaluating functionality equivalence, our prompt is as follows:

> You are an expert UI interaction analyst. [**Expert Prompting**]
> I will give you the following **inputs**: [**Input**]
> The action I performed is a click. [**Contextual Information**]
> First image: Original screenshot of a webpage before an action is performed.
> Second image: Original screenshot of the webpage after the action was performed.
> Third image: Screenshot of the reflowed version of the same webpage before the action is performed.
> Fourth image: Screenshot of the reflowed webpage after the action is performed.
> Your **task** is to: [**Instruction**]
> - Analyze the visual and textual differences between the "before" and "after" states in both the original and reflow versions to determine whether the widget's behavior is consistent across layouts.
> - Please keep the summary:
> 1. Strictly follow the information provided in the images.
> 2. Clear and concise (1–2 sentences).
> 3. Focused on the user-facing functionality.
> 4. Avoid speculating about implementation.
>
> **Output** format: [**Output**]
> Functionally equivalent: Yes/No/Not Sure
> Reason: Brief explanation
> **Example output 1**: [**Demonstration**]
> Functionally equivalent: Yes
> Reason: There are no visible changes in either the original or reflow versions after the click, indicating identical behavior.
> **Example output 2**:
> Functionally equivalent: No
> Reason: Although the UI changed after the click, the displayed information is totally different in the reflow version compared to the original version.

4 Evaluation

To assess ReFLAIR, we study the following research questions:

- **RQ1**: What is ReFLAIR's **effectiveness** in terms of **precision** and **recall**.
- **RQ2**: What is ReFLAIR's **cost** in terms of **API token usage** and **runtime**?

- **RQ3**: To what extent does each component contribute to the overall performance?
- **RQ4**: To what extent does ReFLAIR **generalize**?
  - **RQ4.1**: To what extent does ReFLAIR generalize and scale with increasing dataset sizes across different domains?
  - **RQ4.2**: To what extent is ReFLAIR adaptable to variations in LLM backends and viewport configurations?

4.1 RQ1: Effectiveness

4.1.1 Implementation and Configuration. The evaluation was conducted on a laptop with a 2.6 GHz 6-Core Intel Core i7 processor, 32 GB of 2400 MHz DDR4 memory, running macOS Sonoma 14.7.3. ReFLAIR employs the Selenium WebDriver API to control a Firefox browser instance for rendering webpages across varying viewport configurations. To generate the full-sized page, the browser viewport is configured to dimensions of 1280 × 1024 CSS pixels. Meanwhile, the reflowed version is produced by configuring the viewport to 320 × 1024 CSS pixels, following prior work [43] and established accessibility testing guidelines [20].

We selected GPT-4.1 (*gpt-4.1-2025-04-14*) as the LLM for ReFLAIR's implementation based on several key factors. First, GPT-4.1 represents one of the most advanced multimodal models available, demonstrating exceptional performance in complex tasks such as image understanding and code generation [9, 12], both of which are essential capabilities for our reflow issue detection task. Second, it allows changing the temperature while newer model (i.e., GPT-5) does not. Finally, OpenAI's API has demonstrated superior performance in text evaluation and vision understanding tasks specifically related to accessibility assessment [49], making it particularly well-suited for our domain-specific requirements.

4.1.2 Selecting Subject Webpages. Our dataset comprised of webpages from three distinct sources: popular webpages, webpages from prior datasets, and newly registered webpages.

We selected popular webpages from Semrush [45], following prior study [65], which offers a list of the 20 most popular webpages, based on traffic, across 33 categories (e.g., science, education, etc.). Next, we chose webpages from the artifacts of prior work, such as SALAD [43], ReDeCheck [71] and KAFE [42]. We verified whether the specific version used in the prior study is still available, using the Wayback Machine [38]. Finally, to avoid information leakage issues, we added webpages published after the release date of our model, i.e., *gpt-4.1-2025-04-14* [12]. These webpages were retrieved from the newly registered domain name community list of June 1st, 2025 [63]. We verified their publication dates using Wayback Machine [38], confirming that no archives existed before the model's release.

From these three sources, we only selected webpages that met the following criteria: (1) majority of content was in English, (2) page was rendered properly (e.g., no corrupted page layout), and (3) webpages functioned properly with Firefox [4] and Selenium [17]. Applying these criteria, we finalized our dataset with 12 popular webpages, 7 webpages from prior work and 5 newly registered ones, totalling to 24 webpages.

For analysis, we downloaded copies of the webpages rather than relying on live versions, to address two challenges. First, our analysis involved issuing repeated automated requests (e.g., loading pages and interacting with widgets), which could be misinterpreted as suspicious traffic and trigger temporary access denial due to security defenses against Distributed Denial of Service (DDoS) attacks [18]. Second, live webpages frequently change their content and layout, which would compromise the stability and reproducibility of our evaluation. By working with downloaded copies, we mitigated both issues and ensured reliable conditions for our study.

To account for responsive webpage behavior, we downloaded each webpage under two viewport configurations: the original layout (1280 × 1024 CSS pixels) and a reflowed layout (320 × 1024

CSS pixels). Using these viewport sizes ensures that reflow is handled by the webpage's built-in responsive scripts. Prior to downloading, we scrolled to the bottom of each webpage and waited for all content to render, thereby capturing lazy loading elements.

4.1.3 Building Ground Truth. With the downloaded webpages, we manually constructed the ground truth. The ground truth consisted of annotated screenshots of webpages, marking widgets with reflow issues. The process involved systematic manual inspection and interaction with both full-sized and reflowed versions of each webpage, finding widgets that were absent from or did not function the same in the latter version. The first author conducted the initial inspection, with two others independently verifying the resulting annotations. We used Roboflow [16] to record and compare our labeling across authors.

To build the ground truth systematically, we referenced WCAG *SC 1.4.10 Reflow* [20] and its associated *Failure F102* [5], which addresses content disappearing and not being available when content has reflowed. We adopt WCAG because the guideline establishes the de facto standard for evaluating reflow accessibility, ensuring our dataset aligns with accessibility requirements and remains comparable to prior studies.

For manual inspection, the first author verified whether information corresponding to each *informative widget* remained present and accessible in the reflowed version. When functionality was associated with an *informative widget*, the first author further assessed whether the functionality remained equivalent in terms of user interaction and output. For *actionable widgets*, the first author evaluated whether the associated actions remained accessible from the reflowed page without requiring horizontal scrolling at the page level.

To ensure the reliability of the initial annotation, two co-authors independently verified the results produced by the first author using the same inspection procedure and identified potential labeling discrepancies.

4.1.4 Baseline Techniques. We selected five state-of-the-art tools for our comparative analysis: GenA11y [49], SALAD [43], ReDeCheck [70, 71], WAVE [23] and QualWeb [13]. The selection was based on identifying leading research and industry tools for RWD and accessibility testing. Since none of the five state-of-the-art tools directly detect general reflow issues in terms of information and functionality loss, we adapted their outputs to best match our scenario in identifying reflow issues. In addition, we implemented two baselines using OpenAI GPT-4.1 [54], prompting the model to identify any reflow issues. The baselines are summarized in Table 1.

