InSupport: Proxy Interface for Enabling Efficient Non-Visual Interaction with Web Data Records

Javedul Ferdous
Old Dominion University
Norfolk, VA, USA
mferd002@odu.edu

Hae-Na Lee
Stony Brook University
Stony Brook, NY, USA
haenalee@cs.stonybrook.edu

Sampath Jayarathna
Old Dominion University
Norfolk, VA, USA
sampath@cs.odu.edu

Vikas Ashok
Old Dominion University
Norfolk, VA, USA
vganjigu@odu.edu

ABSTRACT

Interaction with web data records typically involves accessing *auxiliary* webpage segments such as filters, sort options, search form, and multi-page links. As these segments are usually scattered all across the screen, it is arduous and tedious for blind users who rely on screen readers to access the segments, given that content navigation with screen readers is predominantly one-dimensional, despite the available support for skipping content via either special keyboard shortcuts or selective navigation. The extant techniques to overcome inefficient web screen reader interaction have mostly focused on general web content navigation, and as such they provide little to no support for data record-specific interaction activities such as filtering and sorting – activities that are equally important for enabling quick and easy access to the *desired* data records. To fill this void, we present InSupport, a browser extension that: (i) employs custom-built machine learning models to automatically extract auxiliary segments on any webpage containing data records, and (ii) provides an instantly accessible proxy *one-stop* interface for easily navigating the extracted segments using basic screen reader shortcuts. An evaluation study with 14 blind participants showed significant improvement in usability with InSupport, driven by increased reduction in interaction time and the number of key presses, compared to state-of-the-art solutions.

CCS CONCEPTS

• **Human-centered computing → Accessibility technologies**.

KEYWORDS

Web accessibility, Blind, Visual impairment, Screen reader, Data records

ACM Reference Format:
Javedul Ferdous, Hae-Na Lee, Sampath Jayarathna, and Vikas Ashok. 2022. InSupport: Proxy Interface for Enabling Efficient Non-Visual Interaction with Web Data Records. In *27th International Conference on Intelligent User Interfaces (IUI '22), March 22–25, 2022, Helsinki, Finland.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3490099.3511126

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
IUI '22, March 22–25, 2022, Helsinki, Finland
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9144-3/22/03…$15.00
https://doi.org/10.1145/3490099.3511126

1 INTRODUCTION

Interaction with web data records (e.g., products, flights, posts, job listings, emails) is an integral part of everyday web activities such as shopping, seeking information, communication, and social networking. To facilitate convenient interaction with web data records, modern web applications typical arrange these records in the form of lists or grids, and additionally provide various *auxiliary segments* such as filters, sort options, search form, and multi-page links surrounding the records as shown in Figure 1. While sighted users can indeed easily access these auxiliary segments using a pointing device such as a mouse and thereby quickly locate the desired data records, blind users on the other hand struggle to do the same using a screen reader (e.g, JAWS, NVDA, VoiceOver) – their 'go-to' assistive technology for interacting with web applications. This is because the screen reader primarily supports one-dimensional navigation of content (e.g., 'H' shortcut press for the next heading), and thus the blind users have to endure an arduous and tedious process involving a multitude of shortcut presses for navigating back-and-forth between the data records and the auxiliary segments (see Figure 1) [8, 17]. While this interaction burden is reduced to a certain extent in some screen readers which support alternative modalities (e.g., *rotor* in VoiceOver allows users to choose and selectively navigate certain types of HTML elements on the webpage), the onus of locating the auxiliary segments nonetheless still remains on the users. Moreover, to exploit this additional screen reader support, the users should have apriori knowledge of the HTML element types (heading, link, button, etc.) of auxiliary segments and their components; remembering such webpage-specific details can induce cognitive overhead for blind users [15].

Existing research solutions to address this problem are mostly based on speech interaction [8, 21], which is not only susceptible to noise but also requires a tighter integration of a third party speech service with the user's screen reader, thereby limiting their practicality to a few open-source screen readers. The use of speech in public settings has also shown to cause privacy issues for blind users [2, 40]. The few extant non-speech solutions on the other hand have all primarily focused on general webpage navigation

