Automated Detection of Web Application Navigation Barriers for Screen Reader Users

Shubhi Jain, Syed Fatiul Huq, Ziyao He, Sam Malek
University of California, Irvine
Irvine, California, USA
shubhij1@uci.edu, fsyedhuq@uci.edu, ziyaoh5@uci.edu, malek@uci.edu

***Abstract*—An estimated 43.3 million people worldwide live with blindness and rely on screen readers (SRs) to access the web. To support accessible development, software teams often rely on automated tools like WAVE and Lighthouse to detect accessibility issues. However, these tools primarily rely on static rule-based analysis and are largely limited to detecting labeling errors relevant to screen reader users. They fail to capture dynamic accessibility issues—specifically, whether user interface (UI) elements can be located and activated using a screen reader, which is essential for accessing core webpage functionality. To address this gap, we present A11yNavigator, an automated accessibility testing tool that simulates screen reader navigation to detect UI elements that cannot be either (1) located or (2) activated via the screen reader. A11yNavigator leverages NVDA, one of the most widely used screen readers, and supports three common navigation strategies: Tab, Arrow, and Quick Navigation keys. We evaluate A11yNavigator across 26 real-world websites and demonstrate its effectiveness in uncovering issues missed by existing tools. Our results highlight its high precision and recall in detecting barriers that go beyond static analysis.**

***Index Terms*—Web, Software Accessibility, Software Testing, Assistive Technologies, Blind Users**

I. Introduction

The web has become a critical platform for accessing education, healthcare, government services, and other essential resources [1], [2]. Yet, it remains persistently inaccessible to over 16% of the global population with disabilities [3], including 43.3 million blind users relying on screen readers to access digital content [4]. One study [5] estimates that nearly 20% of web traffic may originate from users with disabilities, a number that is even higher for websites in domains like healthcare and senior services. Despite this, the state of web accessibility remains poor. A WebAIM evaluation conducted in February 2024 found that 95.9% of homepages among the world's top 1 million websites had accessibility errors, with an average of 56.8 issues per page when analyzed using the WAVE tool [6]. Alarmingly, this represents only a 1.9% improvement over previous years, highlighting the slow progress and persistent nature of these barriers. These findings underscore the urgent need for stronger policy interventions to address the persistent accessibility barriers that continue to impact users with disabilities.

To address these ongoing challenges, governments and organizations across various countries have introduced formal accessibility standards and guidelines to promote digital inclusion [7]–[11]. However, policy alone has not driven this shift. In recent years, a wave of high-profile lawsuits [12] against inaccessible digital products has pushed accessibility into the spotlight and emphasized the need for legal compliance. As a result, developers now face increasing pressure to ensure their applications are accessible to all users, including people with disabilities. This shift has placed greater focus on integrating accessibility evaluation into the software development process.

Currently, there are three main ways to evaluate web accessibility. The most comprehensive method is user testing [13] [14], where individuals with disabilities assess web interfaces using their preferred assistive technologies (ATs). This approach surfaces nuanced barriers rooted in real-world usage, device diversity, and individual variations in disability. However, despite its depth, user testing is costly, time-intensive, and difficult to scale within fast-paced development environments [15].

A more feasible alternative is manual testing, typically conducted by developers or accessibility specialists who explore applications using screen readers or other ATs. While this method enables expert-driven evaluation and targeted inspection, it remains labor-intensive and is often limited by the tester's familiarity with the wide spectrum of disability experiences [16]–[22].

To improve scalability, automated testing tools have become widely adopted. These tools apply rule-based checks to static DOM structures, verifying compliance with guidelines such as the Web Content Accessibility Guidelines (WCAG) [8]. Tools like WAVE [23], Lighthouse [24], and Axe [25] can identify issues such as missing alt text, low color contrast, or buttons that do not meet minimum size requirements. Unfortunately, these tools often focus on static, rule-based evaluations, which fail to address dynamic accessibility issues that arise during real-world user interactions.

Studies show that such tools may miss nearly 50% of the accessibility barriers experienced by real users [26]. For instance, individuals with visual impairments rely on screen readers, such as NVDA or JAWS on Windows [6], to sequentially navigate UI elements and activate them using keys like Enter or Space. If pressing these keys does not trigger a visible state change, such as opening a menu or submitting a form, the element is considered non-actionable, making the interface effectively inaccessible to screen reader users.

In this work, we address that gap by introducing

A11yNavigator —an automated accessibility navigator that simulates real-time interactions using NVDA, a widely used open-source screen reader [6], to detect accessibility barriers that only appear during real-time use. To ensure the simulation reflects how screen reader users navigate the web, we conducted a preliminary study and identified three common strategies: tab-based, down arrow, and quick-key navigation (e.g., 'b' for buttons, 'k' for links). A11yNavigator supports all three to mirror real-world scenarios.

The tool opens web pages in a browser and simulates keyboard navigation as performed by screen reader users. It then performs user actions like pressing Enter or Space and captures NVDA's output in real time. This process helps detect two key types of accessibility issues: **locatability** issues—elements that are not reachable through standard navigation paths—and **actionability** issues—elements that fail to respond when activated. By simulating real user behavior, A11yNavigator reveals dynamic, context-specific issues that static tools miss.

Our empirical evaluation across 26 real websites shows that A11yNavigator can detect around 200 accessibility issues that remain undetected by existing accessibility testing tools with a precision of 93.24% and recall of 98.97%.

This paper makes the following contributions:

- A novel, high-fidelity, and fully automated approach to accessibility evaluation that simulates real-time screen reader interactions to detect dynamic accessibility issues that are often missed by static tools.
- A publicly available implementation of this approach for Windows environments, called A11yNavigator [27], which integrates directly with the NVDA screen reader.
- An empirical evaluation on a diverse set of real-world websites, demonstrating that A11yNavigator detects around 200 accessibility issues that remain undetected by widely used automated tools.
- An analysis of the locatability and actionability failures uncovered by A11yNavigator, offering insights that can inform future research on dynamic accessibility testing and repair.

The remainder of this paper is organized as follows. Section II provides background on accessibility evaluation and screen reader fundamentals as well as motivating examples. Section III details the design of A11yNavigator. Section IV presents the evaluation results on real-world websites. The paper concludes with a discussion of threats to validity, related work, and future directions.

II. Background and Motivation

In this section, we describe the necessary background on web interfaces and screen reader navigation, and present motivating examples from our preliminary user studies.

A. Structure of Web Interfaces

Web user interfaces are rendered as a *Document Object Model (DOM)* [28], [29]—a tree-like structure where each node corresponds to an element on the page. These elements include interactive widgets like links, buttons, and form fields, as well as structural containers such as *div* and *section*. While some elements are informational and interactive, others may be purely decorative or used for layout.

Each DOM node can include various *HTML attributes*—such as *role*, *tabindex*, and *aria-label* [29]—that enhance accessibility by conveying semantic and interactive information to assistive technologies. For instance, a visually styled element might resemble a dropdown menu, but unless it includes appropriate roles (e.g., *role="combobox"*), states (e.g., *aria-expanded*), and keyboard handlers, it may not be operable or even perceivable to screen reader users.

To analyze and track elements across pages and interactions, we extract their location in the DOM using *XPath* [30]. XPath provides a unique structural path by referencing an element's tags, attributes, and position within the DOM hierarchy.

B. Screen Reader Navigation of Interactive Elements

Screen readers help blind users access web content by reading it aloud using synthesized speech. To do this, they rely on standardized *accessibility APIs* such as Microsoft Active Accessibility (MSAA) [31], IAccessible2 (IA2) [32], and UI Automation (UIA) [33]. These APIs expose details about on-screen elements, such as their roles (e.g., button, link), states (e.g., checked, expanded), and available actions. Screen readers use this information to create a structured representation of these elements, which it presents to users through speech as they navigate using the keyboard. This allows blind users to explore content, find interactive elements, and perform actions.

Screen readers offer several ways to navigate web content, as highlighted by a WebAIM survey [6], where the most common strategies to locate information include navigating through headers or using the find function. Since our paper focuses explicitly on interactive elements, we conducted a preliminary study to better understand how screen reader users locate and operate such elements in real-world scenarios.

To recruit blind participants, we posted an open invitation in Program-L [34], a discussion forum for visually impaired software users and programmers. Five respondents were selected based on matching availability and technology experience. The participants consisted of four males and one female, ages ranging from 25 to 64 years old. All of them reported to have 10 or more years of screen reader experience.

We conducted 30 minute user test sessions with the participants over Zoom. During these sessions, participants were asked to complete a series of realistic web browsing tasks while sharing their screens and following the Think Aloud protocol [35]. For the study, we selected the website Agoda [36]—an online booking platform that provides hotel reservations, flights, and vacation rentals globally. None of the participants had used the website before. Participants were given the following instruction: "*As you navigate the webpage, please think aloud. Describe what you are doing and experiencing, including which keys you are using for navigation. Feel free to share anything that feels difficult or frustrating*". They were then asked to complete the following three tasks, each task building upon the previous one with additional web components to navigate:

1) Search for hotels in Sydney for your stay from today to March 1, for 2 adults.
2) Go back and now search for hotels in Tokyo for a stay from March 1 to March 7, for 1 adult.
3) Go back and search for flights from Sydney to Tokyo on March 1.