4.1.5 Result Verification Process. We verified the results of each technique according to the nature of its output format. For image-based outputs (GenA11y [49]), we compared annotated screenshots against the ground truth. For structurally rich outputs (ReFLAIR, SALAD [43] and ReDeCheck [70, 71]), we leveraged available XPath information when possible and otherwise relied on textual and contextual cues to map reported issues to ground-truth elements. For text-based outputs (GPT-4.1-Base and GPT-4.1-PE), we manually interpreted the descriptions and aligned them with corresponding ground-truth reflow issues. For QualWeb and WAVE, we assessed the number of widgets that violated reflow-related success criteria, specifically SC 1.4.10.

The first author inspected the output against the corresponding ground-truth screenshot to determine whether the reported counts were correct. Another author reviewed a stratified subset of results, based on page complexity, to verify the annotation. After achieving full agreement with the primary assessment, the remaining cases were subsequently verified by the first author.

4.1.6 Result and Analysis. We ran ReFLAIR on the evaluation dataset to assess its accuracy in detecting reflow issues. The tool reports XPath selectors for affected widgets on the original page.

In our evaluation, precision measures the proportion of reported issues that are true reflow problems, reflecting how often ReFLAIR produces correct detections. Recall measures the proportion of ground-truth reflow problems that ReFLAIR successfully identifies, reflecting how completely

Table 1. Overview of Baseline Tools and Methods for Reflow Accessibility Evaluation

Tool/Method	Description
GenA11y [49]	The state-of-the-art automated static web accessibility evaluation framework that leverages LLMs to detect WCAG violations requiring both syntactic and semantic understanding. The framework systematically extracts relevant elements from webpages and employs specialized LLM-based analyzers, each targeting specific WCAG success criteria, to identify accessibility violations that traditional rule-based tools often miss. GenA11y addresses 37 WCAG success criteria through static analysis, including reflow-related accessibility issues from SC 1.4.10 (Reflow), which evaluates content adaptability across different zoom levels. In this study, we utilize GenA11y to assess reflow accessibility violations by analyzing how web content behaves when constrained to narrow viewport widths, particularly focusing on the 320-pixel breakpoint specified in WCAG reflow guidelines.
SALAD [43]	An automated tool that detects reflow accessibility failures in webpages for keyboard users. The tool explores webpages under responsive settings, exercising dynamic interactions that often expose layout and usability issues. SALAD provides automated detection of accessibility violations that emerge during reflow, thereby complementing existing static accessibility checkers.
ReDeCheck [70, 71]	An automated static responsive web design testing framework that validates webpage layouts across multiple viewport configurations. The framework systematically extracts static layout at diverse viewport widths spanning smartphone to desktop resolutions and analyzes layout behavior to identify five predefined categories of responsive design defects: *element collision*, *element protrusion*, *viewport protrusion*, *small-range layouts*, and *wrapping elements*. Since ReDeCheck's default configuration reports layout failures across viewport widths from 480 to 1280 pixels, we modified the tool to focus exclusively on the 320-pixel viewport width corresponding to the reflow breakpoint and evaluated its detected failures against our established ground truth.
WAVE [23]	An automated web accessibility evaluation framework that detects WCAG violations and accessibility issues statically. The tool prioritizes issues that impact end users, and provides educational feedback to developers. We utilize the WAVE browser extension for Firefox [22] to evaluate a subset of WCAG guideline checks specifically related to reflow accessibility errors [20].
QualWeb [13]	An automated static web accessibility evaluation framework that validates webpages against WCAG 2.1 Techniques [2] and Accessibility Conformance Testing Rules [1]. We utilize the QualWeb command-line interface [7] to detect violations of WCAG reflow guidelines [20].
GPT-4.1-Base	Baseline utilizing processed HTML and full screenshots (original vs. reflowed) with direct prompting. In the implementation, we selected GPT-4.1 [54] over GPT-5 [55] due to practical constraints associated with context windows limitation and non-adjustable temperature parameter in GPT-5, which made GPT-4.1 a more suitable choice for our experimental setup. The specific prompt is detailed as follow: *I will provide you with:* *First image: Screenshot of the original webpage (full layout).* *Second image: Screenshot of the reflowed webpage (responsive layout on a smaller screen)* *First HTML: The cleaned HTML source code of the original webpage.* *Second HTML: The cleaned HTML source code of the reflowed webpage.* *Your task is to determine whether any reflow issues are present.*
GPT-4.1-PE	Baseline incorporating prompt engineering strategies (structured prompt patterns described in Section 3.3) while following the same implementation as the Base version.

the approach covers existing issues. We chose precision and recall because reflow issue detection involves a trade-off between avoiding false positives (reporting layout changes that are benign) and avoiding false negatives (missing genuine reflow violations). Together, these metrics provide a clear and intuitive assessment of both the accuracy and completeness of ReFLAIR 's detection capability. To calculate precision and recall, we manually verified each reported issue by examining the webpages' GUIs and annotating the actual reflow problems present. We then compared these annotations against our ground truth dataset to determine the tool's performance.

To address the inherent variability of LLM outputs and ensure reproducibility, we set the temperature parameter to 0, which reduces randomness and yields more stable responses. Furthermore, each experiment of ReFLAIR is repeated three times, and we report the average performance across three runs to account for residual variability and provide robust evaluation results.

Table 2. Precision and Recall Scores of ReFLAIR and Baselines
(* indicates scores calculated on a subset of the dataset. The best score is bolded.)

Metric	ReFLAIR	GenAlly	GPT-4.1-PE	ReDeCheck	GPT-4.1-Base	QualWeb	WAVE	ReFLAIR *	SALAD*
Precision	**80.49%**	60.00%	57.82%	7.29%	6.67%	0.00%	0.00%	**74.90%**	27.85%
Recall	**95.31%**	8.45%	39.91%	4.23%	0.47%	0.00%	0.00%	**96.41%**	67.69%

The experimental results are shown in Table 2. ReFLAIR outperforms all state-of-the-art tools and GPT-4.1 baselines on both precision and recall. Across the 24 subjects in our dataset, ReFLAIR exceeds the second-best tool by 20.49% in precision and 55.40% in recall. This performance gain stems primarily from ReFLAIR 's dynamic analysis capabilities, which enable page exploration through interaction strategies such as scrolling and expanding operations. Moreover, we manually verify and calculate the precision and recall according to the semantic and interactive roles for informative widgets and actionable widgets separately. Informative widgets achieve 81.06% precision and 95.69% recall, and actionable widgets achieve 80.46% precision and 95.18% recall. These results closely align with the overall performance and further validate the robustness of our approach.