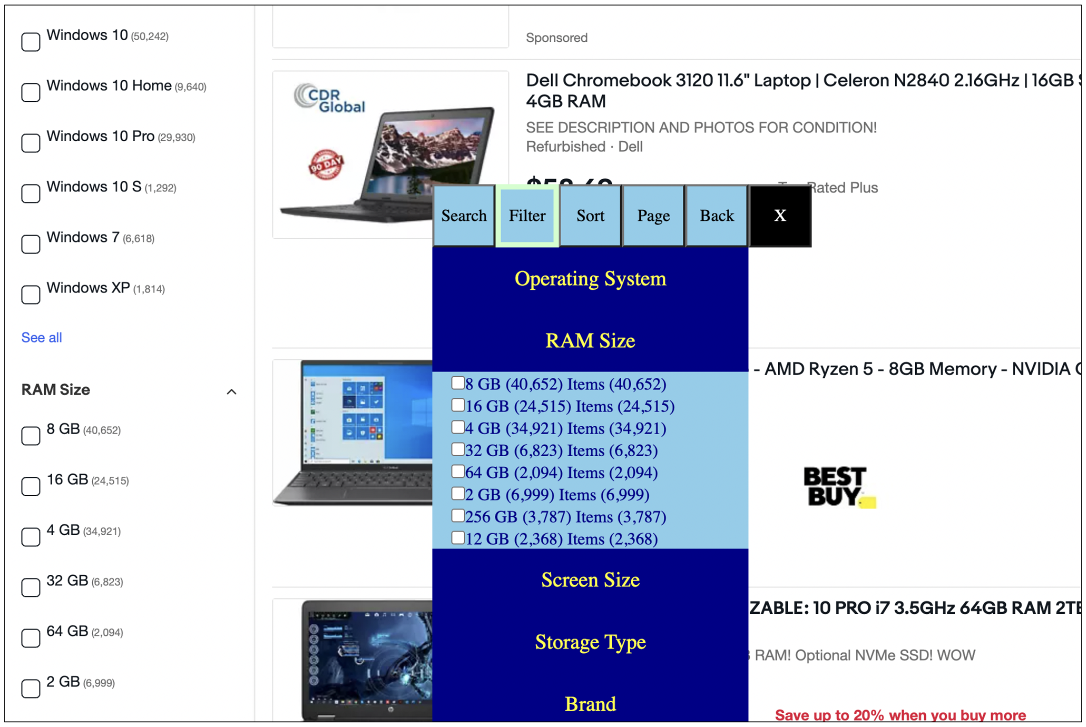

Windows 10 (50,242)
Windows 10 Home (9,640)
Windows 10 Pro (29,930)
Windows 10 S (1,292)
Windows 7 (6,618)
Windows XP (1,814)
See all
RAM Size
8 GB (40,652)
16 GB (24,515)
4 GB (34,921)
32 GB (6,823)
64 GB (2,094)
2 GB (6,999)
Sponsored
Dell Chromebook 3120 11.6" Laptop | Celeron N2840 2.16GHz | 16GB
4GB RAM
SEE DESCRIPTION AND PHOTOS FOR CONDITION!
Refurbished · Dell
Rated Plus
Search
Filter
Sort
Page
Back
X
Operating System
RAM Size
8 GB (40,652) Items (40,652)
16 GB (24,515) Items (24,515)
4 GB (34,921) Items (34,921)
32 GB (6,823) Items (6,823)
64 GB (2,094) Items (2,094)
2 GB (6,999) Items (6,999)
256 GB (3,787) Items (3,787)
12 GB (2,368) Items (2,368)
Screen Size
Storage Type
Brand
- AMD Ryzen 5 - 8GB Memory - NVIDIA
BEST BUY
ZABLE: 10 PRO i7 3.5GHz 64GB RAM 2T
RAM! Optional NVMe SSD! WOW
Save up to 20% when you buy more

Figure 1: InSupport for convenient non-visual interaction with web data records. The auxiliary segments such as filters and sort options are automatically extracted and instantly 'delivered' to a blind user via a proxy interface that is easily navigable using basic screen reader shortcuts such as arrow and TAB keys. Without the proxy interface, the user has to perform multiple key presses to manually navigate between the data records and auxiliary segments – a process that includes traversing realms of irrelevant content.

[8, 14]. Specifically, these existing methods strived to improve interaction efficiency by allowing screen reader users to skip irrelevant content during navigation, i.e., skip directly to the beginning of the records from anywhere in the page [8], skip to the next/previous record irrespective of underlying HTML markup [9], navigate the webpage at a *semantic* level of logically meaningful segments [14], etc. However, these solutions did little to enable convenient and efficient execution of data record specific activities such as filtering, sorting, and searching, which are also essential for quickly locating the desired records, especially when the number of data records is large and distributed across multiple pages.

To overcome these limitations of current approaches, in this paper we present InSupport, a browser extension that automatically extracts auxiliary segments from a webpage containing data records, and then *pushes* these segments to a screen reader user instantly on demand via a proxy interface (see Figure 1) that is easily navigable with basic screen reader shortcuts. To extract the auxiliary segments in the webpage, InSupport uses custom devised machine learning-based detection algorithms that can identify subtrees in the DOM of the webpage that correspond to these segments. With this instant access support from InSupport, blind users will no longer have to spend significant time and effort manually exploring and navigating between the different webpage segments via numerous keyboard shortcuts in order to locate the data records that match their needs or preferences. This is especially beneficial for average and novice blind screen reader users who typically remember only a handful of basic keyboard shortcuts [7]; these users currently spend significantly more time than expert blind users in performing web tasks [7], as they are unaware of advanced shortcuts that enable faster navigation of web content.

A user study with 14 blind participants showed that with InSupport, the average time and the number of input actions required by the participants for locating the desired data records were significantly lower than those with a state-of-the-art solution [9] as well as those with just their preferred screen readers. In sum, our contribution are:

- Machine learning based algorithms to automatically identify auxiliary segments such as filters, sort options, search form, and multi-page links in webpages containing data records.
- A novel browser extension InSupport that enables blind screen reader users to conveniently and instantly access the auxiliary segments (e.g., filters, sort options) while interacting with the web data records.
- The findings of a user study with 14 blind participants who evaluated InSupport and also compared it with a state-of-the-art solution [9] and their preferred screen readers.

2 RELATED WORK

Plethora of prior research works have addressed the interaction issues faced by blind users while browsing the web [8, 11, 17, 26, 30, 32]. While bulk of these works have predominantly focused on the accessibility of web content for blind users [4, 5, 10, 12, 22–24, 29, 32, 34, 37, 39], researchers have also explored the usability of web interaction for blind users [7–9, 11, 17, 20, 26, 30]. For instance, approaches have been proposed to automatically annotate webpages [9, 20] by injecting JavaScript into the DOM of webpages in order to improve their usability. In this regard, Brown et al. [20] presented a JavaScript based method that captured and classified dynamic changes in webpages, and subsequently provided this information to screen reader users via an injected ARIA live region. Similarly, a recent work [9] employed visual saliency-capturing deep neural networks to identify the important parts or 'hot-spots' of a webpage, and then automatically injected ARIA landmark roles into the corresponding DOM subtrees of the identified hot-spots, so as to enable a screen reader user to quickly navigate to these hot-spots with special dedicated screen reader shortcuts (e.g., 'R' in the JAWS screen reader). While some screen readers also provide similar functionalities (e.g., the rotor feature in VoiceOver) for skipping irrelevant content, navigation is nonetheless one-dimensional and dependent on the DOM layout of the webpage; semantics and relative importance of the individual segments are not considered by these screen reader functionalities.

Apart from automatic annotations, researchers have also investigated several other approaches such as web automation [13, 33], speech assistants [8, 21], and alternative input modalities [14]. Web automation techniques [13, 16, 27, 31] facilitate automatic execution of certain repetitive web tasks (e.g., ordering a preferred pizza), thereby significantly reducing the user's manual effort and time for doing these tasks. A common aspect of most automation techniques is the use of task scripts or macros that contain the sequence of actions to execute the corresponding tasks. These macros can either be created through handcrafting [16, 31], or via user demonstration [6, 13, 28, 33]. While these techniques indeed improved interaction experience for blind screen reader users, they were limited to a small set of repetitive tasks, and therefore they would not be able to handle web tasks such as searching for a desired web data record, which involve considerable ad-hoc web browsing. Furthermore, end users face an extra burden of not only creating a script for each web task, but also maintaining and updating the script over time to accommodate changes to either the webpage or their preferences.