During each task, we observed their keypresses, navigation strategies, and how their approach changed when performing similar actions again. We documented the challenges they faced, any concerns they expressed, and their default navigation styles. For users unfamiliar with a website, we noted how they approached it for the first time, including strategies for locating interactive elements. We also examined their change in behavior during a second task on the same site, revealing how familiarity shapes navigation. After participants completed the task, we conducted a five-minute reflection session, asking them to describe the challenges they faced and the navigation modes they relied on. From these reflections and our observations, we identified three main keyboard-based navigation strategies used by screen reader users.

- **Arrow key navigation** lets users read through content line by line, including both interactive and non-interactive elements. In our study, participants commonly used this strategy when visiting a website for the first time, relying on the Down Arrow to get a sense of the page layout before taking action.
- **Quick Navigation keys** are single-key shortcuts (e.g., 'E' for edit fields, 'B' for buttons) that jump directly to specific types of elements. Participants familiar with the page layout or assigned task often used these keys to quickly access relevant controls (e.g., jumping directly to the input field to type in the departure location when searching for flights).
- **Tab navigation** moves focus sequentially across interactive elements such as links, buttons, and input fields. We observed that some users used the Tab key to systematically explore all focusable elements, especially when they were unsure of the element's type or location.

After navigating to the target interactive elements, users would either use Space or Enter to activate that element, e.g., clicking a button, expanding a dropdown menu, toggling a checkbox and more. A successful use of a feature includes both locating an element and activating it for its intended use.

C. Accessibility Issues of Interactive Elements - Motivating Examples

Our work focuses on the accessibility issues of interactive elements, when screen reader users cannot locate or activate such elements, rendering the intended features inaccessible. We present motivating examples to illustrate how these issues manifest and how they go undetected by conventional tools.

Figure 1(a) shows the homepage of Stack Overflow [37], a popular platform for software developers to ask questions, share knowledge, and find solutions to programming problems. The site plays a central role in the global developer community, with over 100 million monthly visits, among whom, 1.7% visitors are reportedly blind or have low vision [38].

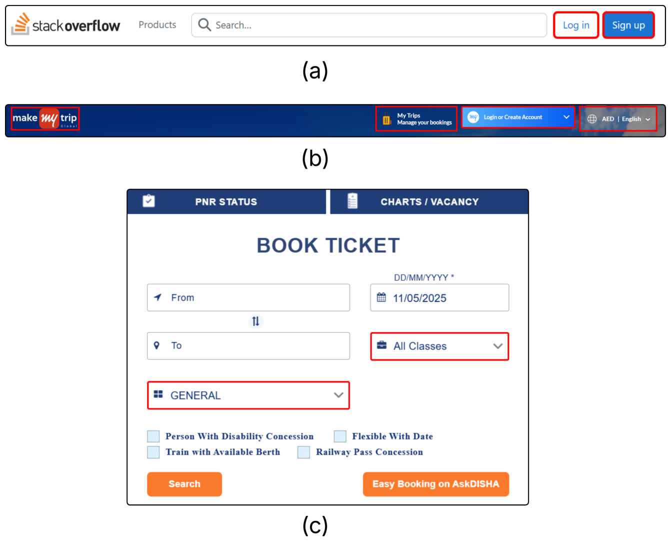

Fig. 1: (a) Stack Overflow: Log in and Sign up buttons are unreachable, (b) MakeMyTrip homepage: Header elements are skipped, (c) IRCTC homepage: Quota and Class dropdowns fail to respond to keyboard activation.

The homepage header includes key actions like "Log in" and "Sign up," highlighted in red, which are critical for initiating and participating in discussions. However, when using NVDA's quick key navigation strategy introduced in Section II-B, it skips the "Log in" and "Sign up" buttons. These essential entry points are inaccessible through standard screen reader shortcuts, posing a critical barrier for blind users.

Another issue appears in Figure 1(b), which displays the homepage of MakeMyTrip [39], a popular travel platform used globally to book flights and manage trips. The header includes essential options like "Login" and "My Trips," marked in red, which are key for accessing user accounts and bookings. However, when navigating with the Tab key, NVDA completely skips the header, leaving these critical controls inaccessible.

These examples illustrate a **locatability issue**—where critical interface elements are visually present and exist in the DOM, yet remain unreachable through standard screen reader navigation methods like quick keys or tabbing.

When evaluated using WAVE [23], a popular accessibility checker for web, both websites pass without flagging the above-mentioned navigation issues. This is because WAVE considers the structural validity of the DOM elements, checking whether they follow proper HTML and ARIA markup. This makes WAVE an effective tool for detecting static issues like missing alt text, low color contrast or small font size. But it cannot detect dynamic issues, where screen reader users are unable to locate elements.

Figure 1(c) shows the booking interface on the IRCTC website [40], a travel booking website with over 10 million monthly visits and a consistent top ranking in the "Airlines" category. The platform allows users to search for trains, check availability, and book tickets, offering a range of travel-related services through its interface. Two key dropdown menus, "Quota" and "Class", allow users to select the appropriate seat category, such as Senior Citizen or Person with Disability, and choose their preferred level of travel comfort. A sighted user can easily interact with these by clicking and choosing from the available options. A blind user using NVDA, however, must rely entirely on keyboard-based interactions. While both dropdowns are detectable and focusable using NVDA's navigation keys, they fail to respond when the user presses "Enter"—the expected action to expand and interact with the dropdown. No visual or auditory feedback is provided, and the menus remain closed. Though present in the DOM and visually styled as interactive, these elements are functionally inaccessible through the screen reader, blocking the user from proceeding with booking a flight.

This represents an **actionability issue**: the user can reach the element, but cannot perform the intended interaction. Similar to the previous issue, when the page is evaluated using WAVE, no issues are flagged. The dropdowns are structurally valid, containing proper HTML syntax, which leads to static tools overlooking its inaccessible behavior.

These examples highlight the limitations of static checks in detecting interaction failures. Interaction issues often go unnoticed unless interfaces are evaluated the way users with disabilities experience them—through assistive technologies. This underscores the need for accessibility evaluation methods that reflect real-world usage, not just structural correctness.

III. Approach

Despite the diversity of modern web interfaces, the ability to locate and interact with elements remains fundamental to accessibility. A11yNavigator addresses this challenge by evaluating two key dimensions: locatability—whether an element can be reached—and actionability—whether it can be interacted with. The goal of A11yNavigator is to automatically identify websites that violate these principles during realistic, screen reader-based navigation.

A11yNavigator consists of three primary components: the ClickElement Extractor, the User Simulator, and the Issue Detector—which together support two distinct detection strategies: locatability and actionability. While these strategies operate independently, they share a common infrastructure. The **ClickElement Extractor** analyzes the webpage's DOM to extract all clickable elements, forming a baseline for evaluation. The **User Simulator** then emulates real-world screen reader behavior using three navigation strategies—Tab, Down Arrow, and Quick Key—to traverse the website, while also simulating user actions such as pressing Enter or Space. Finally, the **Issue Detector** analyzes the collected data to identify locatability and actionability issues. We describe each strategy approach in the following subsections.

A. Locatability Detection Approach

To detect locatability issues, A11yNavigator first extracts all clickable elements using the ClickElement Extractor, forming a ground truth. It then simulates screen reader navigation strategies through the User Simulator to identify which elements are reachable. Comparing the two sets reveals elements that are present in the DOM but inaccessible by screen readers. Figure 2 illustrates an overview of detecting locatability issues.

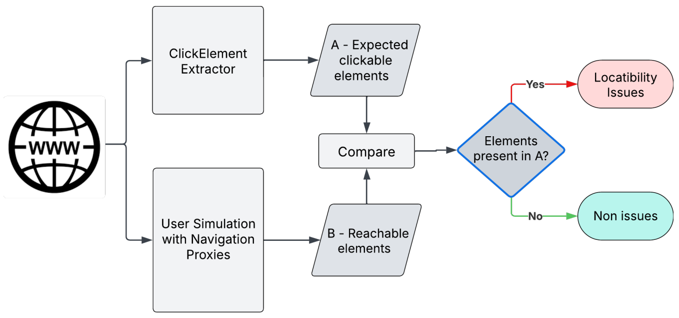

Fig. 2: Workflow of Locatability Detection.

1) ClickElement Extractor: To establish a ground truth of clickable elements, A11yNavigator first selects standard interactive elements, such as `<a>`, `<button>`, and those with ARIA roles like `role="button"`, `role="link"`). It then considers custom elements—like `<div>` or `<span>`—that are not inherently interactive but have JavaScript event listeners attached [41], indicating they are intended to be clickable. In both cases, elements that are visually or semantically hidden are filtered out.