During our evaluation of two GPT-4.1-related baselines (i.e., GPT-4.1-Base and GPT-4.1-PE), we observed that GPT-4.1 possesses knowledge of reflow concepts. When prompted only with the keyword "reflow," GPT-4.1-Base demonstrated awareness of multiple relevant dimensions, including *overlapping elements*, *hidden content*, and *accessibility of interactive elements*. However, for most webpages, the model generated overly optimistic assessments, typically stating that "*The reflowed (mobile) version of the webpage is well-implemented, with no reflow issues. All content is preserved, accessible, and visually organized for smaller screens*," despite observable information or functional loss. Through prompt engineering strategy (i.e., GPT-4.1-PE), we found that the LLM's effectiveness in detecting reflow issues improved significantly. Nevertheless, both GPT-4.1 variants performed substantially worse than ReFLAIR across all evaluation metrics.

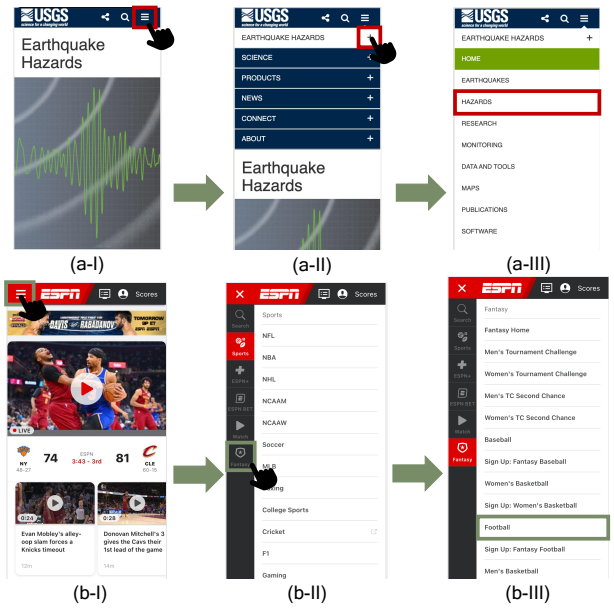


Fig. 6. Widgets Requiring Long Action Sequences to Reveal

GenA11y detects reflow issues by statically analyzing screenshots and HTML code with the support of LLMs. However, since GenA11y does not interact with webpages (e.g., scrolling to reveal dynamically loaded content), it often fails to capture the full page, resulting in limited recall. ReDeCheck attempts to detect visual or responsive issues through layout checking with heuristics. However, it has limited capability for discovering informative content and functionality loss when widgets disappear from the page, resulting in limited precision and recall. Traditional rule-based static checkers, such as WAVE and QualWeb, are designed to detect accessibility issues specified by WCAG. Although reflow is documented in WCAG, these tools have limited coverage of reflow-related success criteria [27] and cannot identify the issues in practice because static checkers operate on static HTML and lack the ability to examine responsive layouts. In contrast, ReFLAIR's dynamic exploration overcomes these limitations, allowing it to capture reflow issues that remain invisible to those static tools.

As SALAD is not publicly available, we relied on the results reported on the SALAD project website [3]. To enable a fair comparison, we evaluated our approach on the same dataset used by SALAD and selected a subset of that dataset for direct comparison against our ground truth. As shown in Table 2, ReFLAIR outperforms SALAD by 47.05% in precision and 28.72% in recall. While

SALAD successfully detects many reflow issues, it suffers from several key limitations. First, SALAD has restricted page exploration capabilities and cannot effectively handle situations where dynamic interaction is needed, such as widgets that become accessible only after expanding a hamburger menu (e.g., as shown in Fig. 6). Second, SALAD's analysis includes widgets not displayed on screen, which increases the number of reported issues and reduces precision. Regarding recall performance, since SALAD focuses on keyboard interaction instead of checking both informative and actionable widgets displayed on screens, it misses text elements that convey meaningful information to users but may not be keyboard-accessible, resulting in incomplete coverage of potential reflow issues.

Our experiments demonstrate ReFLAIR can successfully detect numerous reflow issues across various websites. For instance, consider the reflow issues affecting the Mailinator website, illustrated in Fig. 1. Similarly, ReFLAIR detected issues with the social media follow panel on the ESPN website, shown in Fig. 7. This panel, positioned on the left side of the original page, provides quick links to ESPN's official accounts on several platforms. However, during reflow, this panel disappears entirely and is not relocated to any collapsed menu, leaving users unable to access these social media links and creating both accessibility and discoverability issues.

ReFLAIR also successfully identified content presentation issues, as demonstrated in Fig. 8. On the original page, the section includes a prominent image with the alternative text "*Multiple images of the mouse brain on a black background with various cells highlighted to portray the precision access of new genetic tools developed.*" After reflow, this image disappears, leaving only the text portion of the section. This loss of visual content not only reduces the richness of the presentation but also removes important contextual elements for users, potentially hindering accessibility and overall comprehension.

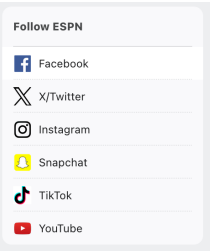


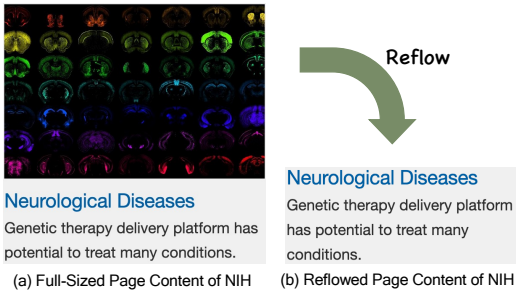
Fig. 7. ESPN [25] Social Media Links

While previous examples demonstrate ReFLAIR's effectiveness in detecting various types of reflow issues, our evaluation also revealed several limitations of the approach.

First, insufficient exploration can miss information that requires multi-step interactions, causing ReFLAIR to fail in revealing the corresponding widgets on the reflowed page. Fig. 6 illustrates that on a mobile interface multi-step interactions may be required before reaching the desired information. In both the USGS (a-I to a-III) and ESPN (b-I to b-III) examples, the initial screen provides only a high-level hamburger menu indicating details are hidden. To access specific content, such as "HAZARDS" on USGS [34] or "Football" on ESPN [25], users need to first open the navigation menu, then expand intermediate categories, and finally select the relevant sub-option. When exploration is not thorough enough, for instance, ReFLAIR fails to click the Fantasy icon in b-II. Consequently, actionable widgets including "Football", "Basketball", and "Fantasy Home" are missing because they are hidden behind layers of menus and require multiple actions to uncover.

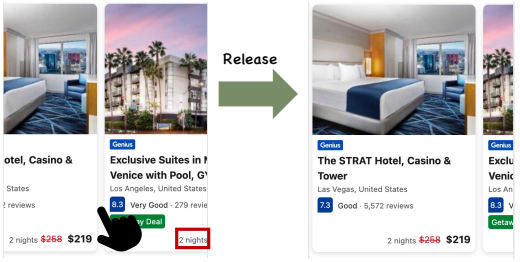
Second, false alarms may arise in scenarios where sliding windows employ magnetic snapping behavior. As shown in Fig. 9, temporary text visibility can occur during horizontal scrolling in a card-based interface. When ReFLAIR drags the carousel to an intermediate position, certain textual elements (e.g., '2 nights' with red box annotation) on a partially visible card become temporarily exposed. However, once the gesture is released and the carousel snaps to align the nearest card, these elements disappear and are no longer visible or interactable in the final layout state. Consequently, ReFLAIR treats such content as inaccessible and reports it as a reflow issue.

