Visual Accessibility Auditing Across Interaction States and Reading Orders

Md Rahat-uz- Zaman, Jake Wagoner, Tingying He, Alexander Lex, Paul Rosen

Abstract

Automated accessibility checkers evaluate a single snapshot of a page, but many barriers only emerge over multiple interactions rather than a static snapshot. Several WCAG 2.2 criteria depend on geometry, focus state, or interaction history, such as a button that becomes obscured once a sticky header appears. Others arise when the visual layout, the screen-reader order, and the keyboard (`Tab`) order of a page disagree, so different users encounter the same content in conflicting sequences. Auditing these requires reasoning about geometry and structure alongside the live page, which makes it both a detection and a visualization problem. We present *Site Accessibility Auditor*, a Chrome DevTools extension with two coordinated lenses. The *interaction* lens automatically drives the page through its reachable states and overlays interaction-dependent issues as inspectable visual evidence on a full-page capture. The *reading & focus order* lens aligns the visual, screen-reader, and keyboard orders, shows where they diverge, and suppresses intentional patterns such as skip links. We actively evaluate the extension as site owners and as an auditor. In a design partnership with the Utah Wastewater Surveillance System dashboard, our tool motivated the team to ship accessibility fixes. We used the tool on the Our World in Data site, which shows that the approach runs on an unmodified in-the-wild example.

Keywords: accessibility, chart description, browser tooling

1 Introduction

Web accessibility has become a standard consideration in modern web development. Recent U.S. regulations on web content provided by state and local governments increasingly state web accessibility as a legal obligation [27]. Yet it remains difficult to achieve in practice. Automated auditing tools such as Lighthouse [7, 8], axe-core [11], and WAVE [35] have become indispensable for identifying common accessibility violations including missing labels, invalid ARIA, and insufficient color contrast. Despite their widespread adoption, accessibility defects remain pervasive: the 2026 WebAIM Million report found an average of 56 WCAG 2 failures on the home pages of the top one million websites, a figure that has remained fairly steady over the past five years [36].

W3C guidance explicitly recognizes that no single tool can determine whether a webpage is accessible [31], and empirical studies have shown that relying solely on automated testing leaves substantial accessibility barriers undiscovered [29]. Consequently, developers still spend considerable effort manually inspecting interaction behavior, keyboard navigation, and screen-reader accessibility after automated checks have finished.

A key limitation of existing accessibility tools is that they primarily evaluate a single snapshot of the Document Object Model (DOM). Many important accessibility barriers, however, cannot be determined from a static page state. Instead, they emerge only through user interaction or by comparing multiple representations of the same interface. We focus on two increasingly common classes of accessibility problems that share this defining characteristic.

The first class comprises *interaction-dependent accessibility barriers*. Several WCAG 2.2 success criteria—including target size, focus visibility and obstruction, dragging alternatives, redundant entry, accessible authentication, and consistent help—depend on geometry, interaction state, or previously observed user actions rather than markup alone [30, 34]. For example, a control that satisfies the target-size requirement when the page initially loads may become partially obscured after a sticky header appears, while determining whether information is entered redundantly requires comparing multiple interaction states. Evaluating these criteria, therefore, requires driving the interface and observing how accessibility changes over time rather than inspecting only the initial DOM.

The second class comprises *reading-order divergence*. Modern webpages expose multiple reading orders: the visual order implied by layout, the accessibility tree (AX) order consumed by screen readers, and the keyboard traversal defined by `Tab`. When these representations disagree, for example, when keyboard focus reaches a control before its instructions, a chart before its legend, or when visually prominent elements are absent from the accessibility tree, different users experience fundamentally different versions of the same page [30]. These problems are particularly common in visualization-rich dashboards [3, 21], where maps, charts, filters, tables, and custom interactive widgets create multiple overlapping navigation structures that are already known to challenge screen-reader users [14, 24, 38].

We argue that both classes of problems are visualization problems as much as they are detection problems. Auditing them requires developers to reason simultaneously about spatial layout, interaction state, accessibility semantics, and navigation order. Rather than presenting another textual checklist of WCAG violations, effective auditing should visually connect this evidence directly to the live webpage so that developers can understand not only what failed, but also why it failed and where it should be repaired. Because developers already inspect layout, styling, and behavior within browser developer tools, an in-browser environment provides a natural setting for accessibility inspection grounded in the live DOM.