For each valid element, A11yNavigator extracts its XPath, accessible name [42] (from `text`, `aria-label`, `alt`, or `placeholder`), and relevant attributes such as `id` and `href`. This process captures both standard and custom interactive elements, ensuring that even non-semantic widgets [7] designed to behave like buttons or links are included. The result is a list of elements that should be reachable and actionable by users, forming the baseline for comparison during simulated navigation.

2) User Simulation with Navigation Proxies: To evaluate accessibility in real-world contexts, A11yNavigator simulates the three main navigation modes used by screen reader users with NVDA. For each mode, it records the XPath of every element visited, which the Issue Detector later analyzes to identify accessibility problems.

Tab Key Navigation Proxy. This proxy simulates navigation by Tab key which sequentially moves through focusable elements. It operates in two main phases:

- Navigation and Data Collection: As the proxy simulates Tab key presses, NVDA focuses on each element and logs its accessibility attributes such as role, accessible name, state (e.g., checked, expanded), and IA2UniqueId [32] into a structured file.
- XPath Extraction: For each focused element, the proxy simultaneously executes a JavaScript snippet to extract its XPath for later analysis.

To handle composite widgets like tab panels and radio groups, the proxy also simulates additional keys (e.g., arrow keys) as defined by standard keyboard interaction patterns [43], ensuring all reachable sub-elements are captured.

Quick Key Navigation Proxy. The Quick Key Navigation Proxy simulates NVDA's shortcut keys [44] that allow users to jump directly to specific element types like links, buttons, and form fields. This efficient strategy avoids sequential navigation and, like the Tab Key Proxy, operates in two main stages:

- Navigation and Data Collection: The proxy simulates only those NVDA single-key shortcuts that correspond to focusable elements (e.g., k for links, b for buttons, x for checkboxes, f for form fields). Each key press prompts NVDA to move focus to the next element of the specified type, capturing its accessibility properties.
- XPath Extraction: For each focused element, the proxy extracts its XPath using JavaScript at the moment NVDA shifts focus.

To ensure complete coverage, it supports both forward (using quick keys) and backward (using Shift + key) navigation. This is important because, unlike Tab or Down Arrow, Quick Navigation involves multiple keys (e.g., 'k' for links, 'b' for buttons). If a user reaches the end of the page with one key and switches to another (e.g., from 'k' to 'b'), they may miss earlier elements unless backward traversal is also performed.

Down Arrow Navigation Proxy. The Down Arrow Navigation Proxy mimics how screen reader users read content line by line using the Down Arrow key, representing the most comprehensive but time-intensive navigation pattern [45]. To handle the unique challenges of sequential reading, this proxy employs a specialized processing pipeline.

- Navigation and Data Collection: The proxy sends repeated Down Arrow key presses to simulate how users explore content sequentially. As NVDA reads each line, it announces the corresponding element (for example, 'link Submit', 'heading level 2'). These announcements are written to a file by our NVDA extension.
- Preprocessing NVDA Announcements: Since NVDA's speech output includes auxiliary descriptors (e.g. 'heading', 'clickable') or splits long labels into multiple announcements, we apply a pre-processing step to improve name precision and reduce noise. This includes: (a) filtering non-essential terminology (e.g., "heading," "link," "clickable"), and (b) merging fragmented announcements (e.g., when a long link text is announced in multiple parts). This step optimizes the announcement data, making it more accurate for matching elements in the DOM.
- XPath Extraction: We then use the preprocessed announcement names as input to identify matching elements in the HTML. These names are compared against accessible properties such as text content, ARIA labels, titles, placeholders, and alt text. For each matched element, we extract its XPath, which serves as a unique identifier for further analysis.

Unlike tab-based navigation, where arrow keys within composite widgets can shift focus between interactive elements (e.g., between tabs), down-arrow traversal does not move DOM focus. It simply advances NVDA's virtual cursor line by line, without triggering browser-level focus events [44].

All three navigation proxies run independently but produce a standardized output containing element XPaths and associated accessibility properties. This is forwarded to the Issue Detector component, which performs a comparative analysis to identify locatability issues specific to each navigation method.

For each navigation technique, A11yNavigator also records efficiency metrics, including:

- Number of keystrokes needed to reach each element
- Success rate, defined as the proportion of clickable elements successfully reached during navigation
- Time taken to traverse the website

These metrics help compare the efficiency of different navigation methods and reveal usability issues that may exist even when elements are technically accessible. We will discuss this in more detail in Section 5.

3) Comparison and Issue Detection: In the final stage, A11yNavigator detects locatability issues by comparing the elements visited during simulated navigation with the complete set of clickable elements extracted in the first phase. For each navigation strategy (Tab, Quick Key, and Down Arrow), it records the XPath of every traversed element and flags any expected clickable elements that were not reached.

Tab and Quick Key Navigation. For both Tab and Quick Key navigation, A11yNavigator uses a direct XPath comparison. It compares the ground truth list of clickable element XPaths with those reached during simulation. Any elements in the ground truth list that were not visited are marked as locatability failures.

Down Arrow Navigation. Down Arrow navigation presents unique challenges for comparison, as it relies on screen reader announcements rather than direct focus events to extract the XPath of elements. Since XPath cannot be captured through focus in this mode [44], we infer it by matching the accessibility properties exposed by NVDA—such as the accessible name, id, and href—with elements in the DOM. To identify reachable elements, we compare both the inferred XPath and key accessibility properties of each announced element with the ground truth. An element announced during Down Arrow navigation is considered successfully reached if its XPath and accessibility properties match those in the ground truth. Otherwise, the element remains in the ground truth list and is flagged as a locatability issue.

The above approaches enable A11yNavigator to detect elements that are present in the DOM but remain inaccessible through real-world screen reader navigation strategies.

B. Actionability Detection Approach

While locatability checks whether screen reader users can find the clickable elements, actionability ensures those elements work as expected when activated. An element might be easy to navigate to but still fail to perform its intended action—posing a serious accessibility issue. This section outlines A11yNavigator's approach to detecting actionability failures. An overview is shown in Figure 3.

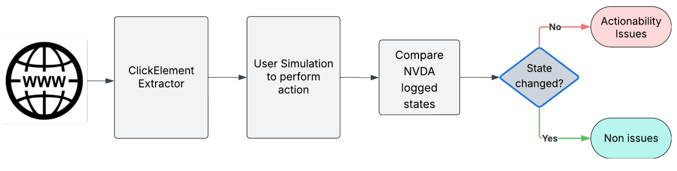

Fig. 3: Workflow of Actionability Detection.

1) User Simulation: A11yNavigator validates each clickable element by simulating real user interactions and observing screen reader feedback. Using the list of elements identified by the ClickElement Extractor (Section III-A1), it performs the following steps for each element: (a) brings the element into focus using its XPath, (b) simulates an appropriate interaction (e.g., pressing Enter or Spacebar) based on the element type, and (c) records NVDA's accessibility state before and after the interaction. This process closely mimics how actual screen reader users interact with web elements, allowing for realistic validation.

2) Comparison and Issue Detection: After the simulation, A11yNavigator analyzes the before-and-after states captured by NVDA to detect actionability issues. It looks for changes in accessibility-related states, such as expansion (e.g., COLLAPSED → EXPANDED) or selection (e.g., UNCHECKED → CHECKED). If no state change is detected after the interaction, the element is flagged to have an actionability issue. For example, if a dropdown remains COLLAPSED after activation, it indicates a failed interaction—even though the element appears clickable. This process allows A11yNavigator to identify elements that are clickable but fail to respond to screen reader actions.

IV. Evaluation

To comprehensively evaluate our approach, we designed a study around the following four research questions:

- **RQ1:** How effectively does A11yNavigator identify accessibility barriers?
- **RQ2:** How does A11yNavigator compare with existing static tools such as WAVE?
- **RQ3:** What are the characteristics of the accessibility issues identified by the tool, and how do they impact user interaction and task completion?
- **RQ4:** What is the performance of A11yNavigator?

A. Implementation Details

A11yNavigator is a modular Python-based framework that simulates real-user keyboard navigation and captures screen reader feedback to detect accessibility barriers on websites. It follows a two-part architecture: a browser automation engine that interacts with web content, and a screen reader interface that logs speech output and accessibility states. All modules run locally on a Windows 10 Pro machine with an Intel Core i7-7660U CPU @ 2.50GHz and 16 GB RAM, using NVDA version 2024.1 as the external assistive technology.

The automation backend leverages Selenium WebDriver [46] to load websites, extract DOM metadata, and focus elements via JavaScript. To simulate user input, it uses the keyboard library [47] to send native keypresses like Tab, enabling realistic interactions that trigger screen reader responses.