Finally, Fig. 10 illustrates a final scenario in which automatic content movement poses challenges for ReFLAIR. The upper part shows the original page layout displaying a banner containing the text "YOU'RE IN GOOD COMPANY" alongside several corporate logos, including Samsung, Coca-Cola, Sony, Johnson & Johnson, Citi, Spotify, and Toyota. In the reflowed version, showed on



(a) Full-Sized Page Content of NIH (b) Reflowed Page Content of NIH

Fig. 8. Image Missing after Reflow in NIH [26]



(a) Transient exposure of hidden text during horizontal scrolling in a card-based UI

(b) Text disappears once the scroll interaction ends and the card snaps into place.

Fig. 9. Automatic Position Change

the bottom, the banner automatically scrolls horizontally, sequentially revealing different subsets of logos over time: screenshot (a) shows Coca-Cola, Sony, and partial Johnson & Johnson logos; screenshot (b) displays Sony and Johnson & Johnson; and screenshot (c) shows Johnson & Johnson, Citi, and Spotify. Since this banner scrolling is automatic and not user-controllable, it creates a timing-dependent detection problem. Suppose an automated reflow detection system is specifically looking for the Sony logo but arrives at the page when it is in the displaying state (c) instead of states (a) or (b). In that case, the system fails to find the Sony logo and incorrectly reports it as a reflow issue, even though the logo is present but temporarily out of view.

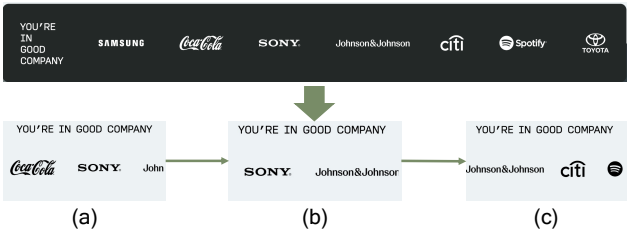


Fig. 10. Automatic Banner Scrolling

☑ **Finding 1:** ReFLAIR outperforms all state-of-the-art tools and standalone LLM approaches in both precision and recall. Specifically, across all 24 subjects, it improves precision by 20.49% and recall by 55.40% compared to the second-best technique. These results demonstrate that ReFLAIR can identify a comprehensive set of reflow issues while maintaining high accuracy.

4.2 RQ2: Cost

To assess the computational cost of ReFLAIR, we evaluate both token usage and runtime performance under the RQ1 setup. Each experiment is repeated three times to account for variability in LLM responses and execution times. For token usage analysis, we report the average, minimum, median, and maximum total tokens across 72 experiments (24 subjects × 3 runs), along with separate breakdowns for prompt and completion tokens. For runtime evaluation, we measure the execution time across 72 experiments of individual components on our experimental laptop, documenting cumulative sequential execution time and estimating potential parallel execution time to assess scalability.

Table 3. Statistical analysis of token usage per subject. The table presents average, minimum, median, and maximum values for total tokens, with corresponding breakdowns of prompt and completion tokens.

	# of Tokens		
	Total	Prompt	Completion
Average	561,590	552,514	9,076
Minimum	13,557	13,396	161
Median	252,294	246,925	5,369
Maximum	2,682,072	2,654,035	28,037

The statistics in Table 3 show that token usage per subject varies considerably. The median values indicate that typical runs involve around 252K total tokens, with the vast majority allocated

to prompts rather than completions, due to the query including images (i.e., screenshots) and HTML code. However, the maximum values highlight extreme cases, where the total token count reaches over 2.6M for a popular website (*booking.com*), driven primarily by very large prompts (i.e., around 0.2M tokens per prompt due to the complex HTML structure). In contrast, completions (i.e., LLM's responses) remain relatively small, with even the maximum completion size (28K) being minor compared to prompt usage. The averages are higher than the medians, reflecting the skew introduced by these outliers. Overall, the results indicate that prompt tokens account for the majority of the cost, whereas completion tokens contribute only marginally.

According to OpenAI's pricing documentation [56], our analysis reveals that each webpage incurs an average processing cost of $1.18, with minimum, median, and maximum costs of $0.03, $0.54, and $5.53, respectively. The substantial difference between the median ($0.54) and mean ($1.18) indicates that the average cost is skewed by computationally intensive webpages, such as booking platforms, which require approximately $4.99 on average across three experimental runs. Overall, these costs remain reasonable for automated accessibility testing, particularly considering that manual accessibility audits [30] typically cost hundreds to even thousands of dollars per webpage and require significantly more time to complete.

☑ **Finding 2:** On average, ReFLAIR uses about 562K tokens to analyze a webpage, costing around $1.18 with GPT-4.1. Despite the expense, ReFLAIR is able to accurately detect the majority of reflow issues, making it both effective and cost-efficient.

We measured the average runtime for different components throughout the process for ReFLAIR separately. The measurement breakdown includes: widget extraction and functionality extraction for the initial original page, widget and functionality extraction for the initial reflow page, initial widget matching between the two page versions, and page exploration activities. The page exploration stage consisted of two main activities, as described in Fig. 2: scrolling to discover additional content and expanding collapsed interface elements. During the page exploration stage, each newly discovered widget underwent the same extraction and matching processes as the initial widgets, with their processing times incorporated into the exploration measurements rather than tracked separately.

Widget extraction and functionality extraction dominate the overall execution time, consuming 73.19% of the total runtime when considering both the original page (41.50%, 4,819s) and reflow page (31.69%, 3,679s) processing stages. Page exploration activities account for 23.39% of the execution time, with scrolling operations requiring 12.40% (1,439s) and expanding operations consuming 10.99% (1,276s) of the total time. Widget matching represents the most efficient stage at only 3.43% (398s) of the total execution time. This timing profile suggests that the extraction and analysis of interactive elements represents the primary computational bottleneck in the testing pipeline, while the relatively lightweight widget matching stage indicates effective optimization in element correlation algorithms.

Cumulative runtime represents the total time when running one browser instance at a time, calculated by summing the time for all subcomponents. Based on the calculation, the average cumulative runtime is 11,610 seconds (3.23 hours). However, the process can be accelerated through parallelization. For example, element and functionality extraction for original and reflow pages can be executed concurrently, with the runtime determined by the slower of the two. Likewise, scrolling and expanding operations during page exploration can also run in parallel, reducing total runtime to the maximum of these operations rather than their sum. With this strategy, the average runtime decreases to 7,333 seconds (2.04 hours).