Accessibility assistants [7, 8, 21] on the other hand enable blind users to use speech input to interact with webpage content. For instance, Gadde et al. [21] proposed a simple speech-based interface that enabled blind users to use simple voice commands to aurally obtain a quick overview of any webpage and also directly navigate (i.e., shift screen reader focus) to a few key segments. Ashok et al. [8] on the other hand supported a richer set of speech commands, including those to navigate and query web data records. Although speech interfaces facilitate faster access to content, they have several limitations, notably speech recognition accuracy (in noisy environments) [1] and blind users' social concerns (e.g., drawing undesired attention from others) and privacy [3]. Also, many of these assistants (e.g., [8, 21]) require tighter integration with third-party screen reader framework, so their scope is limited to open-source screen readers.

Researchers have also explored novel input modalities to facilitate convenient webpage navigation by overcoming some of the core limitations of the keyboard based 'press-and-listen' mode of screen reader interaction. For example, Billah et al. [14] proposed using an off-the-shelf Dial input device as a surrogate for mouse to hierarchically navigate the semantically-meaningful segments (e.g., menu, forms, data records) on the page using simple rotate and press gestures. Similarly, Soviak et al. [36] presented a new tactile input device that enabled blind users to 'feel' the layout of any webpage via tactile sensations provided at boundaries of the webpage segments. Blind users could also employ this tactile device to navigate webpage content in a 2D space and directly select one of the segments on the page, akin to touch exploration on mobile devices. While these interfaces were effective in improving non-visual interaction with web content, they were limited to general navigation of the webpage semantic structure, and as such did not directly assist in accomplishing 'high-level' specific web tasks such as locating desired data records, which involve activities such as filtering, sorting, and searching. Moreover, these approaches were less adoptable as they require additional hardware, which can be potentially expensive to many blind screen reader users.

To overcome the limitations of current research works, in this paper we present InSupport, a scalable approach for improving the usability of non-visual interaction with web data records.

3 APPROACH

Figure 2 presents an architectural schematic illustrating the workflow of InSupport. As shown in the figure, InSupport was implemented as a browser extension that has two core components: (i) Segment Extractor and (ii) Proxy Interface. The Segment Extractor analyzes the webpage DOM and extracts the auxiliary segments (i.e., filters, sort options, search form, and multi-page links) using custom machine learning-based identification algorithms. The content of identified auxiliary segments is then replicated and presented to a user via the Proxy Interface. The user can instantly access the InSupport Proxy Interface using a special 'CTRL+SHIFT+Z' shortcut, and then navigate the content using TAB or arrow keys. All user selections in the Proxy Interface (e.g., "filter by price", "sort by most recent", "next page") are automatically translated by InSupport into equivalent actions on the actual auxiliary segments on the webpage, thereby achieving the same intended effect.

3.1 Extracting Auxiliary Segments

The Segment Extractor leverages custom identification algorithms to extract the auxiliary segments from the webpages containing data records. Specifically, InSupport extracts the following four types of auxiliary segments: (i) Filter Options; (ii) Sort Options; (iii) Search Form; and (iv) Multi-page Links (see Table 1 in Appendix).

3.1.1 Identification Algorithms. The four algorithms for extracting the four types of auxiliary segments all have a similar workflow. They start by first extracting all the candidate DOM nodes by referring to a predefined list as presented in Table 1. This predefined

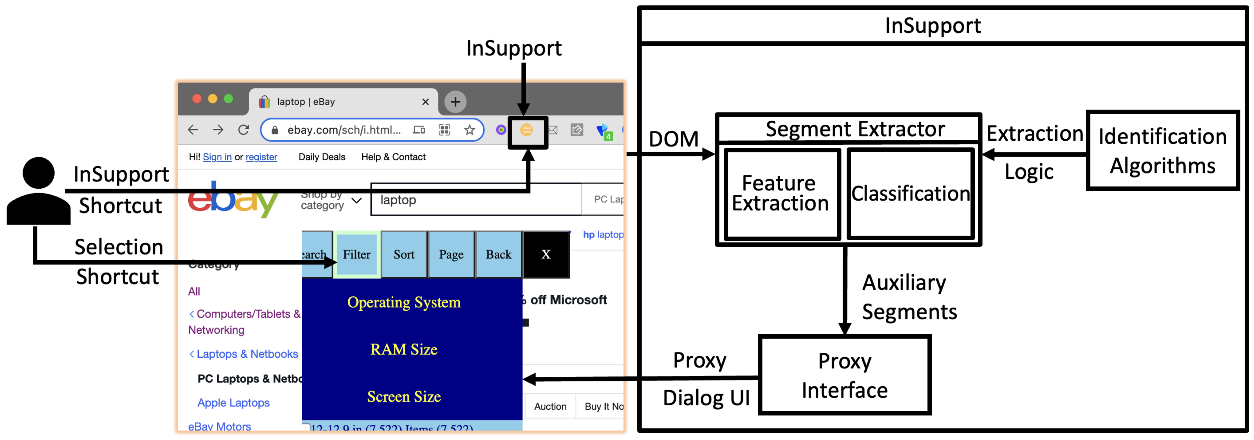


Figure 2: InSupport architectural workflow.

list was compiled after manual analysis of 100 webpages from top-visited websites[1] belonging to different domains such as shopping, travel, job search, and classifieds. Then the algorithms extract a set of handcrafted features (see Table 2 in the Appendix) from the subtrees of all candidates. The extracted features representing each candidate are then fed to the custom trained machine learning models (Table 1 in Appendix) that use classification to determine whether the candidate is the intended auxiliary segment.

Features. The detailed list of features along with their descriptions, for each auxiliary segment type are presented in Table 2 in the Appendix. We handcrafted a unique set of features for each segment type. As seen in Table 2, most features were binary (0 or 1) whereas the remaining features were numerical (integer). We designed several keyword based features (e.g., presence of *search* keyword) because we noticed in our earlier manual analysis that the HTML metadata typically contained several of such keywords within the attribute values of container nodes (e.g., div), even if these words were not shown/rendered on the webpage.

