Using OpenKeyNav to Enhance the Keyboard-Accessibility of Web-based Data Visualization Tools

Lawrence Weru, Sehi L'Yi, Thomas C. Smits, and Nils Gehlenborg

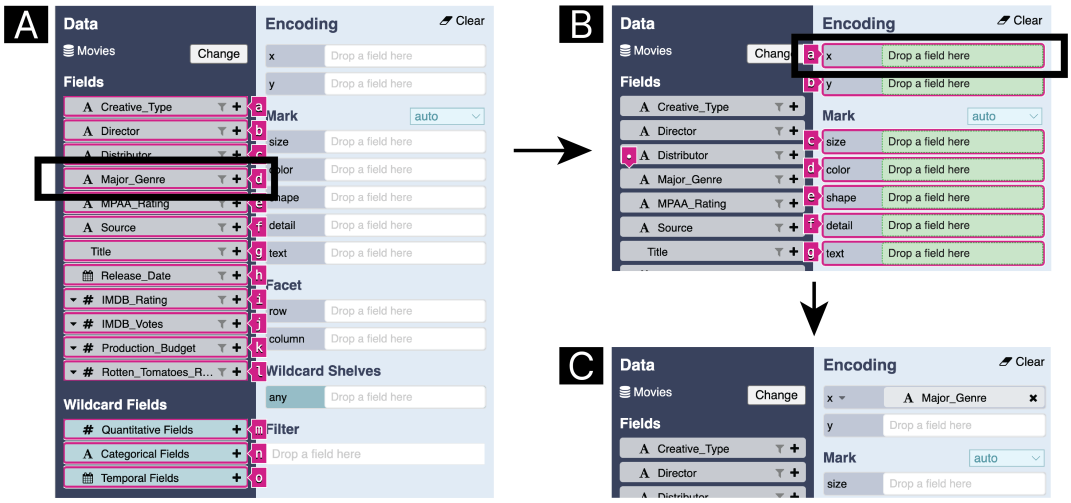

Fig. 1: Drag-and-drop in OpenKeyNav demonstrated in Voyager 2. (A) After OpenKeyNav's drag mode is initiated through a shortcut key, the elements that can be dragged (Voyager's data fields or "pills" in this case) are outlined and assigned shortcut key labels. Pressing the corresponding key, e.g., "d" for `Major_Genre` (indicated with a black rectangle) selects the element to be dragged. (B) Shelves where the element can be dropped are then assigned shortcut key labels and outlined. An element can be dropped by pressing the corresponding key, e.g., "a" for the x-axis (indicated with a black rectangle). (C) After a shelf is selected, the pill is dropped on it.

Abstract—Many data visualization tools require a mouse. While such tools widen access to data communication and expression, their implementations are difficult or impossible to use by people with certain disabilities who experience difficulties using a mouse. What if people could use them as easily with a keyboard? We present OpenKeyNav, a zero-dependency JavaScript code library that exposes a developer-friendly API for initiating keyboard accessibility enhancements. We demonstrate a usage scenario of OpenKeyNav for improving the keyboard-accessibility of Voyager 2, an open-source web-based data visualization tool based on the shelf configuration similar to industry-leading Tableau. Since mouse-driven interactions such as drag-and-drop are found in software in a broad range of industries, the interaction methods we describe have potential implications for the education, employment, and autonomy of people with motor disabilities in various fields. A demonstration is at https://voyager-keyboard-demo.github.io/. Its instructions are at https://github.com/voyager-keyboard-demo/voyager-keyboard-demo.github.io/ The most up-to-date version of the preprint can be accessed at https://osf.io/preprints/osf/3wjsa.