Since runtime and cost are directly correlated with the number of widgets processed by ReFLAIR, we detail the per-widget workflow. During functionality extraction, the page is refreshed and allowed up to 20 seconds to fully load and display the widget. Once the target widget is located,

the system scrolls to its position and waits 3 seconds for any dynamic content to render. After performing the click action, we allow an additional 10 seconds for any resulting changes before capturing a screenshot to document the widget's functionality. In total, this process introduces a maximum waiting time of 33 seconds per widget. On average, ReFLAIR analyzed 181 informative or actionable widgets per webpage, with minimum, median, and maximum counts of 15, 132, and 496 widgets, respectively. And the three webpages with the highest widget complexity were `qualtrics.com` (496 widgets), `espn.com` (446 widgets), and `sportybet.com` (438 widgets). If finer-grained parallelization were applied at the level of individual widget processing, 73.19% of the runtime could be reduced to under one minute. This optimization would further reduce the overall process to approximately 39.05 minutes.

☑ **Finding 3:** The runtime evaluation shows that ReFLAIR requires about 3 hours to analyze a webpage when all steps are executed sequentially. With full parallelization, however, 79.82% of the runtime could be reduced, cutting the total processing time to approximately 39.05 minutes. This highlights the scalability potential of ReFLAIR.

4.3 RQ3: Ablation Study

To study the contribution of each component of ReFLAIR to its overall effectiveness, we conduct an ablation study. Since intuitively, the reflow page itself preserves part of the original functionality, excluding it entirely would not be meaningful. As a result, our ablation study mainly focuses on evaluating the contribution of the page exploration stage. We designed three experiments:

- **No Exploration**: We removed the entire page exploration stage.
- **Scroll Only**: We disabled the expansion mechanism and retained only scrolling.
- **Expansion Only**: We disabled scrolling and retained only the LLM-based expansion mechanism.

Table 4 presents the results of our ablation study, which evaluates the contribution of the page exploration stage and sub-components of the stage. When the exploration stage is completely removed, precision drops substantially by 40.08%, even though recall increases slightly by 3.28%. This suggests that exploration is critical for filtering out false positives since many widgets from the original page are hidden in sub-menus in the reflow page. Using only scrolling reduces precision by 16.65% and increases recall by 1.64%, while using only the LLM-based expansion mechanism results in a 21.61% precision drop with a 1.17% recall gain.

Table 4. Impact of page exploration strategies on precision and recall.

	Precision	Recall
No Exploration	-40.08%	+3.28%
Scroll Only	-16.65%	+1.64%
Expansion Only	-21.61%	+1.17%

Recall increases when these steps are removed because the tool flags more widgets as missing after reflow, capturing more of the ground-truth cases. At the same time, precision decreases, since many of the additional widgets flagged as missing are false positives. Overall, these results demonstrate that both scrolling and expansion are complementary: neither alone is sufficient, and combining them yields the best balance of high precision and recall.

☑ **Finding 4:** The ablation study shows that removing the page exploration stage leads to a sharp 40.08% precision drop, while using only scrolling or only expansion still reduces precision by 16.65% and 21.61%, respectively. This confirms that both mechanisms are complementary, and their combination is essential to achieve the best overall performance.

4.4 RQ4: Generalizability, Scalability, and Robustness

This research question assesses the generalizability and scalability of ReFLAIR beyond the original dataset by examining its effectiveness across a broader range of domains. Complementing the primary experiments (RQ1–RQ3), which focus on effectiveness and ablation on D_p, RQ4 evaluates the feasibility, stability, and consistency of ReFLAIR under diverse domains and execution settings.

4.4.1 RQ4.1 — Generalizability and Scalability.

To evaluate the effectiveness of ReFLAIR under a broader domain, we conducted experiments on the extended dataset D_e.

Experimental Setup. We constructed our extended dataset (D_e) as follows. First, we examined the domains covered by D_p to identify underrepresented domains. We then selected 12 webpages based on Semrush [45] for those domains. For ground truth construction, to ensure data quality, a co-author independently verified the annotations to identify potential labeling discrepancies. To reduce the impact of randomness, we conducted two runs per webpage. Finally, we applied the same verification process detailed in Section 4.1.5 to verify the final results.

Result and Analysis. Our extended dataset (D_e) comprises 36 webpages, adding 12 webpages to the original 24-page primary dataset D_p. Overall, D_e spans 28 domains [45], with each page containing an average of 217 visible informative or actionable widgets.

On the extended dataset, ReFLAIR achieves a precision of 84.62% and a recall of 93.32%. ReFLAIR maintains robust performance with precision increased by 4.13% and recall decreased by 1.99%, indicating consistent effectiveness across diverse domains.

To evaluate scalability, we analyzed computational cost and runtime performance on D_e. The average token usage is 603K, corresponding to 1.07x of the 562K tokens required for D_p, with a per-page cost of $1.27, representing 1.07x of the $1.18 cost per page for D_p. The average runtime per webpage is 4.54 hours. In comparison, the sequential runtime under the same configuration for D_p is 3.23 hours, corresponding to an increase by a factor of 1.41x due to complex pages. As discussed in Section 4.2, ReFLAIR could benefit from parallelization to optimize execution times.

Both token usage and runtime on D_e increase in line with the higher number of visible informative or actionable widgets, which is 1.20x greater than in D_p, indicating a stable cost per widget. Although absolute costs rise for larger pages, the increase appears generally proportional, suggesting that ReFLAIR remains practically scalable for webpages across diverse domains.

☑ **Finding 5:** Experiments on the extended dataset D_e containing 36 webpages demonstrate that ReFLAIR maintains accuracy across 28 domains while exhibiting scalability in cost and runtime.

4.4.2 RQ4.2 — Cross Configuration.

To assess the generalizability of ReFLAIR, we conduct additional experiments focusing on the impact of LLM backends, viewport configurations, all of which may affect ReFLAIR's results.

Experimental Setup. To assess ReFLAIR's robustness, we ordered the 36 webpages by widget count and partitioned them into two equal strata (18 webpages each). From each stratum, we selected three webpages for evaluation across different LLM backends and viewport configurations. This approach balances evaluation coverage with annotation costs. We refer to this dataset as D_{xconf}, which contains an average of 337 informative or actionable widgets, underscoring the structural and functional complexity of the selected webpages. Using the default GPT-4.1 backend, the precision and recall on this subset were 96.20% and 91.24%, respectively. Owing to the exploratory nature of this feasibility study, each configuration was evaluated once.

LLM. To evaluate the portability of ReFLAIR across different LLM backends, we replaced the default GPT-4.1 model with Gemini-3-Pro. We selected Gemini-3-Pro as it represents a state-of-the-art multimodal alternative capable of processing both visual and textual inputs similar to GPT-4.1. We repeated the evaluation on D_{xconf} using identical prompts and default settings [28]. The results indicate that ReFLAIR maintains robust performance when transitioning between backends. Specifically, Gemini-3-Pro achieved a precision of 94.85% and a recall of 94.85%, which corresponds to a marginal decrease of 1.35% in precision and a slight increase of 3.61% in recall.

Viewport. In addition to the default setting, we evaluate a representative tablet viewport of 768 × 1024 CSS pixels, following commonly adopted industrial conventions [35–37]. Because changing the

viewport alters page layout and rendering behavior, reflow issues are re-annotated for the selected viewport, resulting in a newly constructed ground-truth version specific to this configuration. The ground truth for the extra viewport width is constructed using the same annotation protocol as $D_e - D_p$. On the tablet viewport, ReFLAIR achieves a precision of 92.50% and a recall of 91.36%, with both metrics exceeding 91%. These results provide evidence that ReFLAIR maintains strong performance beyond the 320-CSS-pixel layout and does not overfit to that specific viewport configuration.