To capture real-time accessibility feedback, we extend NVDA's open-source screen reader by modifying its core components—specifically the speech and inputCore modules. The speech module is instrumented to log synthesized speech output, while inputCore is used to intercept user input and extract metadata associated with the currently focused UI element. This includes properties such as accessible name, role, IA2UniqueID [32], and state. All data is logged in structured JSON files for post-analysis of element behavior and accessibility coverage.

To ensure accurate evaluation, we refine the ground truth set of clickable elements by filtering out those that are disabled, hidden via aria-hidden, not displayed (e.g., display: none), or visually hidden through zero dimensions. For Down Arrow navigation, we also excluded elements missing accessible names or fallback properties (e.g., aria-label, alt, or value), as such issues are already detected by static tools. A11yNavigator focuses primarily on interactive HTML elements—such as links, buttons, form controls, and widgets—in alignment with WCAG 2.2 Keyboard Accessibility (2.1.1) [48], which prioritizes keyboard operability. However, the framework is designed to be extensible and can be adapted to analyze non-interactive content like static text and headings for broader accessibility coverage.

B. Experiment Setup

To build a representative dataset of subject websites, we followed the website selection strategy proposed by Tafreshipour et al. [26]. Specifically, we referred to Semrush's list of most visited websites [49] across various categories such as e-commerce, finance, government services, and education.

From this list, we selected websites with monthly traffic exceeding 10 million visits, ensuring that our evaluation covers widely-used platforms that impact large user populations. All selected websites were publicly accessible and primarily in English. In total, our dataset includes 26 websites spanning 19 categories, each containing a variety of interactive HTML elements (e.g., links, buttons, forms).

Table I provides an overview of the selected websites along with their corresponding categories, as shown in the "Website" and "Category" columns.

C. RQ1. Effectiveness of A11yNavigator

To evaluate the effectiveness of A11yNavigator in detecting accessibility barriers, we tested it across two screen configurations: desktop mode and compact mode. Desktop mode used the default full-width layout, while compact mode followed WCAG 2.2 Reflow guidance (Success Criterion 1.4.10) [48] by applying 400% zoom, simulating a viewport

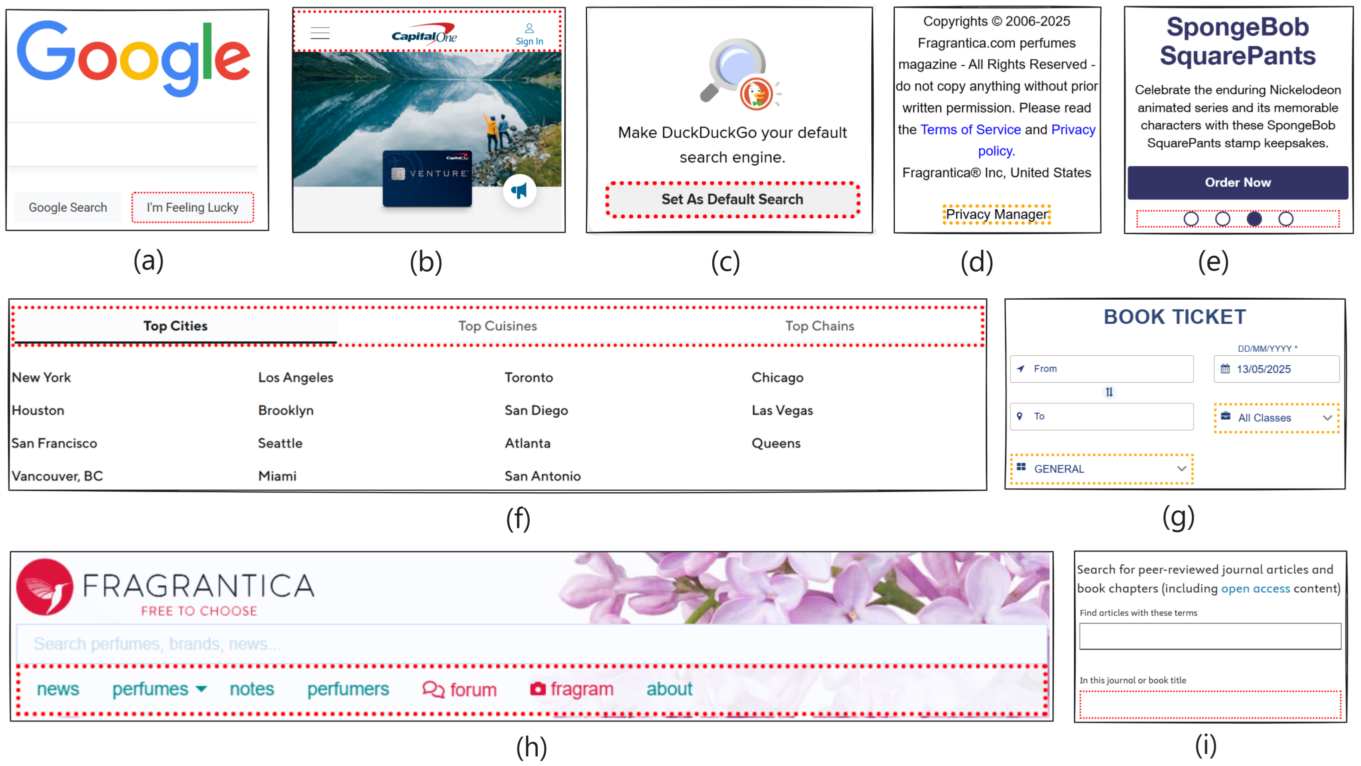
Google
Google Search
I'm Feeling Lucky
(a)
Capital One
Sign In
VENTURE
(b)
Make DuckDuckGo your default search engine.
Set As Default Search
(c)
Copyrights © 2006-2025 Fragrantica.com perfumes magazine - All Rights Reserved - do not copy anything without prior written permission. Please read the Terms of Service and Privacy policy.
Fragrantica® Inc, United States
Privacy Manager
(d)
SpongeBob SquarePants
Celebrate the enduring Nickelodeon animated series and its memorable characters with these SpongeBob SquarePants stamp keepsakes.
Order Now
(e)
Top Cities
Top Cuisines
Top Chains
New York
Los Angeles
Toronto
Chicago
Houston
Brooklyn
San Diego
Las Vegas
San Francisco
Seattle
Atlanta
Queens
Vancouver, BC
Miami
San Antonio
(f)
BOOK TICKET
DD/MM/YYYY *
From
13/05/2025
To
All Classes
GENERAL
(g)
FRAGRANTICA
FREE TO CHOOSE
Search perfumes, brands, news...
news
perfumes
notes
perfumers
forum
fragram
about
(h)
Search for peer-reviewed journal articles and book chapters (including open access content)
Find articles with these terms
In this journal or book title
(i)

Fig. 4: (a) is an example of a false positive, (b-h) are examples of locatability and actionability issues in A11yNavigator, and (i) is an example of a false negative

width of 320 CSS pixels. This dual-mode evaluation ensures that websites remain accessible across varying screen sizes and zoom levels. We focused on four key issue categories:

- **Unlocatable (Tab)**: Focusable elements skipped during Tab navigation.
- **Unlocatable (Quick-Key)**: Elements missed when using single-key shortcuts (e.g., 'B' for buttons, 'K' for links).
- **Unlocatable (Down Arrow)**: Elements not encountered during sequential traversal with the Down Arrow key.
- **Unactionable**: Elements that are reached and announced by NVDA but do not respond to expected keyboard interactions like Enter or Space.

To validate the output of A11yNavigator, we conducted a manual review involving three authors and one external evaluator. Two authors have over four years of accessibility research experience, while the third and the external evaluator have 3.5 years of web industry experience. The evaluators opened each webpage in Chrome Incognito Mode on a Windows machine and highlighted both the clickable elements and the elements flagged by A11yNavigator. They then simulated each navigation strategy—Tab, Down Arrow, and Quick Key—using NVDA to confirm whether the flagged elements were unreachable. To assess actionability, they attempted to activate each element using Enter and Space keypresses.

Table I summarizes the total number of issues detected by A11yNavigator and those manually verified as true positives, from which we computed precision to assess its accuracy. We also report recall separately to measure how thoroughly A11yNavigator captures accessibility barriers. Together, these metrics provide a quantitative evaluation of how effectively A11yNavigator detects real-world locatability and actionability issues across diverse websites.

1) Precision: We assessed precision by verifying each reported issue in both desktop and compact views. As shown in Table I, the tool achieved an overall precision of 93.24% across both locatability and actionability issues. This indicates that the vast majority of reported issues reflected genuine problems experienced by screen reader users.

However, our analysis also revealed false positives, often caused by the dynamic behavior of modern websites. One recurring source of error was carousels, which periodically cycles through a list of cards. During ground truth extraction, A11yNavigator would capture the XPath of one card. But by the time the navigation proxy would reach that region, the card would change, the XPath now pointing to a different element. As a result, the original element appeared unreachable. Another example was observed on Google's homepage 4(a), where the "I'm Feeling Lucky" button changes its label to "I'm Feeling Hungry" in red when it receives keyboard focus. This dynamic label update shifts the focus to a different DOM element with a new XPath, causing a mismatch with the ground truth and leading A11yNavigator to incorrectly flag the original element as unreachable.