Index Terms—Accessibility, visualization, keyboard interaction, drag-and-drop interaction

1 Introduction

Data visualization tools often rely on mouse-only interactions. For example, consumer-facing tools such as the widely used Tableau [8] depend on drag-and-drop to enable users to create and explore visualizations without writing code. While such tools widen access to data communication and expression, their implementations are often difficult or impossible to use by people that pointer devices aren't designed for, such as people with specific motor disabilities (PMDs). As a result, the developers enable some people to create data visualizations with these tools while disabling PMDs from doing so, limiting their participation in the data visualization field and their access to education, jobs, social inclusion, and autonomy. These limitations harm our most vulnerable populations disproportionately, given that the World Health Organization (WHO) recognizes access to education and jobs as Social Determinants of Health, and the National Institutes of Health (NIH) recognizes people with disabilities as a population with health disparities.

One way to make data visualization authoring tools usable by a broader range of people is to make them fully keyboard-accessible. Keyboard-accessible software is also usable by people who use other input interfaces, such as people who control their devices using their voice, since many assistive technologies and alternative input devices leverage the keyboard APIs.

Many data visualization authoring tools have a web interface, and it is possible to make controls on websites and web apps keyboard-accessible through semantic HTML and Accessible Rich Internet Applications (ARIA) [15] attributes in compliance with the Web Content Accessibility Guidelines (WCAG) [14]. However, most web developers do not develop WCAG-compliant websites and web apps. For example, 96% of the top 1,000,000 home pages have accessibility barriers [16], suggesting that the root causes of an inaccessible web are systemic. However, even if developers were to follow all of the web content accessibility guidelines, open problems in digital accessibility remain for complex interactions, such as drag-and-drop and navigating complex

• *Lawrence Weru, Sehi L'Yi, Thomas C. Smits, and Nils Gehlenborg are with Harvard Medical School. E-mail: { lawrence_weru, sehi_lyi, tsmits, nils } @hms.harvard.edu*

interfaces.

Several interaction methods have been proposed in the literature and implemented in the industry, such as browser plugins (e.g., Vimium [6]), OS-level plugins (e.g., ShortCat [12]), and developer-level integration [11]. However, these solutions do not facilitate complex interactions like drag-and-drop, sometimes fail to identify every clickable element, leave developers out of the equation unaware of these issues, or impose unequal time burdens on keyboard users due to cumbersome selection methods.

This paper introduces a novel interaction method for executing drag-and-drop via keyboard. This method, along with a corresponding code library OpenKeyNav, empowers web developers to implement this and other interaction methods, such as clicking on clickable elements and scrolling through scrollable regions. The novelty and main contribution of this method lies in its ability to facilitate complex interactions exclusively, efficiently, and reliably via keyboard.

The proposed solution not only enables web developers to identify and remediate keyboard-related accessibility failures quickly but also provides enhanced keyboard functionality to improve the efficiency of their web apps for people with motor disabilities and those who prefer not to use a mouse.

2 Related Work

If inaccessible websites are a systemic problem, then accessibility solutions can be viewed as attempted interventions. Systemic problems often require a series of interventions that address various interconnected aspects of the system to create conditions for systemic change.

Keyboard-accessibility interventions can be classified by their point of intervention in the web development and consumption system. Some interventions target the builders of the web (web developers and designers), who integrate them into their websites (such as style guides, developer APIs and code libraries). Other interventions target the users of the web, who adopt and use them to navigate various websites (such as browser extensions or desktop software).

Somani et al. [11] introduce a developer-level intervention; a JavaScript API that helps developers provide a keyboard-accessible drag-and-drop interaction to end users. However, their proposed user experience is not equivalent to the experience of mouse users. They enable mouse users to directly select the element they want to drag while requiring keyboard users to tab through potentially many elements to get to the one they want to drag. The time cost imposed on keyboard users grows with the size of the list of draggable elements and applicable drop targets and is multiplied by each drag-and-drop interaction. Instead, OpenKeyNav focuses on creating an equivalent experience for keyboard users, enabling them to select both the drag element and the drop target directly.