☑ **Finding 6:** The effectiveness of ReFLAIR is not strictly tied to a single experimental setup. It remains adaptable to alternative LLM backends (i.e., Gemini-3-Pro) and different screen dimensions (i.e., 768 CSS pixels) with negligible impact on overall accuracy.

5 Discussion and Threats to Validity

5.1 Dynamic Content

ReFLAIR explicitly addresses lazy loading by automatically scrolling each page from top to bottom after the initial load to allow deferred content to render, followed by a fixed 10-second delay after each interaction before capturing screenshots. Manual inspection of five randomly sampled widgets from the 13 pages affected by dynamic loading showed that four were successfully extracted during Stage 1 of ReFLAIR, while the remaining case involved a non-actionable image without alternative text and was therefore outside the evaluation scope.

ReFLAIR supports basic dynamic interactions through automated scrolling and LLM-guided expansion of collapsed menus to reveal hidden content. While complex event-driven behaviors (e.g., multi-step workflows) are not explicitly modeled, such cases were rare in our dataset of high-traffic, real-world websites selected from Semrush [45]. The implemented mechanisms therefore cover most responsive patterns observed. Prior work likewise identifies highly dynamic content as challenging and often excludes it [53, 61, 62]. Handling richer interactions and fully dynamic interfaces is beyond the current scope and remains future work.

5.2 Threats to Validity

Internal Threats. The primary internal threat to validity is potential subjective bias or errors in ground-truth labeling. To mitigate this risk, after one author labeled the dataset, at least one additional authors independently verified the annotations. The disagreement rate[1] was 9.09% and 12.29% on D_p and 2.43% on $D_e - D_p$. All discrepancies were subsequently resolved through discussion until consensus was reached.

Another threat to external validity arises from our reliance on a specific cloud-based LLM (GPT-4.1). We accessed the model directly via the OpenAI API [12] to avoid third-party interference, and set the temperature to 0 to enhance determinism.

The final internal threat to validity stems from the data leakage. We additionally selected 5 websites published after the release date of our model, *gpt-4.1* [12]. These websites were retrieved from the newly registered domain name community list of June 1st, 2025 [63]. We verified their publication dates using the Wayback Machine [38], confirming that no archives existed before the model's release.

External Threats. One threat to the external validity concerns the generalizability of the selected subjects. To mitigate this threat, we employed a systematic project selection strategy, focusing on all (1) popular webpages, (2) prior datasets, and (3) newly registered webpages that exhibit diversity across multiple dimensions: functionality domains and webpage complexity. While downloaded webpage instances may deviate from their live online versions (e.g., missing some iconographic elements), we follow a systematic procedure to ensure the quality of the subjects.

[1]Calculated as $\frac{\textit{Number of Reflow Issues with Discrepancy}}{\textit{Total Number of Reflow Issues}}$.

Our procedure involved capturing webpages under two different viewport configurations and applying manual filtering criteria to exclude subjects that exhibited major issues, as mentioned in Section 4.1.2. Although offline execution modifies HTTP-level settings (e.g., CSP) and may impair interactivity, internal validity is preserved because both ground truth and ReFLAIR were evaluated under identical constraints on the same offline copies.

Another threat to external validity arises from our use of Firefox. We manually inspected 5 webpages (42 reflow issues) in both Firefox and Chrome. All issues were consistently observed across both browsers, suggesting minimal impact from browser-specific rendering in this study.

A final potential threat to the external validity of our study is that we mainly evaluated reflow accessibility issues at a viewport width of 320 CSS pixels. This is a limitation because responsive web design often applies different layout versions at varying viewport widths, and reflow issues may arise outside the 320-CSS-pixel configuration [71]. We selected 320-CSS-pixel, because it aligns with the WCAG evaluation protocol for detecting SC 1.4.10 violations [20] and is consistent with prior work [43]. Moreover, a 320-CSS-pixel viewport corresponds to a common accessibility setting: using a screen magnifier on a standard 1280-CSS-pixel-wide desktop display with 400% zoom [20]. Although our main evaluation focused on 320-CSS-pixel, we provide empirical evidence demonstrates that this methodology can be successfully extended to 768 CSS pixels.

6 Related Work

Responsive Web Design. Beyond ReDeCheck [70, 71], VFDetector [60] presents an automated technique for detecting visibility faults (i.e., inconsistent fault and covered error) in HTML5 webpages. Unlike VFDetector, which analyzes only event objects and their visibility properties, ReFLAIR leverages both source code and visual screenshots to identify potential reflow issues affecting information and functional accessibility, respectively. Due to the unavailability of the VFDetector, we exclude it from our evaluation. Walsh et al. proposed an automated method for detecting layout failures during regression testing by comparing responsive layout information from two webpage versions to identify differences and flag potential regressions [72].

Web Accessibility and Reflow Issue Detection. The WCAG have inspired the development of most existing accessibility checkers [13, 23, 29, 31, 39, 40, 47, 50, 64]. However, due to their rule-based nature and reliance on static analysis, these tools primarily understand webpage syntax and cover only a limited portion of WCAG success criteria. As a result, they often fail to adequately detect accessibility violations on webpages. SALAD [43] automatically detects reflow accessibility issues for keyboard users. Unlike SALAD, which focuses on keyboard actions, ReFLAIR targets GUI and click-based interactions. More recently, GenA11y [49] leverages LLMs to detect WCAG violations requiring both syntactic and semantic understanding. However, this approach only analyzes webpages statically, which is insufficient for accurately identifying reflow-related accessibility issues that require dynamic user interactions.

GUI Testing. A stream of research has focused on detecting visual inconsistencies that arise not from changing viewport sizes, but from differences in how browser rendering engines interpret identical HTML, CSS, and JavaScript code. X-PERT [44, 59] presents an automated technique for identifying Cross-Browser Incompatibilities (XBIs) in web applications. More recently, JANUS [80] introduced a novel test oracle called Visual Delta Consistency for detecting rendering bugs within browsers by comparing visual deltas across browsers.

Semantic understanding remains a key challenge in web testing. Prior work shows that methods relying only on text, DOM structure, or screenshots cannot reliably distinguish functionally identical or distinct pages [77]. Learning-based approaches, such as widget matching and multimodal GUI encoding, have been explored to infer semantic relationships, underscoring both the importance and difficulty of semantic reasoning in UI testing [41, 78]. Moreover, LLMs have emerged as a

promising approach by jointly reasoning over visual layout, textual content, and contextual cues, enabling higher-level semantic understanding for tasks such as test generation, oracle construction, and web GUI accessibility testing [49, 51, 73, 76]. This capability is critical for reflow issue detection, where assessing whether responsive layout changes preserve content and functionality requires semantic understanding beyond purely syntactic or visual similarity.