Model evaluation. To evaluate the classification models, we constructed four separate datasets for each of the four types of auxiliary segments. The source for these datasets was a custom built collection of manually-annotated 209 webpages (webpage collection/datasets and details available on Github [2]). These webpages were chosen from a diverse set of websites belonging to over 15 different domains including shopping, travel, finance, and sports (e.g., *bestbuy*, *airbnb*, *shutterstock*, *nba*). Note that these datasets did not overlap with the earlier dataset of 100 websites that was used for manual analysis to determine candidate tags. We annotated each webpage in the collection by manually inserting custom data attributes, one for each type of auxiliary segment; these data attributes were then exploited while constructing the corresponding datasets. Therefore, each webpage in the collection produced 1 positive data point for each type of auxiliary segment, thereby totalling 209 positive data points per auxiliary segment type. As the number of negative data points from each webpage can be much higher than 1, we randomly picked one negative data point per auxiliary type from each webpage in order to have balanced datasets. In sum, for each type of auxiliary segment, we had a total of 418 data points in the corresponding dataset, with equal number of positive/negative points.

In each of these datasets, we randomly picked and set aside 320 data points (160 positive and 160 negative) for training and the remaining 98 for testing. We performed a 5-fold cross validation for two machine learning algorithms - Logistic Regression Classifier and Multi-Layer Perceptron Classifier on the training dataset for optimization. The best models (based on F-score) for each classifier were then evaluated on each corresponding test dataset. The performances of these classifiers for each auxiliary segment type are presented in Table 3 (see Appendix). As noticeable in Table 3, with the handcrafted features, both machine learning algorithms performed very well in discriminating between auxiliary segments and arbitrary webpage segments. The few erroneous classifications were mostly due to unconventional HTML realizations of auxiliary segments in some test webpages. For example, in one such instance, the filter options were implemented as a collection of drop down menus instead of the conventional group of checkboxes or links. Similarly, in another instance, the Sort Options were implemented as independent buttons instead of the traditional drop-down list. In such scenarios, the algorithms were not able to correctly recognize the auxiliary segments, thereby causing a slight drop in the recall performance of these algorithms. The best performing model (based on F-score) for each segment type was then selected for the corresponding identification algorithm.

[1]Alexa Ranking: https://www.alexa.com/topsites
[2]https://github.com/javedulferdous/InSupport

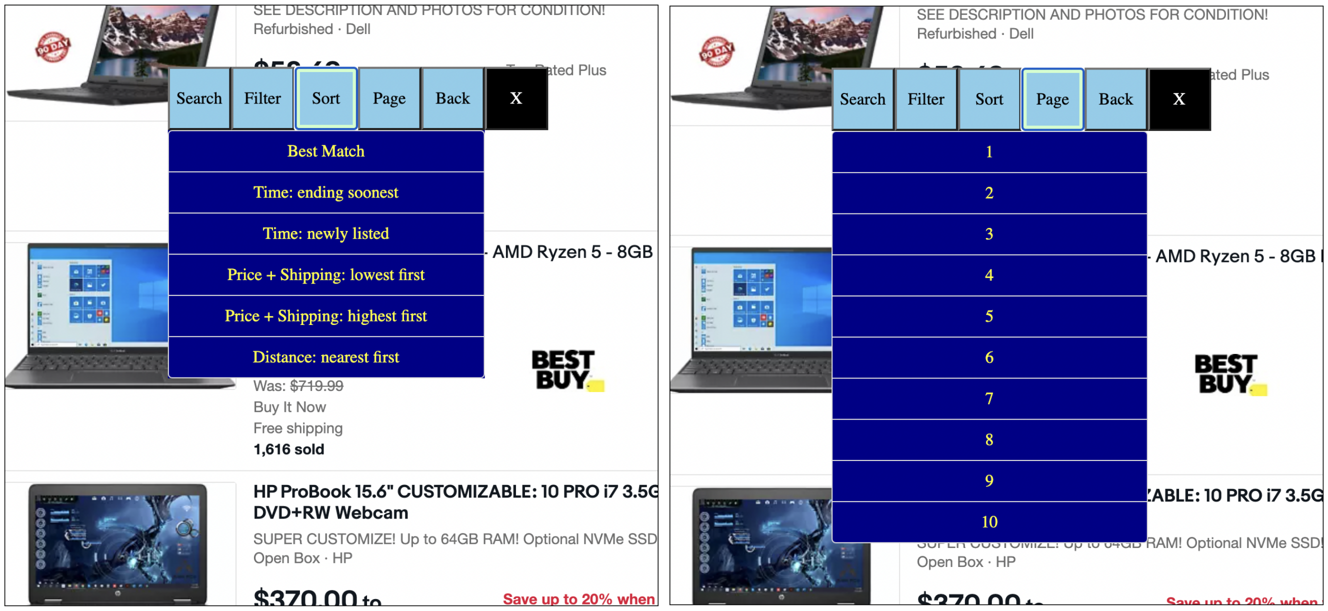


Figure 3: InSupport Proxy interface: (a) Sort Options, (b) Multi-Page Links.

3.2 InSupport Proxy Interface

The proxy interface of InSupport was designed to be navigable with simple TAB and arrow shortcuts. As seen in Figure 3, it is a "one-stop" interface where a user can access all four auxiliary segments. To access the interface, the user needs to press the 'CTRL+SHIFT+Z' shortcut. Initially the focus is set to the first auxiliary segment, namely the *Search* form (e.g., *Search* form in Figure 3). To navigate to other segments, the user can press TAB/SHIFT+TAB or LEFT/RIGHT arrow keys. To select one of the segments, the user has to simply press the ENTER key. Within each segment, the user can navigate the various options using either the TAB/SHIFT+TAB or UP/DOWN arrow keys. To select an option, the user has to again press ENTER. To move focus back to the list of segments, the user can press the ESCAPE key.

When the user selects an option (e.g., "filter by price less than \$50", InSupport automatically translates this action into an equivalent action on the actual auxiliary segment on the webpage, thereby producing the same effect. Furthermore, InSupport refocuses the screen-reader cursor to the beginning of the data records each time they are updated or refreshed, thereby letting the user avoids the burden of pressing numerous screen-reader shortcuts to navigate back to the data records from the top of the webpage (by default, the screen reader starts narrating from the top of a page whenever the page is loaded or refreshed). Also, the use of InSupport to access the auxiliary segments is purely optional; the user can always rely on the standard screen-reader shortcuts to interact directly with the webpage content including the auxiliary segments.

Note that InSupport does not show the tab option for a given auxiliary segment in its proxy interface if the corresponding extraction algorithm is unable to: (i) detect the segment in the webpage, and (ii) detect labels from the extracted segment (possibly due to improper webpage design or due to inaccurate segment identification).