To support this workflow, we present *Site Accessibility Auditor*, a Chrome DevTools extension organized around two complementary inspection lenses:

- An **interaction lens** that automatically explores reachable interface states, evaluates interaction-dependent WCAG 2.2 criteria, and presents findings as inspectable geometric evidence directly over the live webpage.
- A **reading-&-focus-order lens** that aligns visual, accessibility-tree, and keyboard traversals, visualizes where these representations diverge, classifies common divergence patterns, and suppresses intentional navigation behaviors such as skip links, roving-tabindex widgets, and modal focus traps.

We make the extension open-source; the source code can be found in github.com/rahatzamancse/site-accessibility-auditor. The Chrome extension can be found in l.insane.casa/accessibility-auditor. We demonstrate the utility of these lenses through a design partnership with the Utah Wastewater Surveillance System dashboard, where the tool motivated accessibility improvements that were subsequently deployed, and through an in-the-wild audit of Our World in Data [20], showing that the approach scales to any website.

2 Background and Related Work

Accessibility evaluation in repair workflows. WCAG provides the normative vocabulary for evaluating web accessibility, and tools such as Lighthouse, axe-core, WAVE, and W3C evaluation resources have made automated accessibility checks a routine part of development and review [7, 8, 11, 30, 32–35]. However, prior work consistently

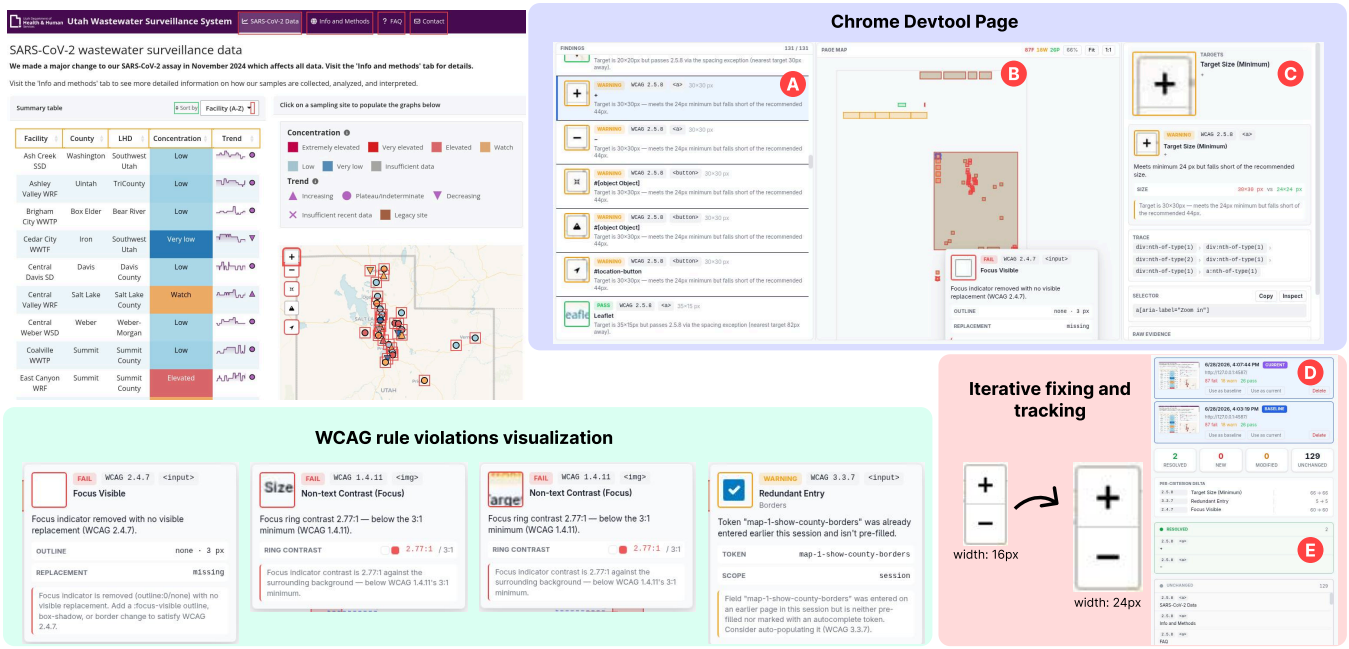

Figure 1: The *interaction* lens on a dashboard with 131 audited targets. The website has colored bounding boxes on all targets based on pass, warning and failed WCAG rules. All these elements are navigable as a list (A), and overview of the website with the same colors are shown in the middle panel (B), and selected element details, their trace in the page and suggestion on how to fix is show on the right panel (C). We cover most WCAG accesssibility rules, along with identified common reasons for failing, and suggestions on how to fix them (four example rules shown in the green area. The extension tool allows running auditing across multiple sessions and compare inbetween. We increased a button size to meet accessibility requirements and compared the new and previous versions in (D), showing that the issue is resolved in (E).