Despite these advances, prior work has not examined reflow issue detection from a GUI- and click-based perspective, leaving open how semantic equivalence can be assessed under responsive transformations using interactive signals.

7 Conclusion

Responsive Web Design has become indispensable as mobile devices now account for the majority of Internet traffic. The Web Content Accessibility Guidelines (WCAG) require that both information and functionality remain accessible during layout reflows. However, existing checkers [13, 23] and research tools [43, 49, 70, 71] remain limited, either due to their inability to process GUIs effectively or because they cannot capture the inherently dynamic nature of reflow.

In this work, we introduced ReFLAIR, a multimodal generative AI–based approach for detecting reflow issues in responsive web design. Unlike prior work, ReFLAIR performs dynamic, GUI-level analysis to capture both informational and functional losses during reflow. Our evaluation across 24 diverse webpages demonstrates that ReFLAIR achieves state-of-the-art effectiveness, improving precision by 20.49% and recall by 55.40% over the best existing tools. Moreover, it does so at reasonable computational cost, averaging 562K tokens ($1.18) per website, and within a practical runtime budget. Through an ablation study, we further showed that the page exploration stage—combining both scrolling and LLM-based expansion—is critical for balancing high precision and recall. Moreover, ReFLAIR demonstrates generalizability across diverse web domains. By expanding our evaluation to 36 webpages across 28 distinct domains, we confirmed that the approach maintains robust performance and scalability. We also demonstrated that ReFLAIR is independent of the underlying LLM backend, supports configurable viewport settings, and demonstrates consistent performance across these configurations.

Overall, ReFLAIR offers an effective, scalable, and cost-efficient solution for advancing accessibility testing in responsive web design. In the future, we plan to extend ReFLAIR to handle a broader range of dynamic interactions, explore integration with automated repair techniques, and evaluate its applicability across larger and more diverse datasets. These directions will help address current limitations and open opportunities for more comprehensive, multimodal accessibility testing.

8 Data Availability

Our source code of ReFLAIR and the dataset publicly available at [14].

9 Acknowledgments

This work has been supported in part by award numbers 2211790 and 2106306 from the National Science Foundation. We are grateful for the feedback from Prof. Thomas Zimmermann, Prof. Iftekhar Ahmed and the anonymous reviewers, which helped improve this work.

References

[1] [n. d.]. About Us | ACT-Rules Community — act-rules.github.io. https://act-rules.github.io/pages/about/. [Accessed 22-06-2025].

[2] [n. d.]. All WCAG 2.1 Techniques | WAI | W3C — w3.org. https://www.w3.org/WAI/WCAG21/Techniques/. [Accessed 22-06-2025].

[3] [n. d.]. Automatically Detecting Reflow Accessibility Issues in Responsive Web Pages. https://sites.google.com/usc.edu/salad/home?authuser=0. [Accessed 18-07-2025].

[4] [n. d.]. Download the fastest Firefox for Mac ever — firefox.com. https://www.firefox.com/en-US/browsers/desktop/mac/?redirect_source=mozilla-org. [Accessed 16-07-2025].

[5] [n. d.]. F102: Failure of Success Criterion 1.4.10 due to content disappearing and not being available when content has reflowed. https://www.w3.org/WAI/WCAG21/Techniques/failures/F102. [Accessed 02-07-2025].

[6] [n. d.]. From a humble beginning 35 years ago, the Web is now central to the daily lives of billions — w3.org. https://www.w3.org/blog/2024/from-a-humble-beginning-35-years-ago-the-web-is-now-central-to-the-daily-lives-of-billions/. [Accessed 30-06-2025].

[7] [n. d.]. GitHub - qualweb/cli: QualWeb command line interface — github.com. https://github.com/qualweb/cli. [Accessed 22-06-2025].

[8] [n. d.]. HTML Standard — html.spec.whatwg.org. https://html.spec.whatwg.org/multipage/dom.html#interactive-content. [Accessed 10-07-2025].

[9] [n. d.]. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/. [Accessed 18-07-2025].

[10] [n. d.]. Learn Responsive Design. https://web.dev/learn/design/. [Accessed 22-06-2025].

[11] [n. d.]. Mobile-first Indexing Best Practices | Google Search Central | Documentation | Google for Developers — developers.google.com. https://developers.google.com/search/docs/crawling-indexing/mobile/mobile-sites-mobile-first-indexing. [Accessed 30-06-2025].

[12] [n. d.]. OpenAI Platform — platform.openai.com. https://platform.openai.com/docs/models/gpt-4.1. [Accessed 18-07-2025].

[13] [n. d.]. QualWeb. https://qualweb.di.fc.ul.pt/evaluator/about. [Accessed 22-06-2025].

[14] [n. d.]. ReFLAIR. https://github.com/seal-hub/ReFLAIR.

[15] [n. d.]. Responsive web design. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design. [Accessed 10-07-2025].

[16] [n. d.]. Roboflow Documentation. https://docs.roboflow.com/. [Accessed 28-08-2025].

[17] [n. d.]. Selenium — selenium.dev. https://www.selenium.dev. [Accessed 16-07-2025].

[18] [n. d.]. Understanding Denial-of-Service Attacks. https://www.cisa.gov/news-events/news/understanding-denial-service-attacks. [Accessed 24-08-2025].

[19] [n. d.]. Understanding Success Criterion 1.1.1: Non-text Content | WAI | W3C — w3.org. https://www.w3.org/WAI/WCAG22/Understanding/non-text-content.html. [Accessed 03-07-2025].

[20] [n. d.]. Understanding Success Criterion 1.4.10: Reflow | WAI | W3C — w3.org. https://www.w3.org/WAI/WCAG22/Understanding/reflow.html. [Accessed 22-06-2025].

[21] [n. d.]. W3Schools.com — w3schools.com. https://www.w3schools.com/xml/xpath_syntax.asp. [Accessed 16-07-2025].

[22] [n. d.]. WAVE Accessibility Extension – Get this Extension for Firefox (en-US) — addons.mozilla.org. https://addons.mozilla.org/en-US/firefox/addon/wave-accessibility-tool/. [Accessed 22-06-2025].

[23] [n. d.]. WAVE Web Accessibility Evaluation Tools — wave.webaim.org. https://wave.webaim.org. [Accessed 22-06-2025].

[24] [n. d.]. Write helpful Alt Text to describe images | Digital Accessibility — accessibility.huit.harvard.edu. https://accessibility.huit.harvard.edu/describe-content-images. [Accessed 03-07-2025].

[25] 2025. ESPN. https://www.espn.com/. [Accessed 03-04-2025].

[26] 2025. NIH. https://www.nih.gov/. [Accessed 17-06-2025].

[27] 2025. WAVE Doc. https://wave.webaim.org/api/docs?format=html. [Accessed 29-08-2025].

[28] 2026. Gemini 3 Developer Guide. https://ai.google.dev/gemini-api/docs/gemini-3. [Accessed 11-02-2026].

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

[30] LLC Accessible.org. 2025. *Accessibility Services Pricing.* Retrieved Auguest 20, 2025 from https://accessible.org/pricing/

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

[32] Fernando Almeida and José Monteiro. 2017. The Role of Responsive Design in Web Development. *Webology* 14, 2 (2017).