4 EVALUATION

To assess the effectiveness of InSupport's proxy interface, we conducted an IRB approved user study with blind participants. The details of the study and the findings are described next.

4.1 Participants

We recruited 14 blind participants via email lists and snowball sampling. Table 4 in Appendix presents the participants' demographic details. The average age of the participants was 43.57 (Median = 44, Min = 26, Max = 64), and the gender representation was equal (7 female, 7 male). Our inclusion criteria required the participants to be proficient with JAWS screen reader and Chrome web browser. No participant had any motor impairments that affected their ability to do the study tasks. All participants stated that they regularly accessed a wide range of e-commerce websites for doing activities such as shopping, searching for jobs, and browsing classifieds.

4.2 Design

In the study, the participants were asked to perform the following two tasks related to data record interaction:

- T1 – Locate a data record on a travel website that matches a predefined criteria (e.g., morning flight, Delta carrier, price less than \$300).
- T2 – Locate a desired data record on a shopping website based on the participant's own personal preferences.

In a within-subjects experimental setup, the participants were asked to perform the above two tasks under three study conditions:

- Screen reader – The participants could rely only on their preferred screen readers to complete the tasks. This condition represented the status quo for all the participants.

- SaIL [9] – The participants could rely only on their screen readers in this condition too, except that the webpage was preprocessed by a state-of-the-art transcoding technique SaIL [9]. Specifically, SaIL uses visual saliency to detect important regions of a webpage, and then injects ARIA landmarks to the webpage DOM so that screen reader users can access the salient regions via special shortcut (e.g., 'R' in JAWS screen reader).
- InSupport – The participants could not only interact with webpages directly via screen reader shortcuts, but also instantly access the four auxiliary segments (i.e., filter options, sort options, search form, and multi-page links) via the InSupport interface.

To mitigate learning effect and avoid confounds, we used three different travel websites (Kayak, Travelocity, and Orbitz) for T1, and also three different shopping websites (Amazon, eBay, and Target) for T2. Furthermore, for T2, we also chose three different but similar query items ('laptop', 'desktop', and 'tablet'). For T1, the target data record was located in the second data records webpage (between fifth and eighth positions) on all three chosen websites. Also, in all the websites, the SaIL model [9] was able to identify all four auxiliary segments as being salient, and therefore these segments had corresponding aria landmarks injected into them for the SaIL study condition. We also ensured that InSupport too was able to accurately extract the auxiliary segments so as to prevent the confounding impact of extraction algorithm accuracy on InSupport user interface evaluation. The assignment of websites to conditions, items to websites, and the ordering of both tasks and conditions were counterbalanced using the Latin square method [18].

4.3 Apparatus

The experiment was conducted remotely, and the participants used their own computers – either laptop or desktop ('Computer Type' column in Table 4). All participants had JAWS screen reader and Google Chrome web browser installed on their computers. The InSupport extension was sent to the participants via email (as a Google Drive link) just prior to the study, and the experimenter also assisted the participants (via Zoom or Skype conferencing software) in installing the extension onto their Chrome browser. Four participants (P3, P8, P11, and P14) needed assistance from their cohabiting family members or friends to install InSupport. Note that for convenience, both the SaIL and InSupport systems were included in the single extension that was emailed to the participants. A special shortcut was provided to turn 'ON/OFF' each condition, and only one condition could be turned on at any given time. Specifically, if the SaIL condition was turned 'ON', the InSupport condition was automatically turned 'OFF', and vice versa.

4.4 Procedure

The experimenter first assisted the participant in downloading and installing the InSupport extension. Next, the experimenter gave the participant enough time to practice (~ 20 minutes) so as to make them familiar and comfortable with the study conditions. The participant was then asked to complete all the tasks under different study conditions in the predetermined counterbalanced order. For each task, the experimenter allowed a maximum of 20 minutes for the participant to complete the task. The study lasted for a maximum of 3 hours, and all conversations were in English. After completing the tasks, the participant was asked to respond to subjective questionnaires (System Usability Scale (SUS) [19] for measuring usability and NASA Task Load Index (NASA-TLX) [25] to measure perceived user workload), and also participate in an exit interview to collect suggestions and other qualitative feedback. Throughout the study, the screen-sharing and recording features were turned on so as to capture (with the participant's permission) all user interaction activities for subsequent data analysis.

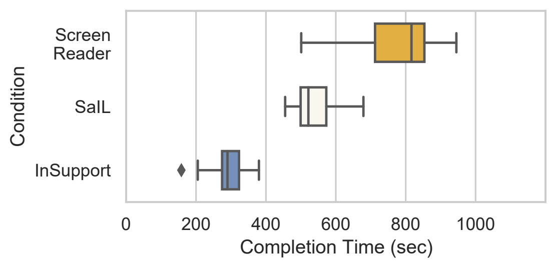
Measurements. During the study, the experimenter recorded task completion times and the number of user actions for each task performed by the participant. The experimenter also recorded the responses to previously mentioned System Usability Scale (SUS) [19] and NASA Task Load Index (NASA-TLX) [25] questionnaires. Qualitative feedback and peculiar interaction behavior during the study were also noted by the experimenter. We used grounding theory (open coding technique [35]) for analyzing the transcribed qualitative feedback from the participants. We iteratively went over the user responses and identified key recurring themes or insights in the data.

4.5 Results

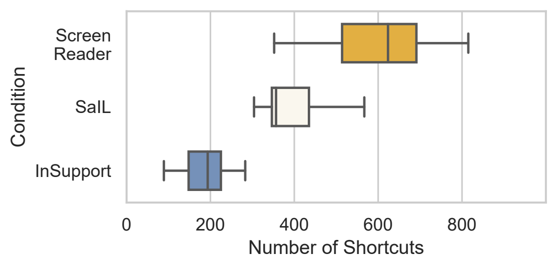
4.5.1 Task T1 - Travel. **Task completion time.** Figure 4a presents the results for task completion times of the participants under all three study conditions for Task T1. Overall, in the screen reader condition, the participants spent an average of 780.64 seconds (Median = 816.5, Min = 501, Max = 945), whereas they spent an average of 540.85 seconds (Median = 521.5, Min = 455, Max = 679) with SaIL, and 288.64 seconds (Median = 290, Min = 158, Max = 380) with InSupport. A Kruskal-Wallis test showed that the difference in task completion times between the three study conditions was statistically significant (see Table 5 in Appendix).