shows that accessibility evaluation cannot be reduced to automated rule checking. Mankoff et al. compared automated tools, expert review, and user testing, showing that different methods expose different barriers and that developer-facing evaluations must support more than syntactic checking [16]. Vigo et al. found that sole reliance on automated tests leaves substantial WCAG coverage unexamined and can introduce false positives [29]. Brajnik et al. further showed that even expert conformance judgments can vary, emphasizing the interpretive nature of many accessibility criteria [5]. Studies with blind users likewise show that guideline conformance captures only part of the accessibility problems users encounter in practice [17], and domain-specific instruments, such as usability checklists for public health dashboards, similarly depend on structured human inspection [1]. Our work builds on this view of accessibility evaluation as a mixed automated-and-human process: automation gathers evidence at scale, while visual inspection helps developers understand where barriers arise and how to repair them in context.

Dynamic and interaction-aware accessibility testing. Modern web interfaces challenge static accessibility evaluation because many barriers appear only after interaction. Menus open, sticky regions obscure controls, form state accumulates, and custom widgets expose different content to the DOM, the accessibility tree, and the keyboard focus sequence over time. Several systems have explored dynamic accessibility evaluation. Schiavone and Paternò's MAUVE environment supports guideline-based evaluation of dynamic web pages through browser-based validation [22]. Demodocus models a web application as a graph of reachable states and compares how users with and without disabilities can traverse that graph [4]. More recently, QualState explores states in single-page applications before passing those states to an automated evaluator [19]. Keyboard-specific systems such as BAGEL automatically detect navigation-based barriers that hinder keyboard users [6]. These systems motivate dynamic exploration as a necessary complement to static audits. Site Accessibility Auditor differs in emphasis: rather than primarily reporting coverage over a state graph, it presents interaction-dependent WCAG 2.2 findings as page-grounded visual evidence, including target geometry, focus indicators, obstruction regions, keyboard paths, and cross-state signatures.

Visualizing accessibility structure and reading order. Accessibility barriers are often produced by mismatches among multiple representations of the same page: rendered layout, DOM order, accessibility-tree order, and keyboard focus order. WCAG criteria such as meaningful sequence, focus order, focus visible, and focus not obscured make these relationships central to accessibility [30,34]. Prior tools have long recognized that accessibility feedback must be made visible to authors. For example, aDesigner visualized blind-user usability by overlaying accessibility problems on web pages, helping authors see issues that are otherwise hidden from sighted inspection [26]. Browser developer tools and accessibility inspectors expose parts of this structure, such as DOM nodes, accessibility properties, or tab order, but they usually present these views separately. Our reading-&-focus-order lens extends this inspection-oriented tradition by aligning visual, accessibility-tree, and keyboard traversals in one representation. It treats order divergence not simply as a rule violation, but as a comparison problem: authors need to see which elements move, disappear, or change meaning across modalities, and whether those differences reflect intentional patterns such as skip links and modal focus traps or barriers that strand users.

3 Design Goals

Three goals, distilled from the accessibility and visualization literature and from our partnership with the Utah Wastewater Surveillance System (UWSS) dashboard team [28] shape the tool. They position it as an inspection aid that makes accessibility evidence visible on the live page, where repair decisions are actually made, rather than as a replacement for expert review.

G1: Keep the audit in the repair context. Developers already debug layout, data, and interaction in the browser, over the live page. Accessibility review should appear in the same environment and at the same spatial specificity, letting an author move fluidly between an issue, the element it concerns, and the surrounding visual context.