[33] Angela. [n. d.]. The Importance of Responsive Design in Modern Web Development — europeanbusinessreview.com. https://www.europeanbusinessreview.com/the-importance-of-responsive-design-in-modern-web-development/. [Accessed 30-06-2025].

[34] anonymous. 2025. USGS. https://web.archive.org/web/20200603065656/https://www.usgs.gov/natural-hazards/earthquake-hazards/. [Accessed 29-08-2025].

[35] anonymous. 2026. Breakpoints. https://getbootstrap.com/docs/5.0/layout/breakpoints. [Accessed 19-01-2026].

[36] anonymous. 2026. medium. https://medium.com/theymakedesign/breakpoints-in-web-design-4e3b334066e8. [Accessed 19-01-2026].

[37] anonymous. 2026. Responsive Web Design - Media Queries. https://www.w3schools.com/css/css_rwd_mediaqueries.asp. [Accessed 19-01-2026].

[38] Internet Archive. 2025. Wayback Machine. https://web.archive.org/. [Accessed 10-06-2025].

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

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

[41] Chunyang Chen, Sidong Feng, Zhengyang Liu, Zhenchang Xing, and Shengdong Zhao. 2020. From lost to found: Discover missing ui design semantics through recovering missing tags. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW2 (2020), 1–22.

[42] Paul T Chiou, Ali S Alotaibi, and William GJ Halfond. 2021. Detecting and localizing keyboard accessibility failures in web applications. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 855–867.

[43] Paul T Chiou, Robert Winn, Ali S Alotaibi, and William GJ Halfond. 2024. Automatically detecting reflow accessibility issues in responsive web pages. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[44] Shauvik Roy Choudhary, Mukul R Prasad, and Alessandro Orso. 2013. X-PERT: Accurate identification of cross-browser issues in web applications. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 702–711.

[45] Semrush Company. 2025. Semrush, Online Marketing Can Be Easy. https://www.semrush.com/. [Accessed 01-04-2025].

[46] Barry Doyle and Cristina Videira Lopes. 2008. Survey of technologies for web application development. *arXiv preprint arXiv:0801.2618* (2008).

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

[48] This GmbH. [n. d.]. 8 Reasons Responsive Design is Essential in UX | THIS — this.work. https://this.work/en/knowledge/responsive-design/. [Accessed 30-06-2025].

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

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

[51] 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.

[52] Mailinator. 2025. Mailinator. https://web.archive.org/web/20170108005252/https://www.mailinator.com/. [Accessed 10-06-2025].

[53] Forough Mehralian, Ziyao He, and Sam Malek. 2025. Automated accessibility analysis of dynamic content changes on mobile apps. (2025).

[54] OpenAI. 2025. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/. [Accessed 29-08-2025].

[55] OpenAI. 2025. Introducing GPT-5 in the API. https://platform.openai.com/docs/models/gpt-5. [Accessed 29-08-2025].

[56] OpenAI. 2025. *Pricing*. Retrieved Auguest 20, 2025 from https://platform.openai.com/docs/pricing

[57] Accessibility Guidelines Working Group (AG WG) Participants. 2025. WCAG 2 Overview. https://www.w3.org/WAI/standards-guidelines/wcag/. [Accessed 04-09-2025].

[58] rocketfuel. [n. d.]. The Importance of Website Layout for SEO Success — launchux.com. https://launchux.com/2025/04/21/the-importance-of-website-layout-for-seo-success/. [Accessed 30-06-2025].

[59] Shauvik Roy Choudhary, Mukul R Prasad, and Alessandro Orso. 2014. X-PERT: a web application testing tool for cross-browser inconsistency detection. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. 417–420.

[60] Yeonhee Ryou and Sukyoung Ryu. 2018. Automatic detection of visibility faults by layout changes in HTML5 web pages. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 182–192.

[61] Navid Salehnamadi, Ziyao He, and Sam Malek. 2023. Assistive-technology aided manual accessibility testing in mobile apps, powered by record-and-replay. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–20.

[62] Navid Salehnamadi, Forough Mehralian, and Sam Malek. 2022. Groundhog: An automated accessibility crawler for mobile apps. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[63] Shreshta. 2025. Newly Registered Domain Names Community Lists. https://newly-registered-domains.whoisxmlapi.com. [Accessed 02-06-2025].

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

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

[66] Markel Vigo, Justin Brown, and Vivienne Conway. 2013. Benchmarking web accessibility evaluation tools: measuring the harm of sole reliance on automated tests. In *Proceedings of the 10th international cross-disciplinary conference on web accessibility*. 1–10.

[67] W3C. 2018. Scalable Vector Graphics (SVG) 2: The svg element. https://www.w3.org/TR/SVG2/struct.html#SVGElement.

[68] W3C. 2021. Accessible Rich Internet Applications (WAI-ARIA) 1.2: img role. https://www.w3.org/TR/wai-aria-1.2/#img.

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

[70] Thomas A Walsh, Gregory M Kapfhammer, and Phil McMinn. 2017. Automated layout failure detection for responsive web pages without an explicit oracle. In *Proceedings of the 26th ACM SIGSOFT international symposium on software testing and analysis*. 192–202.

[71] Thomas A Walsh, Gregory M Kapfhammer, and Phil McMinn. 2017. ReDeCheck: an automatic layout failure checking tool for responsively designed web pages. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 360–363.

[72] Thomas A Walsh, Gregory M Kapfhammer, and Phil McMinn. 2020. Automatically identifying potential regressions in the layout of responsive web pages. *Software Testing, Verification and Reliability* 30, 6 (2020), e1748.

[73] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering* 50, 4 (2024), 911–936.

[74] WHATWG. 2025. HTML Living Standard: The img element. https://html.spec.whatwg.org/multipage/embedded-content.html#the-img-element. Living Standard.

[75] WHATWG. 2025. HTML Living Standard: The img element. https://html.spec.whatwg.org/#images. Living Standard.

[76] Jiahong Xiang, Xiaoyang Xu, Xiaopan Chu, Hongliang Tian, and Yuqun Zhang. 2026. Empowering Autonomous Debugging Agents with Efficient Dynamic Analysis. *Proceedings of the ACM on Software Engineering* 3, FSE, Article FSE031 (July 2026). doi:10.1145/3797126

[77] Rahulkrishna Yandrapally, Andrea Stocco, and Ali Mesbah. 2020. Near-duplicate detection in web app model inference. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 186–197.

[78] Yakun Zhang, Wenjie Zhang, Dezhi Ran, Qihao Zhu, Chengfeng Dou, Dan Hao, Tao Xie, and Lu Zhang. 2024. Learning-based widget matching for migrating gui test cases. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

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

[80] Chijin Zhou, Quan Zhang, Bingzhou Qian, and Yu Jiang. 2024. Janus: Detecting Rendering Bugs in Web Browsers via Visual Delta Consistency. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 153–164.

Received 2026-02-25; accepted 2026-03-24