One notable observation in Table I is that Down Arrow navigation detected far fewer accessibility issues than other strategies. This is likely caused by its linear traversal. Unlike Tab or Quick Keys, which jump between semantic elements, the Down Arrow moves through every element in sequence, interactive or otherwise, reducing the chance of missing elements due to incorrect roles or structure.

At first glance, this may suggest that almost all clickable elements are reachable through at least one mode of navigation, causing users no major barriers when traversing websites. While comprehensive, it is time-consuming to navigate relying only on the Down Arrow.

We analyzed the interaction effort required by each navigation mode, computing the average number of keypresses per website and reporting the median across all 26 websites. The results showed clear differences: Quick Navigation was the most efficient, with a median of 28.5 keypresses, followed by Tab with 41.5. In contrast, Down Arrow was the most exhaustive, requiring a median of 100.5 keypresses, over three times the effort of Quick Keys. This suggests that while Down Arrow provides comprehensive coverage, it is highly inefficient for routine navigation [45].

Our preliminary study also confirmed that users tend to prefer quicker and more targeted strategies like Quick Keys or Tab over the exhaustive Down Arrow approach. These findings underscore the need for developers to support all navigation methods, as users choose strategies based on their needs and the specific browsing context.

2) Recall: To assess the recall of A11yNavigator in detecting real accessibility issues, we categorized the 26 subject websites into large, medium, and small based on the number of elements contained and conducted a detailed manual evaluation on a representative subset of six websites—two from each category. Full-scale recall verification across all 26 websites was impractical due to the intensive manual effort involved: each website required an average of 200 key presses across all navigation modes. Moreover, the evaluation had to be performed in both desktop and compact views with all results independently verified by three authors and reviewed by an external evaluator.

Within this subset, A11yNavigator successfully detected all but two issues, resulting in an overall recall of 98.97%. Both false negatives occurred on the ScienceDirect website (Figure 4(i)) during Down Arrow navigation—one in desktop view and one in compact view. These misses were caused by the page's repetitive DOM structure combined with overlapping accessibility attributes. In particular, nested containers reused identical classes and ARIA roles, which made multiple elements expose nearly indistinguishable accessible properties. Because of this ambiguity, A11yNavigator extracted the XPath for the wrong element, causing the actual issue to go undetected.

D. RQ2. Comparison with WAVE

To evaluate the usefulness of A11yNavigator, we compared it with WAVE [23], the most widely adopted accessibility testing tool [50], [51], which analyzes HTML and CSS against WCAG rules to identify issues such as low contrast, missing or overly long alternative texts, and small font sizes. However, WAVE performs static analysis without simulating user interaction. In contrast, A11yNavigator leverages assistive technologies (AT)—specifically the NVDA screen reader—to perform a dynamic evaluation that mirrors real user navigation. This allows A11yNavigator to detect interaction-level barriers, such as elements that are unreachable or unresponsive during screen reader traversal, which static tools like WAVE often miss. These dynamic issues directly impact blind users' ability to access functionality and align with WCAG Success Criterion 2.1.1, which requires all content to be operable via keyboard. By running both tools on the same set of web pages, we illustrate how static and dynamic methods detect complementary sets of issues rather than overlapping ones. WAVE excels at uncovering visual and structural violations, while A11yNavigator reveals interaction-level failures that static tools inherently miss. This comparison matters because, in practice, developers should ideally use both approaches together to achieve a comprehensive accessibility evaluation. To the best of our knowledge, no other fully automated dynamic testing tools currently exist for a one-to-one comparison (as discussed in Section VI). While previous research explored aspects of dynamic evaluation—such as long navigation paths, invalid ARIA labels, or keyboard traps—these efforts do not provide an automated, end-to-end solution for detecting unreachability and inoperability during screen reader navigation, as A11yNavigator does.

E. RQ3. Characteristics and Impact of Issues

To better understand the root causes of accessibility barriers detected by A11yNavigator, we analyzed the types of implementation patterns that commonly led to locatability and actionability issues. These patterns often stemmed from incorrect use of roles, missing attributes, or non-standard UI components.

a) Unlocatable elements: One of the most prevalent issues we identified was the use of non-focusable elements as interactive controls. Developers frequently implemented clickable regions using non-sematic elements such as `<div>` or `<li>` tags with attached JavaScript listeners, without setting the `tabindex` attribute. For example, Figures 4(e) and 4(f) illustrate this issue on USPS and Doordash, respectively. On Doordash, elements like "Top Cities," "Top Cuisines," and "Top Chains" appear visually interactive but are skipped by screen readers during tab and quick-key navigation due to their lack of focusability and semantic roles. We observed similar problems on several high-traffic websites, including Discord, Genius, NIH, Zerodha, MakeMyTrip, IRCTC, and ScienceDirect.

Another frequent issue involves anchor tags missing href attributes. Developers often use `<a>` tags as buttons by attaching JavaScript event listeners, but omit the href. Without it, these elements lose their built-in focusability and keyboard accessibility. As a result, they may appear visually as links or buttons but remain invisible to screen reader users during

TABLE I: Evaluation of websites with the details of detected accessibility issues by A11yNavigator

Id	Website	Category	View Mode	#Traffic	Unlocatable (Tab)		Unlocatable (Quick-key)		Unlocatable (Down Arrow)		Unactionable		All Issues		WAVE
					Total	TP	Total	TP	Total	TP	Total	TP	Total	TP	
1	ADP	Human Resources	Desktop	>50M	0	0	3	3	2	2	0	0	5	5	10
			Compact	>50M	0	0	0	0	2	2	0	0	2	2	11
2	Agoda	Hospitality	Desktop	>10M	1	0	1	1	0	0	0	0	2	1	10
			Compact	>10M	1	0	0	0	0	0	0	0	1	0	10
3	Capitalone	Banking	Desktop	>50M	0	0	0	0	0	0	0	0	0	0	28
			Compact	>100M	0	0	3	3	3	3	0	0	6	6	28
4	Chase	Banking	Desktop	>100M	0	0	8	8	0	0	0	0	8	8	4
			Compact	>10M	0	0	0	0	0	0	0	0	0	0	4
5	Craigslist	Real Estate	Desktop	>50M	6	5	1	0	0	0	0	0	7	5	7
6	Discord	Computer Software & Development	Desktop	>500M	0	0	0	0	0	0	0	0	0	0	10
			Compact	>100M	1	1	1	1	0	0	0	0	2	2	10
7	Doordash	Restaurants	Desktop	>10M	5	5	3	3	0	0	0	0	8	8	7
			Compact	>10M	6	5	3	3	0	0	0	0	9	8	6
8	Doubleclick	Online service	Desktop	>10M	0	0	0	0	0	0	0	0	0	0	8
			Compact	>100M	0	0	0	0	0	0	0	0	0	0	4
9	DuckDuckGo	Food & Beverages	Desktop	>100M	1	1	1	0	0	0	0	0	2	1	3
			Compact	>1B	1	1	1	0	0	0	0	0	2	1	2
10	Formula1	Automative	Desktop	>10M	0	0	0	0	1	0	0	0	1	0	34
			Compact	>10M	0	0	0	0	1	0	0	0	1	0	34
11	Fragrantica	Beauty and Cosmetics	Desktop	>10M	0	0	7	7	0	0	1	1	8	8	381
			Compact	>10M	0	0	0	0	0	0	1	1	1	1	381
12	Genius	Music	Desktop	>50M	5	5	6	6	0	0	0	0	11	11	63
			Compact	>100M	5	5	6	6	0	0	0	0	11	11	62
13	Google	Online services	Desktop	>100B	1	0	0	0	0	0	0	0	1	0	1
14	IRCTC	Airlines	Desktop	>10M	4	4	3	3	0	0	11	11	18	18	34
			Compact	>50M	3	3	3	3	0	0	2	2	8	8	31
15	Makemytrip	Airlines	Desktop	>10M	25	25	7	6	0	0	0	0	32	31	8
16	Microsoft	Information Technology	Desktop	>1B	0	0	0	0	0	0	0	0	0	0	21
			Compact	>500M	0	0	0	0	0	0	0	0	0	0	21
17	NIH	Healthcare	Desktop	>100M	10	10	10	10	0	0	0	0	20	20	0
			Compact	>100M	5	5	5	5	0	0	0	0	10	10	1
18	OpenAI	Computer Software & Development	Desktop	>100M	0	0	1	0	0	0	0	0	1	0	2
			Compact	>500M	0	0	1	0	0	0	0	0	1	0	85
19	Progressive	Insurance	Desktop	>10M	0	0	0	0	0	0	0	0	0	0	0
			Compact	>10M	0	0	0	0	0	0	0	0	0	0	0
20	Samsung	Telecom	Desktop	>10M	0	0	1	0	0	0	0	0	1	0	274
			Compact	>100M	0	0	0	0	0	0	0	0	0	0	67
21	Sciencedirect	Science	Desktop	>50M	0	0	1	1	0	0	0	0	1	1	0
			Compact	>10M	0	0	1	1	0	0	0	0	1	1	0
22	Stackoverflow	Distance Learning	Desktop	>100M	0	0	2	2	0	0	0	0	2	2	65
			Compact	>10M	0	0	4	4	0	0	0	0	4	4	64
23	USPS	Transportation and Logistics	Desktop	>50M	4	4	4	4	0	0	0	0	8	8	39
			Compact	>100M	4	4	4	4	0	0	0	0	8	8	11
24	Wikipedia	Newspapers	Desktop	>1M	0	0	0	0	0	0	0	0	0	0	7
25	Youtube	Newspapers	Desktop	>10B	0	0	0	0	0	0	0	0	0	0	31
26	Zerodha	Investment	Desktop	>10M	1	1	1	1	0	0	0	0	2	2	49
			Compact	>10M	1	1	1	1	0	0	0	0	2	2	49
Total Issues:					90	85	93	86	9	7	15	15	207	193	1767
Precision					94.44%		92.47%		77.78%		100.00%		93.24%		