End users may employ various browser-level and OS-level interventions such as Vimperator [13], Conkeror [5], Hit-a-Hint [10], ShortCat [12], and Voice Control of OS X [2] to enable keycode-driven clicking of clickable elements. However, those tools may not identify all clickable elements. Some OS-level interventions such as VoiceOver [1] and JAWS [9] offer drag-and-drop commands, but they are not known to operate reliably. As a result, keyboard users may default to seeking workarounds that can impose greater physical, cognitive, or time demands. While following accessibility best practices such as WGAC and ARIA can help web developers ensure compatibility for standards-compliant third-party navigation tools, it is not realistic for developers to test and provide support for every toolkit that end users might use with their website to ensure compatibility. Lastly, debugging the keyboard-accessibility of a website using these solutions is challenging because these tools do not provide helpful debugging information to developers.

3 Proposed Solution

3.1 Keyboard-based Interactions

Drag-and-Drop Using a mouse, users can execute drag-and-drop in four steps, (1) mousing over to the element they want to drag, (2) clicking down on the element, (3) dragging the element to the drop target, (4) releasing the click. With the proposed solution, users can execute a three-step drag-and-drop using a keyboard by (1) typing a developer-configurable key code such as "m" to enter drag mode, (2) typing a dynamically generated unique key code, typically 1-2 letters, to directly select a draggable element, (3) typing a similarly generated key code to "drop" the element on that target. This interaction is demonstrated in 4.1.

Scrollable Regions Using a mouse, users can execute a scroll in two steps, (1) mousing over to the region they want to scroll, 2) scrolling the region element with scroll wheel (trackpad swipe), or the scrollbar that appears. Traditionally, users must achieve keyboard focus on an element inside of a scrollable region before they can scroll the region using spacebar or arrow keys. To get there, users are required to tab through keyboard-focusable elements on the page in the order they appear in the DOM Dense interfaces such as data visualization authoring tools can have hundreds of focusable elements on the page, imposing a time burden. With the proposed solution, users can cycle through scrollable regions by entering a keycode, such as "s," quickly moving the active focus to the bounding box of the next scroll region, increasing efficiency by skipping over all of the tab stops in-between. Then, the user can scroll the region using spacebar or arrow keys as usual.

Click Using a mouse, users can click on an element in two steps, (1) mousing over to the element they want to click, (2) clicking on it. With the proposed solution, users can execute a two-step drag-and-drop using a keyboard by (1) typing a developer-configurable key code such as "k" to enter click mode, (2) typing a dynamically generated unique key code, typically 1-2 letters, to click the element they want to click. While similar interactions are seen in browser extensions such as Vimium [6], Shortcat [12], and Voice Control [2], this developer-level intervention provides an improved user experience using collision detection to minimize keycode labels covering the clickable elements or other labels, as well as outlining the elements being labeled. Its debug mode helps developers identify inaccessible interactive elements so that they can remediate them.

Debug Mode A debug mode is on by default which highlights the mouse-clickable elements that aren't keyboard-focusable. When people develop web elements that are mouse-clickable but not keyboard-focusable, they limit the groups of people who can interact with those elements, violating WCAG guidelines. Such accessibility failures are often attributed to the accessibility barriers remaining invisible to developers. A keyboard-accessibility developer tool creates an opportunity for exposing keyboard-related accessibility barriers of a webpage to its developers. This default setting, which can be disabled in production, makes such accessibility barriers visible to developers so they can remediate them during development.

3.2 Developer Integration

OpenKeyNav is a zero-dependency JavaScript code library that exposes a developer-friendly API for initiating keyboard accessibility enhancements, as well as customizing parameters (e.g., a modifier key, color or labels).

4 Demonstration

In this demonstration, OpenKeyNav is implemented into Voyager 2 [17], an open-source data visualization authoring tool The demonstration showcases the authoring of a complex data visualization exclusively via keyboard from start to finish, highlighting the effectiveness and potential impact of the proposed solution. The demonstration is available as a hosted TypeScript React application at https://voyager-keyboard-demo.github.io/.