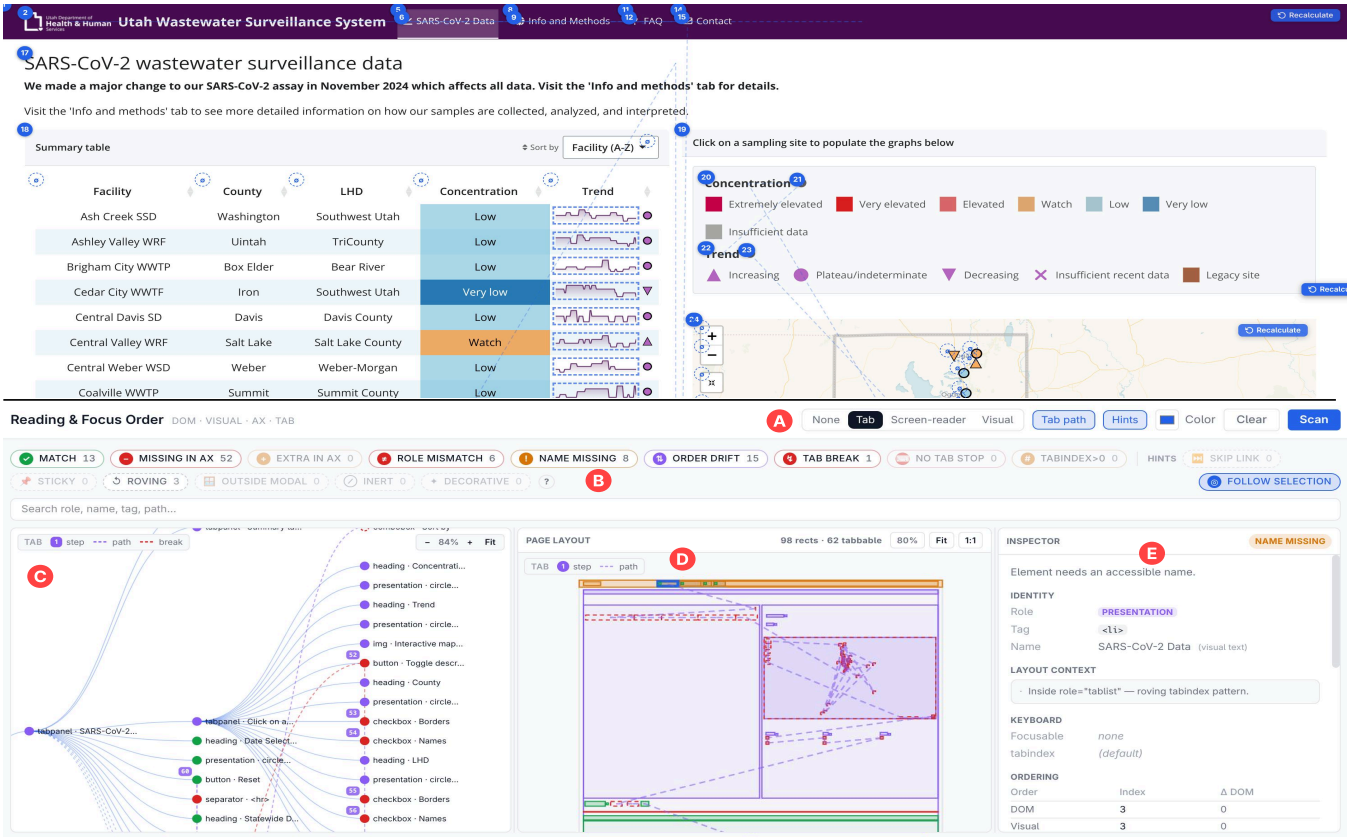

Figure 2: The Reading & Focus Order lens auditing the live UWSS dashboard, whose keyboard order is overlaid as numbered badges (top). Developers can change this overlay visualization between visual, AX-tree, and keyboard traversal order (A). All orders are analyzed and classified into various divergence categories (e.g., missing-in-AX, element role mismatch, missing name, element order drift) (B). The element containment order of the screen reader is visualized in (C), and each divergence is color-coded based on these categories. Besides overlays on the website, (D) shows a simplified overview of the element order for all order diverging elements, and (E) shows details of the divergences along with explanations.

G2: Make interaction-dependent state inspectable. Many barriers appear only after an action: a menu opens, focus moves, a sticky bar obscures a control, or a field reappears. The tool should audit not just the initial DOM but the reachable states, and present each verdict as concrete evidence (e.g., focus and obstruction rectangles, size comparisons, field and help signatures) rather than a bare label.

G3: Make order divergence legible across modalities. Because visual, AX, and keyboard traversals can disagree, the tool should represent all three together, expose where and how they drift, and distinguish intentional reordering (skip links, roving tabindex, modal traps) from divergences that strand users.