Number of user actions. Figure 4b shows the statistics regarding the number of input actions performed by the participants under the three study conditions. In the screen reader condition, the participants needed an average of 611.85 input actions (Median = 623.5, Min = 352, Max = 815) to complete the task, whereas in the SaIL condition they needed an average of 395.92 input actions (Median = 356.5, Min = 304, Max = 567) to finish the task. However, in the InSupport condition, the participants performed significantly fewer input actions – an average of 187.21 input actions (Median = 193.5, Min = 89, Max = 283) to complete the task. As in case of task completion times, we observed a statistically significant difference between the number of user actions for the three study conditions (Table 5 in Appendix).

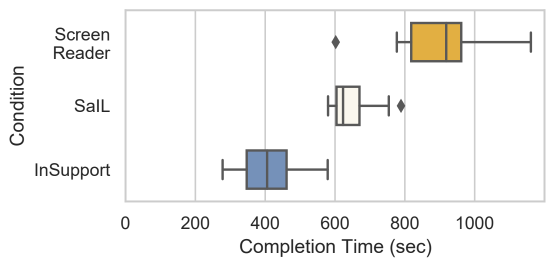
An analysis of the study data revealed the underlying reasons for the observed difference in task completion times and the number of actions between conditions for Task T1. In the screen reader condition, most users (12) exhibited the following two types of interaction behavior: (i) navigate the data records one-by-one while accessing only the multi-page links auxiliary segment (7 participants); and (ii) navigate back-and-forth between data records and the filters segment, specifically, repeat the process of navigating the first few records one-by-one and then go back to selecting filters, until the target data record is found (5 participants). These two



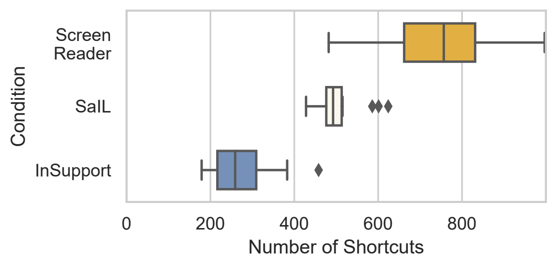
(a) T1 completion time



(b) T1 number of shortcuts



(c) T2 completion time



(d) T2 number of shortcuts

Figure 4: User performance statistics for the two study tasks T1 and T2.

types of behavior significantly contributed to the increase in completion time and number of actions as the participants traversed a vast amount of DOM content in the process. Only the two remaining participants decided to set all the desired filters first and then serially navigate the data records to locate the target record. Even in the SaIL condition, despite practice, slightly over one-third of the participants (5) did not use the special shortcut for quickly navigating between different landmarks injected by SaIL that covered both the data records and the filters segment. Instead, they chose to navigate over the data records one-by-one as in case of the screen reader condition, which considerably increased the time and input actions overhead. Even in case of the remaining 9 participants who used special SaIL shortcuts, they still had to navigate over quite a few irrelevant segments that were present between the filter options and the data records, since these irrelevant webpage segments too were landmarked by the SaIL saliency model. This problem did not manifest in the InSupport condition as the participants could directly access the auxiliary segments including filter options.

4.5.2 Task T2 - Shopping. **Task completion time.** Figure 4c presents the results for task completion times under all three study conditions for Task T2. Overall, in the screen reader condition, the participants spent an average of 894.35 seconds (Median = 918.5, Min = 602, Max = 1161), whereas they took an average of 646.14 seconds (Median = 623, Min = 580, Max = 789) with SaIL, and 413.64 seconds (Median = 405.5, Min = 278, Max = 579) with InSupport. Similar to Task T1, a Kruskal-Wallis test showed that the difference in task completion times between the three study conditions was statistically significant (see Table 5 in Appendix). All participants were able to successfully complete the task in all conditions.

Number of user actions. Figure 4d presents the statistics regarding the number of input actions performed by the participants in T2 under the three study conditions. In the screen reader condition, the participants needed an average of 751.64 input actions (Median = 756.5, Min = 482, Max = 997) to complete the task, whereas in the SaIL condition, they needed an average of 506.85 input actions (Median = 492.5, Min = 428, Max = 624) to finish the task. However, in the InSupport condition, the participants only performed an average of 276.64 input actions (Median = 259, Min = 179, Max = 458) to finish the task. This difference in the number of user actions between the three study conditions was statistically significant (see Table 5 in Appendix).

In contrast to Task T1 where the participants were focused on locating a pre-specified target record, in Task T2 they exhibited a more exploratory navigational behavior. In the screen reader condition, all participants on at least one occasion navigated to-and-fro between the data records and the filters segment. However, each back-and-forth added significant time and effort overhead given the large of number of DOM elements they had to traverse while navigating between these segments. A majority (9) of the participants exhibited a similar behavior in the SaIL condition, however, the time and effort were considerably reduced due to the advantage of special landmark shortcuts. However, these participants still had to navigate over extraneous segments to go back-and-forth between data records and filters segment. The remaining 5 participants did not use the landmark shortcut and instead performed the task just like how they did in the screen reader condition. In the exit interviews, these participants stated that they forgot the special landmark shortcut (despite practice) and that they were hesitant to try out their guesses for the fear of losing context in the task webpage. In the InSupport condition, all participants were more liberal in their use of filters via the InSupport proxy interface – all participants accessed the filters at least three times while doing the tasks. As the InSupport interface was instantly accessible, the participants did not expend any time or effort accessing the filters. Instead, most of their time and effort (i.e., input actions) were dedicated towards navigating the data records and also the linear list of filters in the proxy interface.

4.5.3 Usability. As mentioned earlier, we administered the standard SUS questionnaire to measure usability [19]. The SUS questionnaire requires the participants to rate alternating positive and negative statements about each study condition on a Likert scale from 1 - strongly disagree to 5 - strongly agree, with 3 - neutral. These responses are then assimilated into a score between 0 to 100, with higher values indicating better usability ratings. Overall, the SUS ratings for the InSupport condition were much higher (μ = 86.07, σ = 6.38) compared to those for the screen reader (μ = 53.92, σ = 13.61) and SaIL (μ = 68.92, σ = 14.66) conditions. A one-way ANOVA test revealed that this difference in SUS scores was statistically significant ($F = 22.865, p < 0.0001$). In the exit interviews, most (12) participants attributed their high ratings to the instant interface access feature of InSupport that saved multitude of key presses which were otherwise necessary to navigate between webpage segments. 8 participants also mentioned that the short learning curve of InSupport motivated them to provide favorable usability ratings for InSupport.