A hypothetical persona, Alex, works as a data analyst for a hypothetical film production company. During a recent meeting, senior executives at Alex's company express concerns about the declining box office revenue and mixed audience reviews of their recent movie releases, highlighting the need for a more data-driven approach for their next project. They appoint Alex to create data visualizations using the default `Movies` dataset in Voyager. Unfortunately, Alex recently developed a repetitive strain injury (RSI). The long hours spent at the

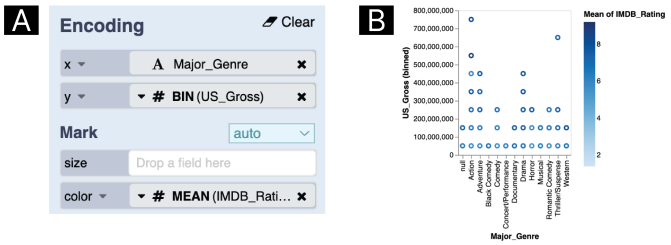

Fig. 2: Data visualization in Voyager. (A) The channel configuration for the Target Starting Graph. (B) The Target Starting Graph.

computer and continuous mouse usage exacerbate the underlying strain in their wrist and hand, causing significant pain. With this condition, it is difficult for Alex to use a mouse effectively, but they can still use their keyboard.

Here, we demonstrate how the interaction methods and code library introduced in this paper enable Alex to create a data visualization efficiently using a keyboard, ensuring they can continue their work despite their RSI challenges.

4.1 Keyboard-Accessible Drag-and-Drop

The executives request an analysis of which movie genres are performing well in terms of revenue and audience ratings. Alex wants to recreate a graph that their marketing manager shared to explore more effective visualizations It visualizes the genre (`Major_Genre`), gross revenue (`US_Gross`), and rating (`IMDB_Rating`) for each film in the dataset. Alex is comfortable with data visualization authoring tools like Voyager [17], which allow users to control the structure and appearance of visualizations by placing "pills" representing data "fields" on different encoding "shelves" (visual channels) via drag-and-drop interaction. To create the Target Starting Graph in Fig. 2B, Alex needs to place the `Major_Genre` pill in the x-axis, `US_Gross` in the y-axis, and `IMDB_Rating` as the color channel (Fig. 2A).

Drag Mode Such drag-and-drop actions typically require a mouse, but with OpenKeyNav Alex can move these pills using their keyboard by leveraging the accessible drag-and-drop interaction. Alex types a developer-configurable keyboard command, in this case "m" to enter drag mode (Fig. 1A). In drag mode, draggable elements are labeled with dynamically generated keyboard shortcuts. These labels are given a strong contrast ratio and a dual-tone outline that is distinguishable on both light and dark backgrounds by default. Typing the key combination of one of these labels selects the respective element as the element to be dragged. Once a draggable element is selected, its drop zones (i.e., droppable regions) are similarly given dynamically-generated keyboard commands.

Drag-and-Drop Alex wants to place `Major_Genre` on the x-axis, so they type the shortcut for selecting the `Major_Genre` pill (in this case, "d") (Fig. 1A) to begin moving it. The software gives the pill a visible indicator that it is being dragged. Then the software labels the pill's drop targets with keyboard shortcuts (Fig. 1B). Developers often add drag effects to their user interfaces. In the case of Voyager, drop zones are highlighted green when a draggable element is dragged with a mouse. These effects should also be shown to keyboard users executing a drag and drop. OpenKeyNav dispatches a concoction of mouse drag events such as "`mousedown`" and "`dragstart`", to help trigger any user interface changes.

Alex types the dynamically-generated keyboard shortcut for the x-axis drop target ("a"), dropping the selected pill into the intended encoding shelf (Fig. 1C).

Through this simple interaction, they are able to drag and drop the three pills to their respective shelves, quickly and efficiently. They can author the desired data visualization using just a few keyboard commands. Notably, they did not have to memorize any of the keyboard commands used inside drag mode, as they were presented on a just-in-time basis. This reduced cognitive load increases the accessibility of the interaction method.