tab navigation or quick key access. For example, in Figure 4(c), the control for setting DuckDuckGo as the default search option is inaccessible to screen reader users because the `href` attribute is missing. We observed this problem on websites such as NIH, Agoda, and MakeMyTrip, where critical interactive actions were silently rendered inaccessible.

We also observed misuse of ARIA roles—specifically, the inappropriate use of `role="dialog"` on elements that are not meant to behave like dialog boxes. This role is intended for modal or pop-up dialogs that require user interaction and typically trap keyboard focus. On the CapitalOne website (Figure 4 (b)), however, the `<header>` element was incorrectly assigned `role="dialog"` along with `aria-modal="false"`. Although focus trapping was not intended, NVDA treated it as a modal and created a separate virtual buffer, making the header content unreachable via the Down Arrow or Quick Navigation Keys. A similar issue was found on ADP, showing how even well-meaning use of ARIA roles can disrupt navigation. This can be fixed by removing the `role="dialog"` attribute.

Lastly, we identified role conflicts in composite widgets, where conflicting role assignments broke expected semantics. For example, in Figure 4(h), the website Fragrantica assigned conflicting roles that caused navigation menu items such as News, Perfumes, and Notes to be invisible to screen reader users, leaving these functions inaccessible. We also observed cases on Chase, Fragrantica, and StackOverflow where `<a>` elements with role=”menuitem” were nested inside `<li>` elements with role=”none”. This mismatch disrupted both list structure and link semantics, causing NVDA to skip the elements during list and link traversal. Such subtle misconfigurations rendered critical content completely inaccessible to screen reader users.

b) Unactionable elements: A11yNavigator also revealed cases where actionability broke down—elements that failed to respond to expected inputs like Enter or Space. One such case was found on Fragrantica in Figure 4(d), where a visible button labeled “Privacy Manager” was defined using a semantic `<button>` element. Although this button was correctly marked up and reachable through tab and quick key navigation, it lacked any associated JavaScript handler or functionality. As a result, pressing Enter or Space produced no observable behavior. For screen reader users, this creates a misleading experience—NVDA announces the element as a button, signaling expected interactivity, but the control is ultimately inert.

Another pattern occurred on the IRCTC website, where dropdown menus were implemented with correct ARIA markup, including attributes such as `aria-haspopup="listbox"`, `role="listbox"`, and `aria-expanded="false"`. While these attributes suggest full accessibility support, the component failed to register the necessary JavaScript event listeners to toggle the dropdown in response to keyboard actions like Enter or Space. As a result, keyboard-only users relying on screen readers could reach the dropdowns but could not open or use them as shown in Figure 4(g). Though locatable, the elements could not be activated via screen reader, creating a false sense of support and hindering interaction.

c) Broader Impact of Issues: The issues flagged by A11yNavigator not only make navigation difficult for screen reader users—leading to frustration and exclusion—but also reduce overall usability and negatively impact SEO, since search engines rely on semantic structures in much the same way as assistive technologies. Ensuring that accessibility features are locatable and reachable therefore has direct implications for searchability. As highlighted by Moreno and Martínez [52] and Lieke [53], there is a strong overlap between accessibility and search engine optimization (SEO), particularly in how both depend on the underlying structure and semantics of a website. Screen readers and search engine crawlers both navigate and interpret content non-visually through code, links, headings, and metadata, meaning that elements inaccessible to assistive technologies—such as those missing alt-text or labels—are often equally invisible to crawlers. Thus, improving accessibility by making elements navigable, properly labeled, and semantically structured benefits users with disabilities while enhancing search engine visibility; conversely, neglecting these practices limits both user access and indexing.

F. RQ4: Performance

To evaluate the practicality of integrating A11yNavigator into a typical development workflow, we measured and analyzed its execution time across all 26 websites in our dataset, focusing separately on locatability and actionability issues. For locatability issues, A11yNavigator required an average of 43 minutes (2,580 seconds) using TAB key navigation, 39 minutes (2,340 seconds) using quick key navigation, and 92 minutes (5,520 seconds) using down arrow key navigation. For identifying actionability issues, the average execution time per website was approximately 60 minutes (3,600 seconds). Thus, on average, A11yNavigator can comprehensively analyze a single website, covering all three navigation modes along with actionability analysis, within approximately 234 minutes. Each detection strategy runs independently, which means they can be executed in parallel. While the current version of A11yNavigator runs them sequentially, multi-threading and multi-processing can be implemented to significantly reduce the running time. We expect its performance to improve even further if it is deployed on a server with many processors. In addition, since A11yNavigator runs automatically without requiring developer intervention, it can be integrated into a standard development pipeline, such as nightly builds or continuous integration workflows.

V. Threats to Validity

We acknowledge several potential threats to the validity of our evaluation and describe the steps taken to mitigate them:

External Validity: To reduce selection bias, we chose websites used in prior accessibility study [26], covering popular sites across diverse domains. We included both desktop and responsive views to capture a wide range of layout structures and improve the generalizability of our findings.

In addition, unlike precision, which can be verified directly against tool-reported results, recall requires establishing the complete set of accessibility issues on a page through exhaustive manual verification. This process is highly time-consuming—for instance, validating a single site across three navigation modes and two issue types can take more than three hours and requires extensive cross-checking. Prior work has also noted these challenges [54]. To address this, we focused our recall evaluation on six representative websites, selecting two from each size category (large, medium, small).

Internal Validity: We manually reviewed the detected issues to assess precision and recall, which may introduce subjectivity. To mitigate this, three authors and an external reviewer independently verified and discussed the results to strengthen the reliability of our evaluation.

A11yNavigator relies on tools like Selenium WebDriver, JavaScript-based extractors, and NVDA's logging interface. These may introduce errors during interaction or DOM extraction, such as timing issues or incorrect XPath resolution. Our prototype may also contain bugs. To reduce these risks, we used the latest stable library versions and added retry logic for unstable cases to ensure reliability.

VI. Related Work

The Web Content Accessibility Guidelines (WCAG) [8] have served as the foundation for the development of most existing accessibility checkers [23]–[25], [55]–[61]. However, due to their rule-based nature and reliance on static analysis, these tools mainly flag straightforward issues such as missing `alt` text or insufficient color contrast. They often struggle to detect more complex accessibility problems that arise from dynamic content or user interactions.

To address these limitations, several studies have investigated the detection of accessibility violations through dynamic analysis [54], [62], [63]. These efforts include identifying issues such as excessive reaching time to access a web element or overly long navigation paths. Other examples involve detecting invalid or empty ARIA labels assigned to visible elements [64], [65], keyboard navigation problems such as traps or dialog-based navigation failures, and reflow issues on web pages [66]–[69].

Researchers have found that certain accessibility violations can only be detected through the use of assistive technologies, which might otherwise be overlooked [70]. As a result, several studies have incorporated assistive technologies such as screen readers during evaluations to uncover these issues [70]–[75]. However, most of these approaches still rely on manual effort, such as interacting with the application under test or the provision of GUI test cases.

While Latte [70] leverages the TalkBack screen reader to detect accessibility issues in Android apps through existing scripted GUI tests, our work differs in several important ways. We focus on the web domain, where GUI test suites are rarely available. A recent study found that over 95% of web applications on GitHub lack GUI test suites [76], and prior research shows that even when present they are often slow, fragile, and resource-intensive, leading to frequent abandonment, continued reliance on manual testing, and high breakage rates in production [77], [78]. To address these limitations, our approach dynamically explores websites in real time and evaluates accessibility using NVDA screen reader interactions, eliminating the dependency on pre-existing GUI tests. Additionally, the interaction model in our work differs significantly from Latte: we rely on keyboard-based navigation strategies for the web, whereas Latte uses swipe-based gestures for mobile applications, resulting in a different implementation and detection of different issue types.