4.5.4 Perceived Workload. For workload estimation, we administered the NASA-TLX questionnaire to the participants [25]. The TLX questionnaire measures perceived task workload as a value between 0 and 100, with lower values indicating lower workloads, and hence better results. Overall, the TLX scores were significantly better for InSupport (μ = 25.61, σ = 6.71) than those for screen reader (μ = 74.38, σ = 4.93) and SaIL (μ = 45.57, σ = 6.76). As in case of SUS score, the difference in TLX scores between the three study conditions was statistically significant (one-way ANOVA, $F = 203.401, p < 0.0001$). The individual subscales of TLX that contributed the most towards the high total workloads in the screen reader condition were temporal demand, effort, and frustration, i.e., the ratings for these subscales were significantly higher than those for the other subscales (mental demand, physical demand, and overall performance). Effort and frustration subscales were also the major contributors to the workloads in the SaIL condition. For the InSupport condition, however, the ratings were much lower and uniform across all subscales with no obvious patterns.

4.5.5 Qualitative Feedback. In addition to the responses to the questionnaires, the participants also provided subjective feedback in their exit interviews that also included suggestions for improvement and feature requests. Some of the notable recurring themes identified from the exit interview data are mentioned next.

Separate interface for auxiliary segments is important. Almost all (12) participants stated that having a separate proxy interface for accessing the auxiliary segments was important because it helped them "separate their concerns", i.e., use screen reader shortcuts only for navigating within the data records and not worry about how to navigate to the auxiliary segments. As quoted by P4, "I only have to remember the layout of content in each item of the results, and not the entire webpage." In fact, a few (3) participants even suggested providing a different input method such as speech to access the InSupport proxy interface, so as to completely disentangle InSupport from the screen reader keyboard shortcuts.

Mitigating webpage exploration reduces frustration and leads to better record selection. All participants stated much of the frustration and fatigue during web browsing stems from the tedious serial exploration of webpage content using keyboard shortcuts, and therefore they typically cannot explore many data records before their selection. A majority (11) of the participants further stated that due to limited exploration caused by fatigue, they often miss out on the "best deals". These participants expressed that as fatigue and frustration are significantly lower with InSupport, they can explore more data records and therefore take advantage of better deals. As quoted by P7, "More coverage means more options, and more options means more likelihood of finding a better product".

Remembering and reusing past filters can increase efficiency of data record interaction. More than one third (5) of participants expressed a desire for remembering the past selection of filters and then automatically applying in future interactions involving the same or similar data records. For example, P7 suggested that InSupport should be able to remember his flight preferences based on prior interaction data and then automatically apply these filters every time he searches for flights, not only on the same website but only on other travel websites. These participants indicated that such a feature would significantly reduce their interaction burden as the list of filters itself can sometimes be very long.

All data records on one single page is preferable. 7 participants mentioned that they would like to have all data records on single webpage, so that they did not have to rely on multi-page links to go over multiple webpages. The main reason given by these participants was that every new page load in the browser tends to refocus the screen reader cursor to the top of the page, and sometimes it is tedious and cumbersome to navigate to the data records from the top of the page. These participants desired the InSupport to be capable of prefetching all the data records and appending them to the list on the first webpage.

5 DISCUSSION

The user study demonstrated the efficacy of InSupport in significantly improving the interaction experience of blind screen reader users with web data records. However, it also highlighted certain limitations of InSupport as well as promising directions for future work. Some of the notable ones are discussed next.

Limitations. A limitation of our work is that the evaluation of segment extraction algorithms was performed on a small sample of components extracted from webpages instead of arbitrary webpages themselves "in the wild". Our user study too was limited to evaluating the InSupport's proxy interface on webpages where the extraction algorithms could accurately identify all the auxiliary segments, and as such we did not consider webpages where one or more of the extraction algorithms had errors. Further elaborate validation of the algorithms and InSupport interface is thus required to determine the extent to which our findings generalize across arbitrary webpages having different kinds of content layouts and designs. Moreover, the extraction algorithms were designed exclusively for webpages in English, and therefore need to be extended with additional language-agnostic features to be able to accurately identify auxiliary segments in webpages written in other languages.

Another limitation of our work is that we evaluated InSupport only with JAWS screen reader users. Apart from JAWS, a recent survey also found that a larger proportion (59.9%) of blind users rely on other screen readers including NVDA and VoiceOver [38]. While our observations are very likely to generalize across different screen readers due to the similarities in how they support web browsing, a formal study to compare the interaction experiences between user groups relying on different screen readers can shed light on the effectiveness of InSupport both within and between groups. Also, the current InSupport is only supported for the Chrome web browser. While Chrome browser is currently the most widely used browser within the blind user community, there are still significant proportions of blind users relying on other browsers such as Firefox and Internet Explorer [38]. In future, we will explore options to extend InSupport support for other browsers. Lastly, the current InSupport prototype is limited to desktop/laptop web browsing, and does not yet support mobile web browsing. As smartphones are becoming ubiquitous and users are increasingly interacting with web data records on smartphone browsers, e.g., online shopping, support for convenient non-visual interaction with data records is especially important for blind users, given that smartphone screen readers offer far fewer input gestures compared to the plethora of keyboard shortcuts offered by desktop screen readers. Recognizing this emerging need, we will explore options in the future to port InSupport for smartphone web browsers.

Automatic filter selection and reordering. As mentioned by the participants in the exit interviews, sometimes the list of filters can be very long, and therefore navigating this list can be tedious and cumbersome even with InSupport. Therefore, mechanisms are needed that can automatically determine the set of filters that the user will mostly likely apply, given the user's past interaction history. By leveraging these mechanisms, InSupport will be able to proactively either apply the desired filters on user's behalf or suggest them to the user. InSupport will also be able to dynamically reorder the filters in the proxy interface so that the filters with high likelihood of being selected are placed near the front of the list. Exploring such solutions is scope of future research.