4 Two Inspection Lenses

Site Accessibility Auditor is a Chrome DevTools extension that keeps accessibility review in the browser, where developers already inspect layout, styling, and interaction behavior (achieving G1). The tool organizes findings around two repair-oriented questions: does the page remain operable across reachable interaction states, and does it unfold coherently across visual layout, screen-reader exposure, and keyboard navigation? Both lenses share a common evidence model: each finding is linked to a page state, DOM trace, page-coordinate region, category, and explanation, so highlights, inspectors, and run-to-run comparisons remain consistent across the tool.

4.1 Interaction lens

The interaction lens asks whether the page can be operated comfortably and predictably, not just as the first loaded state, but across the states a user can reach. Because the relevant WCAG 2.2 criteria are decided by geometry, focus, and history rather than markup (G2), a static pass over the initial DOM is insufficient. The lens instead drives the page with browser automation, opening menus, moving focus, and exercising controls, and re-evaluates each reachable state.

In every state, it checks the interaction-dependent criteria. It measures target size and spacing, flagging interactive targets below WCAG 2.5.8 minimum specifications unless they satisfy the spacing exception, for which it reports the nearest-neighbor distance. It simulates keyboard focus to expose missing focus indicators (2.4.7), controls hidden or clipped by fixed and sticky overlays (2.4.11/2.4.12), and focus-ring contrast below the required 3:1 against the walked background (1.4.11). It inspects drag-like widgets and flags those lacking a keyboard, button, slider, or numeric alternative (2.5.7). For criteria that require memory across states, the lens records compact, privacy-preserving signatures: hashed field tokens for redundant entry (3.3.7); paste-blocking, missing `autocomplete`, CAPTCHA, and transcription prompts for accessible authentication (3.3.8); and help affordances compared by type and page region (3.2.6).

Crucially, findings are presented as evidence (Fig. 1). Issues are rendered over a stitched full-page capture using a minimap, so developers see where problems cluster; selecting one opens a thumbnail of the affected element and an inspector with typed "why" chips, size comparison, alongside cross-session DOM traces. Successive runs can be diffed into new, resolved, and persistent findings so developers can confirm fixes during iterative development.

4.2 Reading & focus order lens

The reading & focus order lens asks whether the page unfolds in a coherent sequence across modalities (G3). It reconstructs three parallel traversals and compares them: an *AX order* over named and semantic nodes (what a screen reader encounters), a *visual order*

from a row/column-aware reading of the layout, and the *keyboard order* defined by `Tab`, including `tabindex`s. Each visible element carries layout signals (e.g., modal context, inert subtrees, roving-tabindex groups, skip links, sticky/fixed positioning) explaining why an order looks the way it does.

These parallel orders are exposed through coordinated views (Fig. 2). A numbered overlay shows the keyboard tab order in situ on the page. A drift ribbon then aligns the lanes and draws connecting curves wherever an element changes position between modalities, while a page-space wireframe overlays the same differences on the layout and traces the keyboard traversal as a numbered path, making it immediate when users meet controls before instructions, a chart before its legend, or content before its heading. Selecting any element opens an inspector that reports its role, the source of its accessible name, its layout context, its keyboard focusability, and its order indices with the signed offset from DOM order.

To help developers with fixing various order divergences, a layout-aware classifier downgrades known, deliberate patterns (e.g., skip links, modal focus traps, roving-tabindex widgets, inert subtrees, and decorative nodes) from errors to informative hints. This helps the developers reason about when an ordering divergence is intentional and when it strands a screen-reader or keyboard user.

5 Use Cases and Formative Evaluation

We assess two lenses formatively on two visualization-rich targets: a design partnership with the Utah Wastewater Surveillance System (UWSS) dashboard team [28], and an in-the-wild walkthrough of Our World in Data (OWID) [20]. Our goal is not an exhaustive WCAG coverage or a controlled user study, but to show evidence that the interaction and reading & focus order lenses support practical accessibility reasoning in realistic settings.

5.1 Design partnership: UWSS dashboard

Public health dashboards became critical communication infrastructure during the COVID-19 pandemic [2, 10, 12], and a growing literature examines their design and usability [9, 15, 18, 23, 25] as well as the practices of the teams who build them [13, 37]. Among them, the UWSS dashboard [28] presents SARS-CoV-2 wastewater data through a sortable facility table, color-coded concentration categories, trend glyphs, a geographic map, and supporting legends and controls (Fig. 2). Tabular, geographic, and interactive elements must work together, which makes it a representative test for both lenses. We audited the live site, shipped fixes with the team, and used the tool's run-to-run comparison to confirm them.