Sticky Drag Sometimes Alex wants to move a pill around while observing the output, as part of an exploratory strategy to data visualization authoring. He doesn't want to go in and out of drag mode each time he wants to move the same pill around. Alex can use a modifier key while initiating drag mode to enter a sticky drag mode, in this case "Shift", which enables Alex to move the selected draggable element to different drop zones. When in this mode, he can type the keyboard commands of the drop targets as usual to move the pill around without exiting drag mode. This is especially useful when Alex wants to quickly see how different visual channels produce different visualization outputs using the same field.

Tab to Drag While in Sticky Drag mode, he can press the Tab key to move the pill to the next drop target. A modifier key, in this case shift, in combination with Tab, moves the pill to the previous drop target. This enables them to keep their eyes on the data visualization output and tab around to see different possibilities.

Click Mode In order to produce the intended visualization, Alex needs to add a "bin" function to the `US_Gross` pill, and a "mean" function to the `IMDB_Rating` pill. When a user clicks on the dropdown on the respective pill, these functions appear in a popup menu.

While it is possible to click on such elements using the spacebar when they are keyboard-focusable and properly-coded with semantic HTML, placing keyboard focus on the right element is not trivial. Traditional keyboard navigation requires tabbing through the elements on the page in sequence before placing keyboard focus on the desired element. For highly interactive web-apps with potentially hundreds of clickable elements on a page, having no option besides tabbing through each of them can impose a disability tax in terms of time costs.

To avoid having to tab through the interface to click on a desired element and achieve an equivalent experience to mouse users, Alex types a developer-configurable keyboard shortcut, in this case "k", to enter click mode. (Figure). Like browser extensions such as Vimium or OS-level software such as Shortcat, click mode labels the clickable elements on the page with keyboard shortcuts, enabling Alex to click the dropdown for the `US_Gross` pill (Figure). Entering click mode again, he can click the "bin" function. He does the same to add the "mean" function to the `IMDB_Rating` pill. Since this software is a developer tool, the developers ensured clickable elements were fully functional when integrating the tool. Browser extensions and other third-party tools that the developers do not integrate into their websites cannot ensure compatibility, limiting their reliability. However, developers who integrate a keyboard-accessibility tool into their websites take ownership of the user experience, helping to ensure a reliable one.

4.2 Developer-Friendly API

To enable keyboard-accessible data visualization authoring for Alex and others, developers in Alex's company add OpenKeyNav, a JavaScript code library, to their self-hosted instance of Voyager, an open-source React app written in TypeScript.

Simple Setup Inside of `App.tsx`, the developers import the library and configure it.

```
// App.tsx:
    import OpenKeyNav from 'openkeynav';
```

If drag-and-drop is not needed, all that is required to initiate the code library with all default settings is:

```
// App.tsx > componentDidMount():
    const openKeyNav = new OpenKeyNav();
    openKeyNav.init();
```

Drag-and-Drop To initiate with drag-and-drop configured, the developers use standard CSS selectors to define the `field pill` draggable elements (`fromElements`) and their `encoding shelf` drop zones (`toElements`). Voyager has two types of field pills, regular and "wildcard." All pills can be dropped into encoding shelves, but only regular field pills can be dropped into Voyager's "filter" pane. Using OpenKeyNav, the developers define two from-to pairs, one for regular field pills, and another for wildcard field pills. All field pills are given a `field-pill` class name. Wildcard field pills are given an additional `wildcard` class name.

```
openKeyNav.init({
  modesConfig:
    { move: { config: [
      { fromElements: ".field-pill",
        toElements: ".encoding-shelf, .filter-pane" },
      { fromElements: ".field-pill.wildcard",
        toElements: ".encoding-shelf" }
  ]}}
});
```

For more granularity, OpenKeyNav's API provides additional parameters to define the draggable elements' selection criteria, such as direct children inside a `fromContainer`, or `resolveFromElements`, which accepts a function that returns a list of DOM elements on the page to be treated the same as `fromElements`. Developers can also define exclusion criteria (`fromExclude`).