An exception is Groundhog [72], an automated system that uses the TalkBack screen reader to crawl and interact with Android app interfaces, identifying issues related to element locatability and actionability. However, to the best of our knowledge, all existing work that incorporates assistive technologies has been limited to mobile platforms. Our work differs from Groundhog in several key ways, as the web environment presents unique challenges. First, websites are inherently more complex than native mobile apps, often containing a larger number and greater variety of interface elements. Additionally, interacting with websites using a screen reader introduces more barriers compared to native mobile apps [79]. Second, screen readers used on the web offer more diverse navigation modes than those available on mobile devices. While mobile navigation is typically linear and gesture-based, web environments support multiple strategies, including tabbing, arrow-key traversal, and semantic shortcuts for quick navigation.

Our work, A11yNavigator, differs from prior studies by integrating assistive technology directly into the detection process and focusing on a distinct environment—the web. A11yNavigator simulates diverse navigation strategies using the NVDA screen reader, enabling the discovery of a wider and more nuanced set of locatability and actionability issues.

VII. Conclusion

In this paper, we presented A11yNavigator, a fully automated accessibility testing tool designed to detect issues that arise during screen reader navigation of web pages. Unlike static tools like WAVE, which focus on rule-based compliance, A11yNavigator uncovers elements on a web page that are either unreachable (locatability issues) or unresponsive to interaction (actionability issues). Evaluated across 26 popular websites, A11yNavigator achieved a precision of 93.24% and recall of 98.97%.

Interesting avenues of future work include (i) automatically categorizing the detected issues and generate suggestions for fixes, (ii) extending the A11YNavigator approach to build a large-scale benchmark dataset of recurring accessibility issues, and (iii) conducting qualitative studies with web developers to explore why non-semantic patterns persist, and how better guidance or tooling might help.

The artifacts for this research are publicly available [27].

Acknowledgment

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

References

[1] Y. Nagano, K. Suginome, K. Yoshimoto, and Y. Tsuchiya, "Activities for improving web accessibility," *Fujitsu Scientific and Technical Journal*, vol. 45, no. 2, pp. 239–246, 2009.

[2] I. Basdekis, I. Klironomos, I. Metaxas, and C. Stephanidis, "An overview of web accessibility in greece: a comparative study 2004–2008," *Universal Access in the Information Society*, vol. 9, pp. 185–190, 2010.

[3] World Health Organization, "Disability and health," https://www.who.int/news-room/fact-sheets/detail/disability-and-health, Geneva, Switzerland, 2024, accessed: May 10, 2025. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/disability-and-health

[4] R. R. A. Bourne, J. D. Steinmetz, S. Flaxman, P. S. Briant, H. R. Taylor, S. Resnikoff, and V. L. E. G. of the Global Burden of Disease Study, "Trends in prevalence of blindness and distance and near vision impairment over 30 years: An analysis for the global burden of disease study," *The Lancet Global Health*, vol. 9, no. 2, pp. e130–e143, 2021. [Online]. Available: https://doi.org/10.1016/S2214-109X(20)30425-3

[5] B the Change, "How many people with disabilities use my website?" https://bthechange.com/how-many-people-with-disabilities-use-my-website-2e737caf55bf, 2020, accessed: May 10, 2025. [Online]. Available: https://bthechange.com/how-many-people-with-disabilities-use-my-website-2e737caf55bf

[6] WebAIM, "Screen reader user survey #10 results," https://webaim.org/projects/screenreadersurvey10/, 2021, accessed: Nov. 5, 2024.

[7] W3C, "Wai-aria authoring practices 1.2," https://www.w3.org/WAI/ARIA/apg/, 2021, accessed: 2025-04-30.

[8] ——, "Web content accessibility guidelines (wcag) 2.2," https://www.w3.org/TR/WCAG22/, 2024.

[9] Apple, "Accessibility on ios," https://developer.apple.com/accessibility/ios/, 2022, accessed: May 6, 2021. [Online]. Available: https://developer.apple.com/accessibility/ios/

[10] Android Developers, "Build more accessible apps," https://developer.android.com/guide/topics/ui/accessibility, 2022, accessed: May 6, 2022. [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility

[11] European Union, "Directive (eu) 2019/882 of the european parliament and of the council on the accessibility requirements for products and services," https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32019L0882, 2019, accessed: May 10, 2025. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32019L0882

[12] J. Frank, "Web accessibility for the blind: Corporate social responsibility or litigation avoidance?" in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, 2008, pp. 284–284.

[13] S. F. Huq, M. Tafreshipour, K. Kalcevich, and S. Malek, "Automated generation of accessibility test reports from recorded user transcripts," in *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE)*. Ottawa, Canada: IEEE/ACM, 2025, distinguished Paper Award.

[14] A. Aizpurua, M. Arrue, S. Harper, and M. Vigo, "Are users the gold standard for accessibility evaluation?" in *Proceedings of the 11th Web for All Conference*, 2014, pp. 1–4.

[15] D. A. Mateus, C. A. Silva, A. F. De Oliveira, H. Costa, and A. P. Freire, "A systematic mapping of accessibility problems encountered on websites and mobile apps: A comparison between automated tests, manual inspections and user evaluations," *Journal on Interactive Systems*, vol. 12, no. 1, pp. 145–171, 2021.

[16] S. F. Huq, A. Alshayban, Z. He, and S. Malek, "#a11ydev: Understanding contemporary software accessibility practices from twitter conversations," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Hamburg, Germany: Association for Computing Machinery, 2023, pp. 1–18. [Online]. Available: https://doi.org/10.1145/3544548.3581455

[17] N. Salehnamadi, Z. He, and S. Malek, "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*, ser. CHI '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3544548.3580679

[18] A. Alshayban, I. Ahmed, and S. Malek, "Accessibility issues in android apps: State of affairs, sentiments, and ways forward," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*. New York, NY, USA: ACM/IEEE, 2020, pp. 1323–1334. [Online]. Available: https://doi.org/10.1145/3377811.3380364

[19] Y. Inal, F. Guribye, D. Rajanen, M. Rajanen, and M. Rost, "Perspectives and practices of digital accessibility: A survey of user experience professionals in nordic countries," in *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*. ACM, 2020, pp. 1–11.

[20] T. Bi, X. Xia, D. Lo, J. Grundy, T. Zimmermann, and D. Ford, "Accessibility in software practice: A practitioner's perspective," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 4, pp. 1–26, 2022.

[21] C. Vendome, D. Solano, and M. Linares-Vásquez, "Can everyone use my app? an empirical study on accessibility in android apps," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 424–434.

[22] M. V. R. Leite, L. P. Scatalon, A. P. Freire, and M. M. Eler, "Accessibility in the mobile development industry in brazil: Awareness, knowledge, adoption, motivations and barriers," *Journal of Systems and Software*, vol. 177, p. 110942, Jul. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412122100039X

[23] WebAIM, "Wave: Web accessibility evaluation tool," https://wave.webaim.org/, 2023, accessed: May 10, 2025. [Online]. Available: https://wave.webaim.org/

[24] Microsoft Developer Documentation, "Accessibility testing with lighthouse in microsoft edge devtools," https://learn.microsoft.com/en-us/microsoft-edge/devtools-guide-chromium/accessibility/lighthouse, 2024, accessed: May 10, 2025. [Online]. Available: https://learn.microsoft.com/en-us/microsoft-edge/devtools-guide-chromium/accessibility/lighthouse

[25] Deque Systems, "axe-core: Accessibility engine for automated web ui testing," https://github.com/dequelabs/axe-core, 2023, accessed: May 10, 2025. [Online]. Available: https://github.com/dequelabs/axe-core

[26] M. Tafreshipour, A. Deshpande, F. Mehralian, I. Ahmed, and S. Malek, "Ma11y: A mutation framework for web accessibility testing," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 100–111. [Online]. Available: https://doi.org/10.1145/3650212.3652113

[27] "A11ynavigator: Artifact submission (anonymous repository)," https://anonymous.4open.science/r/A11yNavigator-942F/, 2025, artifact under review.

[28] WHATWG, "Introduction to the dom — dom living standard," https://dom.spec.whatwg.org/#introduction-to-the-dom, 2024, accessed: May 10, 2025. [Online]. Available: https://dom.spec.whatwg.org/#introduction-to-the-dom

[29] E. Castro, *HTML for the world wide web*. Peachpit Press, 2003.