Interaction lens. The lens flagged many target-size failures: small info icons, table sort glyphs, and map zoom controls. In mobile view the facility table collapsed to rows too thin to be reliable touch targets, and the Plotly-based charts exposed no touch- or keyboard-operable controls. In response, the team converted the table to a responsive card layout on small screens, enlarged interactive targets, and adopted more uniform target shapes; the diff view confirmed that the corresponding failures cleared across iterations.

Reading & focus order lens. The lens surfaced divergences invisible to a static check. Although the screen-reader (AX) order presented the navigation links as intended, the keyboard order skipped two of them: a *tab break* that made those destinations unreachable for keyboard-only users. More subtly, map markers positioned outside the visible bounds were still picked up by both the AX and keyboard orders, off-screen; we noticed this only because those two orders drifted from the visual order in the drift ribbon. The team repaired the traversal with explicit `tabindex` values, but found that Leaflet.js does not permit reordering its DOM and that Plotly.js does not expose keyboard-operable chart controls—library limits that bounded how far the order could be fixed.

5.2 In-the-wild walkthrough: Our World in Data

We ran the lenses against OWID, a public site with many interactive charts, and found several key issues with the website.

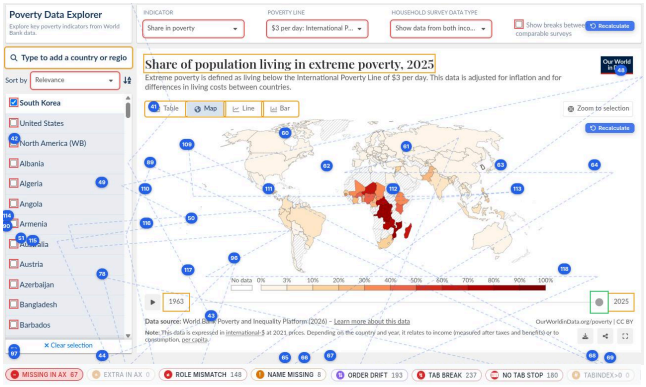

Figure 3: The Reading & Focus Order lens on Our World in Data's Poverty Data Explorer, tab order shown as numbered badges. Dense menus around a choropleth map yield heavy divergence, navigable with the legend buttons, showing how sharply keyboard, AX, and visual orders disagree in this website.

Interaction lens. Dense controls for selecting countries, time ranges, and metrics produce long keyboard paths, and many hover-revealed affordances offer no keyboard or screen-reader equivalent, privileging mouse users on exactly the interactions that drive the charts. Furthermore, even though the website claims mobile responsiveness, we found poor touch size compatibility with most of the visualizations used on the website. Although automatic CSS injection to fix common interaction accessibility issues are out of scope for this paper, we applied external CSS styles on the website for map control buttons, similar to our development workflow with the UWSS developer team, and found those issues to be fixed in the cross-session tracking feature of our tool.

Reading & focus order lens. Auditing using the reading & focus order lens showed good results in article descriptions in the website. However, traversal through toolbars, legends, and chart panels diverged sharply from the visual reading flow, with the AX and keyboard orders drift from the visual order. We found key errors in the accessibility of the chart panels that they have many artificial elements that are only added for visual interactivity purposes, however, they are visually hidden, selectable with keyboard, and screen reader readable without any proper alt-texts or names.

6 Discussion

A central contribution of the workflow is a shift in the unit of accessibility review. Traditional reports enumerate violations by rule or element, which is necessary but often insufficient for visualization-rich pages where accessibility depends on relationships among encodings, controls, legends, summaries, and navigation paths. By organizing evidence into two coordinated lenses, the tool supports a design-oriented conversation: what is the reader trying to learn, what evidence is available across modalities, and where does the page assume a particular sensory or interaction pathway?

Our deployments also showed that popular libraries remain a bottleneck: Leaflet.js and Plotly.js limited how far keyboard order and operability could be fixed regardless of author intent.

Limitations. The interaction lens audits the states its automation can reach; flows behind authentication or bespoke gestures may go unvisited, so absence of a finding is not proof of conformance. Reconstructing the AX and visual orders relies on heuristics that can misjudge unusual layouts, and the layout-aware classifier can mislabel a deliberate pattern or a genuine barrier. We added an ignore functionality to mitigate certain cases. Finally, our evaluation is formative; controlled studies with dashboard authors and accessibility practitioners remain future work, alongside broader criterion coverage and deeper integration with assistive-technology APIs.