In Drag Mode, after a draggable element and its drop target are selected, the code library mimics the mouse's drop behavior. In the case of Voyager, this removes the need for a callback function. However, a `callback` function can be used to ensure the proper actions occur after the drop.

Developers can configure the visual appearance of the key code labels and focus ring , and some of the key codes .

5 Discussion

Safety Assistive technologies not only remove access barriers but also minimize health risks. Computer technologies can be designed to interface with our bodies in ways that cause physiological harm. Notable examples include seizures triggered by flashing lights for individuals with photosensitive epilepsy, neck and back pain from poor ergonomic posture during extended computer use, migraines and computer vision syndrome from prolonged screen exposure, and sleep disruption and reduced melatonin production from exposure to blue light at night. These effects can be especially harmful for people with musculoskeletal disorders, visual impairments, and sleep disorders, and in the case of evening blue light exposure of women who are pregnant, the reduced birth weight of infants. [7]

Keyboard interfaces that require significantly more actions for keyboard users compared to mouse users impose a disproportionate physical, cognitive, and time burden on keyboard users. The increased physical burden of such keyboard interfaces can lead to repetitive stress injuries such as carpal tunnel syndrome or "Emacs pinky." Keyboard interfaces commonly require users to memorize a long list of shortcuts, what they do, and when they can use them. In addition to increasing stress for users, such high demands on working memory, cognitive load, attention and focus, and organization and planning can create access barriers for individuals with executive function impairments. Increased time demands can create barriers for completing tasks.

Keyboard accessibility interventions must aim to minimize cognitive load, physical burden, and additional time demands. For reducing cognitive load, this toolkit leverages mitigation strategies such as visual aids, on-demand shortcuts, and uniform design. For reducing physical burden, the novelty of OpenKeyNav's keyboard-accessible drag and drop is that users can directly select their intended element and drop zone, without needing to traverse a series of elements through repetitive key presses. To reduce the physical and cognitive burden on users who rely on magnification, OpenKeyNav's labels and elements scale with the browser zoom and re-flow to avoid collisions with each other, remaining clear and usable in a range of scales. To minimize additional time demands, the time complexity of this keyboard-accessible drag-and-drop is comparable to that of the mouse-based action, since the number of steps a user takes does not increase with the number of the selectable elements. The number of steps required for keyboard-based clicking is also comparable to mouse-based clicking.

Effectiveness As of today, there is no automated way to ensure that a website is fully accessible, and this tool does not attempt to function as an automated one. Instead, it observes a need for behavior change among the builders of the web, who usually do not test their websites for keyboard-accessibility. The web is an inaccessible environment because people publish inaccessible code. With a debug mode turned on by default, this keyboard-accessibility toolkit attempts to function as a behavior change intervention, encouraging developers to test their websites for keyboard-accessibility when they integrate this tool. An accessibility intervention that targets the website-building process and interacts with the builders themselves stands a chance at producing positive, sustainable, systemic change.

Limitations Formal user tests are not yet conducted. It is important to keep the focus on the lived experience of people with disabilities. Evaluation, refinement, and user testing can help ensure that the focus remains on the impact on people's quality of lives instead of the technology.

Future work may include user studies with target populations including people who use a range of assistive technologies, as well as exploring opportunities for co-designing. The feature roadmap includes exploring support for more mouse interactions such as hovering on hoverable elements and dragging sliders, as well as providing customizable keyboard navigation aids such as skip links. Optimized key code combinations can improve accessibility for people with fine motor skill impairments, people with dyslexia, or people who use switch-input devices. End-user-defined shortcuts and guided tutorials can potentially improve usability. OpenKeyNav is under active development and refinement and is designed to evolve in response to user needs. The most up-to-date version of the preprint can be accessed at https://osf.io/preprints/osf/3wjsa.

6 Conclusion

Data visualization tools have become critical in various industries, underscored by SalesForce's 2019 acquisition of industry-leading Tableau [8] for $15.7 Billion [4].