[30] Mozilla Developer Network, "Xpath," https://developer.mozilla.org/en-US/docs/Web/XML/XPath, 2024, accessed: May 10, 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/XML/XPath

[31] Microsoft, "Microsoft active accessibility (msaa)," https://learn.microsoft.com/en-us/windows/win32/winauto/microsoft-active-accessibility, 2024, accessed: May 10, 2025. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/winauto/microsoft-active-accessibility

[32] Linux Foundation, "Iaccessible2 specification documentation," https://accessibility.linuxfoundation.org/a11yspecs/ia2/docs/html/index.html, 2024, accessed: May 10, 2025. [Online]. Available: https://accessibility.linuxfoundation.org/a11yspecs/ia2/docs/html/index.html

[33] Microsoft, "Ui automation overview (.net framework)," https://learn.microsoft.com/en-us/dotnet/framework/ui-automation/ui-automation-overview, 2024, accessed: May 10, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/framework/ui-automation/ui-automation-overview

[34] Program-L, "Program-l: Vi programmers discussion list," https://www.freelists.org/list/program-l, 2025, discussion group for visually impaired computer programmers.

[35] M. Van Someren, Y. F. Barnard, and J. Sandberg, *The Think Aloud Method: A Practical Approach to Modelling Cognitive Processes*. London: Academic Press, 1994.

[36] "Agoda official site — free cancellation & booking deals — over 2 million hotels," https://www.agoda.com/, 2025, accessed: August 10, 2025.

[37] Stack Overflow, "Stack overflow: Questions page," https://stackoverflow.com/questions, 2024, accessed: May 10, 2025. [Online]. Available: https://stackoverflow.com/questions

[38] StackvOverflow, "Stack overflow developer survey 2022," https://survey.stackoverflow.co/2022/, 2022, accessed: May 30, 2025.

[39] MakeMyTrip, "Makemytrip - online flight, hotel, and travel booking," https://www.makemytrip.global, 2025, accessed: May 14, 2025.

[40] Indian Railway Catering and Tourism Corporation (IRCTC), "Irctc train search page," https://www.irctc.co.in/nget/train-search, 2024, accessed: May 10, 2025.

[41] MDN, "Eventtarget," https://developer.mozilla.org/en-US/docs/Web/API/EventTarget, MDN, 2025.

[42] WAI, "Providing accessible names and descriptions," https://www.w3.org/WAI/ARIA/apg/practices/names-and-descriptions/, W3C, 2025.

[43] World Wide Web Consortium (W3C), "Wai-aria authoring practices guide: Design patterns and widgets," https://www.w3.org/WAI/ARIA/apg/patterns/, 2024, accessed: May 10, 2025.

[44] NV Access, "Nvda screen reader user guide," https://www.nvaccess.org/files/nvda/documentation/userGuide.html, 2024, accessed: May 10, 2025. [Online]. Available: https://www.nvaccess.org/files/nvda/documentation/userGuide.html

[45] Y. Borodin, J. P. Bigham, G. Dausch, and I. V. Ramakrishnan, "More than meets the eye: a survey of screen-reader browsing strategies," in *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, ser. W4A '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1805986.1806005

[46] Selenium Project, "Webdriver — selenium documentation," https://www.selenium.dev/documentation/webdriver/, 2024, accessed: May 10, 2025. [Online]. Available: https://www.selenium.dev/documentation/webdriver/

[47] PyPI Contributors, "keyboard: Hook and simulate global keyboard events on windows and linux," https://pypi.org/project/keyboard/, 2024, accessed: May 10, 2025. [Online]. Available: https://pypi.org/project/keyboard/

[48] W3C Web Accessibility Initiative (WAI), "Understanding wcag 2.2," https://www.w3.org/WAI/WCAG22/Understanding/, 2023, accessed: May 10, 2025. [Online]. Available: https://www.w3.org/WAI/WCAG22/Understanding/

[49] Semrush Company, "Semrush: Online marketing can be easy," https://www.semrush.com/, 2023, accessed: May 10, 2025. [Online]. Available: https://www.semrush.com/

[50] T. Nguyen, "Evaluating automated accessibility checker tools," 2024.

[51] J. Ara, C. Sik-Lanyi, and A. Kelemen, "Accessibility engineering in web evaluation process: a systematic literature review," *Universal Access in the Information Society*, vol. 23, no. 2, pp. 653–686, 2024.

[52] L. Moreno and P. Martinez, "Overlapping factors in search engine optimization and web accessibility," *Online information review*, vol. 37, no. 4, pp. 564–580, 2013.

[53] K. Lieke, "Accessibility, usability and seo," 2020.

[54] F. Durgam, J. Grigera, and A. Garrido, "Dynamic detection of accessibility smells," *Universal Access in the Information Society*, pp. 1–12, 2023.

[55] G. Broccia, M. Manca, F. Paternò, and F. Pulina, "Flexible automatic support for web accessibility validation," *Proceedings of the ACM on Human-Computer Interaction*, vol. 4, no. EICS, pp. 1–24, 2020.

[56] G. Gay and C. Q. Li, "Achecker: open, interactive, customizable, web accessibility checking," in *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*. Raleigh, USA: Association for Computing Machinery, 2010, pp. 1–2.

[57] A. M. Agency, "Access monitor plus," https://accessmonitor.acessibilidade.gov.pt/, Administrative Modernization Agency, 2021.

[58] C. Benavidez, "Examinator," http://examinator.net/, 2015.

[59] IBM, "Verify - automated – ibm accessibility," https://www.ibm.com/able/toolkit/verify/automated, IBM, 2024.

[60] A11yWatch, "A11ywatch: Web accessibility evaluation tool," https://a11ywatch.com/, A11yWatch, 2024.

[61] QualWeb, "Qualweb," https://qualweb.di.fc.ul.pt/evaluator/, QualWeb, 2024.

[62] H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda, "Accessibility designer: visualizing usability for the blind," *ACM SIGACCESS accessibility and computing*, no. 77-78, pp. 177–184, 2003.

[63] T. Bostic, J. Stanley, J. Higgins, D. Chudnov, J. Brunelle, and B. Tracy, "Automated evaluation of web site accessibility using a dynamic accessibility measurement crawler," *arXiv preprint arXiv:2110.14097*, 2021.

[64] C. Duarte, A. Salvado, M. E. Akpinar, Y. Yeşilada, and L. Carriço, "Automatic role detection of visual elements of web pages for automatic accessibility evaluation," ser. W4A '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3192714.3196827

[65] M. Bajammal and A. Mesbah, "Semantic web accessibility testing via hierarchical visual analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1610–1621.

[66] P. T. Chiou, A. S. Alotaibi, and W. G. J. Halfond, "Detecting and localizing keyboard accessibility failures in web applications," ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 855–867. [Online]. Available: https://doi.org/10.1145/3468264.3468581

[67] P. T. Chiou, A. S. Alotaibi, and W. G. Halfond, "Bagel: An approach to automatically detect navigation-based web accessibility barriers for keyboard users," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–17.

[68] ——, "Detecting dialog-related keyboard navigation failures in web applications," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1368–1380.

[69] P. T. Chiou, R. Winn, A. S. Alotaibi, and W. G. Halfond, "Automatically detecting reflow accessibility issues in responsive web pages," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[70] N. Salehnamadi, A. Alshayban, J.-W. Lin, I. Ahmed, S. Branham, and S. Malek, "Latte: Use-case and assistive-service driven automated accessibility testing framework for android," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–11.

[71] M. Taeb, A. Swearngin, E. School, R. Cheng, Y. Jiang, and J. Nichols, "Axnav: Replaying accessibility tests from natural language," *arXiv preprint arXiv:2310.02424*, 2023.

[72] N. Salehnamadi, F. Mehralian, and S. Malek, "Groundhog: An automated accessibility crawler for mobile apps. in 2022 37th ieee," in *ACM International Conference on Automated Software Engineering. IEEE, ACM New York, NY, USA, Rochester, Michigan, USA*, 2022.

[73] N. Salehnamadi, Z. He, and S. Malek, "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*, 2023, pp. 1–20.

[74] A. S. Alotaibi, P. T. Chiou, and W. G. Halfond, "Automated detection of talkback interactive accessibility failures in android applications," in *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2022, pp. 232–243.

[75] A. Alshayban and S. Malek, "Accessitext: automated detection of text accessibility issues in android apps," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 984–995.

[76] S. DI MEGLIO, L. L. L. Starace, V. Pontillo, R. Opdebeeck, C. De Roover, and S. Di Martino, "Investigating the adoption and maintenance of web gui testing: Insights from github repositories," *Available at SSRN 5182165*, 2025.

[77] L. Christophe, R. Stevens, C. De Roover, and W. De Meuter, "Prevalence and maintenance of automated functional tests for web applications," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 141–150.

[78] R. Coppola, M. Morisio, and M. Torchiano, "Scripted gui testing of android apps: A study on diffusion, evolution and fragility," in *Proceedings of the 13th international conference on predictive models and data analytics in software engineering*, 2017, pp. 22–32.

[79] I. Xie, T. H. Lee, H. S. Lee, S. Wang, and R. Babu, "Comparison of accessibility and usability of digital libraries in mobile platforms: Blind and visually impaired users' assessment," *Information Research an international electronic journal*, vol. 28, no. 3, pp. 59–82, 2023.