Acknowledgments

This work was supported by the Centers for Disease Control and Prevention's Center for Forecasting and Outbreak Analytics Cooperative agreement CDC-RFA-FT-23-0069. We acknowledge the feedback and collaborative efforts of Nathan Lacross, Kerry Regan, Mary Jewell, Bree Barbeau, and Abigail Collingwood from the Utah DHHS and Matthew Samore and George Vega Yon at the University of Utah.

Use of Generative AI

The authors used various generative AIs to improve the manuscript's grammar and writing and to develop the Chrome extension. The authors reviewed, edited, fact-checked, and take full responsibility for all content.

References

[1] B. Ansari and E. G. Martin. Development of a usability checklist for public health dashboards to identify violations of usability principles. *Journal of the American Medical Informatics Association*, 29(11):1847–1858, 2022. doi: 10.1093/jamia/ocac140 2

[2] A. Arleo, R. Borgo, J. Kohlhammer, R. A. Ruddle, H. Scharlach, and X. Yuan. Reflections on the use of dashboards in the COVID-19 pandemic. *IEEE Computer Graphics and Applications*, 45(2):135–142, 2025. doi: 10.1109/MCG.2025.3538257 4

[3] B. Bach, E. Freeman, A. Abdul-Rahman, C. Turkay, S. Khan, Y. Fan, and M. Chen. Dashboard design patterns. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):342–352, 2023. doi: 10.1109/TVCG.2022.3209448 1

[4] 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, 2021. 2

[5] G. Brajnik, Y. Yesilada, and S. Harper. Is accessibility conformance an elusive property? a study of validity and reliability of wcag 2.0. *ACM Trans. Access. Comput.*, 4(2), article no. 8, 28 pages, Mar. 2012. doi: 10.1145/2141943.2141946 2

[6] 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*, CHI '23, article no. 45, 17 pages. Association for Computing Machinery, New York, NY, USA, 2023. doi: 10.1145/3544548.3580749 2

[7] Chrome for Developers. Introduction to Lighthouse. https://developer.chrome.com/docs/lighthouse/overview/. 1

[8] Chrome for Developers. Lighthouse accessibility score. https://developer.chrome.com/docs/lighthouse/accessibility/scoring. 1

[9] M. D. Clarkson. Web-based COVID-19 dashboards and trackers in the United States: Survey study. *JMIR Human Factors*, 10(1):e43819, 2023. doi: 10.2196/43819 4

[10] N. Dasgupta and F. Kapadia. The future of the public health data dashboard. *American Journal of Public Health*, 112(6):886–888, 2022. doi: 10.2105/AJPH.2022.306871 4

[11] Deque Systems. axe-core: Accessibility engine for automated testing of HTML-based user interfaces. https://github.com/dequelabs/axe-core. 1

[12] E. Dong, H. Du, and L. Gardner. An interactive web-based dashboard to track COVID-19 in real time. *The Lancet Infectious Diseases*, 20(5):533–534, 2020. doi: 10.1016/S1473-3099(20)30120-1 4

[13] M. Elshehaly, R. Randell, M. Brehmer, L. McVey, N. Alvarado, C. P. Gale, and R. A. Ruddle. QualDash: Adaptable generation of visualisation dashboards for healthcare quality improvement. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):689–699, 2021. doi: 10.1109/TVCG.2020.3030424 4

[14] D. Fan, A. Fay Siu, H. Rao, G. S.-H. Kim, X. Vazquez, L. Greco, S. O'Modhrain, and S. Follmer. The accessibility of data visualizations on the web for screen reader users: Practices and experiences during covid-19. *ACM Trans. Access. Comput.*, 16(1), article no. 4, 29 pages, Mar. 2023. doi: 10.1145/3557899 1

[15] B. Lechner and A. Fruhling. Towards public health dashboard design guidelines. In *HCI in Business*, pp. 49–59. Springer International Publishing, 2014. doi: 10.1007/978-3-319-07293-7_5 4

[16] J. Mankoff, H. Fait, and T. Tran. Is Your Web Page Accessible? A Comparitive Study of Methods for Assessing Web Page Accessibility for the Blind. 6 2018. doi: 10.1184/R1/6470195.v1 2

[17] C. Power, A. Freire, H. Petrie, and D. Swallow. Guidelines are only half of the story: accessibility problems encountered by blind users on the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, 10 pages, p. 433–442. Association for Computing Machinery, New York, NY, USA, 2012. doi: 10.1145/2207676.2207736 2