Web apps in various fields have similar mouse-driven interactions that can benefit from such a tool. OpenKeyNav was originally developed to provide a keyboard-accessible user experience in a note-taking app [3], which has a different user interface than data visualization tools. Therefore such a tool has a broad range of applications including but not limited to e-learning tools, video games, web development and design tools, content management systems, project management software, e-commerce platforms, business intelligence dashboards, general websites and web apps, and assistive technologies.

7 Acknowledgments

We are grateful to Trevor Manz, Huyen Nguyen, David Kouril, PJ Van Camp, Eric Moerth, and Sofía Rojas for assistance with exposure to prior art, research goals and aims, informal testing, and general feedback. We thank the creators of Voyager for open sourcing their tool, enabling it to be used as part of the demonstration.

References

[1] Apple. Use VoiceOver to drag and drop items on Mac. https://support.apple.com/guide/voiceover/drag-and-drop-items-vo14056/mac. Accessed: 2024-9-2. 2

[2] Apple. Voice Control: Use Voice Control on your iPhone, iPad, or iPod touch. https://support.apple.com/en-us/111778. Accessed: 2024-7-24. 2

[3] Aster Enterprises LLC. Columns. https://app.columnsapp.com/. Accessed: 2024-7-26. 4

[4] D. Clarke. Making data coherent drives salesforce's $15.3 billion deal for tableau. https://www.nytimes.com/2019/06/10/technology/salesforce-tableau-deal.html, jun 2019. Accessed: 2024-7-26. 4

[5] Conkeror developers. Conkeror: a keyboard-oriented, highly-customizable, highly-extensible web browser. http://conkeror.org/. Accessed: 2024-7-24. 2

[6] S. B. Ilya Sukhar, Phil Crosby. Vimium: A browser extension that provides keyboard-based navigation and control of the web. https://github.com/philc/vimium. Accessed: 2024-7-24. 2

[7] Izci Balserak, B., Hermann, R., Hernandez, T. L., Buhimschi, C., Park, C. Evening blue-light exposure, maternal glucose, and infant birthweight. *Annals of the New York Academy of Sciences*, 1515(1):276–284, 2022. doi: 10.1111/nyas.14852 4

[8] I. Salesforce. Tableau: Business Intelligence and Analytics Software. https://www.tableau.com/. Accessed: 2024-7-24. 1, 4

[9] F. Scientific. JAWS Hotkeys. https://www.freedomscientific.com/training/jaws/hotkeys/. Accessed: 2024-9-2. 2

[10] slaypni. Hit-a-Hint: Surf web with a keyboard. https://chromewebstore.google.com/detail/moly-hah/pjoacnohgednppackhamgfalpkffeeek?hl=en. Accessed: 2024-7-24. 2

[11] R. Somani, J. Xin, B. Bhaskar Deo, and Y. Huang. Building keyboard accessible drag and drop. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility - ASSETS '14*. ACM Press, New York, New York, USA, 2014. doi: 10.1145/2661334.2661342 2

[12] Sproutcube. ShortCat: The universal command palette for your Mac. https://shortcat.app/. Accessed: 2024-7-24. 2

[13] M. Stubenschrott. Vimperator: A Firefox browser extension with strong inspiration from the Vim text editor. http://vimperator.org/. Accessed: 2024-7-24. 2

[14] W3C. Web Content Accessibility Guidelines (WCAG) 2.1. https://www.w3.org/TR/WCAG21/, 2023. 1

[15] W3C. WAI-ARIA Overview. https://www.w3.org/WAI/standards-guidelines/aria/, 2024. 1

[16] WebAIM. The webaim million; the 2024 report on the accessibility of the top 1,000,000 home pages. https://webaim.org/projects/million/, Mar 2024. Accessed: 2024-7-25. 1

[17] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pp. 2648–2659. Association for Computing Machinery, New York, NY, USA, May 2017. doi: 10.1145/3025453.3025768 2, 3