[18] R. Rabiei, P. Bastani, H. Ahmadi, S. Dehghan, and S. Almasi. Developing public health surveillance dashboards: A scoping review on the design principles. *BMC Public Health*, 24(1):392, 2024. doi: 10.1186/s12889-024-17841-2 4

[19] F. Rosa Martins, L. Seixas Pereira, and C. Duarte. Qualstate: Finding website states for accessibility evaluation. In *Proceedings of the 21st International Web for All Conference*, W4A '24, 10 pages, p. 96–105. Association for Computing Machinery, New York, NY, USA, 2024. doi: 10.1145/3677846.3677851 2

[20] M. Roser. Our world in data. https://ourworldindata.org/. 1, 4

[21] A. Sarikaya, M. Correll, L. Bartram, M. Tory, and D. Fisher. What do we talk about when we talk about dashboards? *IEEE Transactions on Visualization and Computer Graphics*, 25(1):682–692, 2019. doi: 10.1109/TVCG.2018.2864903 1

[22] A. G. Schiavone and F. Paternò. An extensible environment for guideline-based accessibility evaluation of dynamic web applications. *Univers. Access Inf. Soc.*, 14(1):111–132, 22 pages, Mar. 2015. doi: 10.1007/s10209-014-0399-3 2

[23] A. Schulze, F. Brand, J. Geppert, and G.-F. Böl. Digital dashboards visualizing public health data: A systematic review. *Frontiers in Public Health*, 11:999958, 2023. doi: 10.3389/fpubh.2023.999958 4

[24] A. F. Siu, D. Fan, G. S.-H. Kim, H. V. Rao, X. Vazquez, S. O'Modhrain, and S. Follmer. COVID-19 highlights the issues facing blind and visually impaired people in accessing data on the web. In *Proceedings of the 18th International Web for All Conference*, W4A, article no. 11, pp. 11:1–11:15. ACM, 2021. doi: 10.1145/3430263.3452432 1

[25] G. Stahlman, I. Yanovitzky, and M. Kim. Design, application, and actionability of US public health data dashboards: Scoping review. *Journal of Medical Internet Research*, 27:e65283, 2025. doi: 10.2196/65283 4

[26] H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda. Accessibility designer: Visualizing usability for the blind. In *ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS, pp. 177–184, 2004. doi: 10.1145/1029014.1028662 2

[27] U.S. Department of Justice, Civil Rights Division. Fact sheet: New rule on the accessibility of web content and mobile apps provided by state and local governments, 2024. https://www.ada.gov/resources/2024-03-08-web-rule/. 1

[28] Utah Department of Health & Human Services. Utah wastewater surveillance system. https://avrpublic.dhhs.utah.gov/uwss/. 2, 4

[29] M. Vigo, J. Brown, and V. Conway. 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*, W4A '13, article no. 1, 10 pages. Association for Computing Machinery, New York, NY, USA, 2013. doi: 10.1145/2461121.2461124 1, 2

[30] W3C Accessibility Guidelines Working Group. Web content accessibility guidelines (WCAG) 2.2. W3C recommendation, World Wide Web Consortium, 2023. https://www.w3.org/TR/WCAG22/. 1, 2

[31] W3C Web Accessibility Initiative. About ACT rules. https://www.w3.org/WAI/standards-guidelines/act/rules/about/. 1

[32] W3C Web Accessibility Initiative. Evaluating web accessibility overview. https://www.w3.org/WAI/test-evaluate/. 1

[33] W3C Web Accessibility Initiative. Evaluation tools overview. https://www.w3.org/WAI/test-evaluate/tools/. 1

[34] W3C Web Accessibility Initiative. What's new in WCAG 2.2. https://www.w3.org/WAI/standards-guidelines/wcag/new-in-22/. 1, 2

[35] WebAIM. WAVE web accessibility evaluation tools. https://wave.webaim.org/. 1

[36] WebAIM. The WebAIM million: The 2026 report on the accessibility of the top 1,000,000 home pages. https://webaim.org/projects/million/. 1

[37] Y. Zhang, Y. Sun, J. D. Gaggiano, N. Kumar, C. Andris, and A. G. Parker. Visualization design practices in a crisis: Behind the scenes with COVID-19 dashboard creators. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):1037–1047, 2023. doi: 10.1109/TVCG.2022.3209493 4

[38] J. Zong, C. Lee, A. Lundgard, J. Jang, D. Hajas, and A. Satyanarayan. Rich Screen Reader Experiences for Accessible Data Visualization. *Computer Graphics Forum (Proc. EuroVis)*, 2022. doi: 10.1111/cgf.14519 1