Prefetching data records. In the study, some participants also expressed desire to have all the data records on one single page. While some website do provide an option to choose the number of records per page, many websites do not offer this feature. Also, prefetching a large number of data records from possibly a multitude of webpages can be challenging due to the significant time overhead that may potentially render the prefetching method impractical for real-time interaction. Exploring feasible alternative approaches to address this issue under such challenging constraints will also be part of our future work.

Societal impact. Accessibility of webpages is important to ensure equality of access to digital content for people with disability, including those with severe visual impairments. However, accessibility in and of itself is not equivalent to usability, which is more concerned with how easily people can accomplish their tasks on the web. As most websites are primarily designed for convenient sighted interaction, blind screen reader users have to expend significantly more time and effort to do the same web tasks compared to those exerted by their sighted peers [8], thereby creating a usability gap in the interaction experience. This paper seeks to narrow this gap for one of the important everyday web tasks – interaction with web data records. By facilitating more convenient interaction with web data records, blind screen reader users too will be able to find 'better deals', complete transactions faster on e-commerce websites, read more posts from their friends with less effort, and so on.

6 CONCLUSION

Interaction with web data records is an important and ubiquitous activity in web browsing. The present interface design for data records however is primarily tailored for sighted interaction and therefore blind screen reader users struggle to locate desired data records with the same ease and efficiency as their sighted peers. To reduce this usability gap between sighted and blind users, this paper presented InSupport, a browser extension that automatically extracts the important auxiliary segments such as filters and multi-page links from webpages containing data records, and subsequently makes them instantly accessible to blind screen reader users via an easily navigable proxy interface. Evaluation of InSupport in a user study with blind participants showed that InSupport significantly reduced the interaction effort when compared with a state-of-the-art solution and also the participants' preferred screen readers, thereby demonstrating the efficacy of InSupport in aiding non-visual interaction with web data records. The study also revealed promising avenues for future research that included automatic selection of record filters that a user will most likely apply for a given set of data records.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their insightful feedback. This research work was funded by NSF awards 1805076, 1936027, 2113485, and NIH awards R01EY030085, R01HD097188.

REFERENCES

[1] Ali Abdolrahmani, Ravi Kuber, and Stacy M Branham. 2018. " Siri Talks at You" An Empirical Investigation of Voice-Activated Personal Assistant (VAPA) Usage by Individuals Who Are Blind. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*. 249–258.

[2] Tousif Ahmed, Roberto Hoyle, Kay Connelly, David Crandall, and Apu Kapadia. 2015. Privacy Concerns and Behaviors of People with Visual Impairments. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3523–3532. https://doi.org/10.1145/2702123.2702334

[3] Tousif Ahmed, Patrick Shaffer, Kay Connelly, David Crandall, and Apu Kapadia. 2016. Addressing physical safety, security, and privacy for people with visual impairments. In *Twelfth Symposium on Usable Privacy and Security ({SOUPS} 2016)*. 341–354.

[4] Chieko Asakawa and Hironobu Takagi. 2000. Annotation-based transcoding for nonvisual web access. In *Proceedings of the fourth international ACM conference on Assistive technologies*. 172–179.

[5] Chieko Asakawa, Hironobu Takagi, Shuichi Ino, and Tohru Ifukube. 2002. Auditory and tactile interfaces for representing the visual effects on the web. In *Proceedings of the fifth international ACM conference on Assistive technologies*. 65–72.

[6] Vikas Ashok, Syed Masum Billah, Yevgen Borodin, and IV Ramakrishnan. 2019. Auto-Suggesting Browsing Actions for Personalized Web Screen Reading. In

Proceedings of the 27th ACM Conference on User Modeling, Adaptation and Personalization (Larnaca, Cyprus) *(UMAP '19)*. Association for Computing Machinery, New York, NY, USA, 252–260. https://doi.org/10.1145/3320435.3320460

[7] Vikas Ashok, Yevgen Borodin, Yury Puzis, and IV Ramakrishnan. 2015. Capti-speak: a speech-enabled web screen reader. In *Proceedings of the 12th International Web for All Conference*. 1–10.

[8] Vikas Ashok, Yury Puzis, Yevgen Borodin, and IV Ramakrishnan. 2017. Web screen reading automation assistance using semantic abstraction. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*. 407–418.

[9] Ali Selman Aydin, Shirin Feiz, Vikas Ashok, and IV Ramakrishnan. 2020. SaIL: saliency-driven injection of ARIA landmarks. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*. 111–115.

[10] Sean Bechhofer, Simon Harper, and Darren Lunn. 2006. Sadie: Semantic annotation for accessibility. In *International Semantic Web Conference*. Springer, 101–115.

[11] Jeffrey P Bigham, Jeremy T Brudvik, and Bernie Zhang. 2010. Accessibility by demonstration: enabling end users to guide developers to web accessibility solutions. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*. 35–42.

[12] Jeffrey P Bigham and Richard E Ladner. 2007. Accessmonkey: a collaborative scripting framework for web users and developers. In *Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*. 25–34.

[13] Jeffrey P. Bigham, Tessa Lau, and Jeffrey Nichols. 2009. Trailblazer: Enabling Blind Users to Blaze Trails through the Web. In *Proceedings of the 14th International Conference on Intelligent User Interfaces* (Sanibel Island, Florida, USA) *(IUI '09)*. Association for Computing Machinery, New York, NY, USA, 177–186. https://doi.org/10.1145/1502650.1502677

[14] Syed Masum Billah, Vikas Ashok, Donald E. Porter, and IV Ramakrishnan. 2017. Speed-Dial: A Surrogate Mouse for Non-Visual Web Browsing. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (Baltimore, Maryland, USA) *(ASSETS '17)*. Association for Computing Machinery, New York, NY, USA, 110–119. https://doi.org/10.1145/3132525.3132531

[15] Syed Masum Billah, Vikas Ashok, Donald E. Porter, and I.V. Ramakrishnan. 2017. Ubiquitous Accessibility for People with Visual Impairments: Are We There Yet?. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 5862–5868. https://doi.org/10.1145/3025453.3025731

[16] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C Miller. 2005. Automation and customization of rendered web pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*. 163–172.

[17] Yevgen Borodin, Jeffrey P. Bigham, Glenn Dausch, and I. V. Ramakrishnan. 2010. 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)* (Raleigh, North Carolina) *(W4A '10)*. ACM, New York, NY, USA, Article 13, 10 pages. https://doi.org/10.1145/1805986.1806005

[18] James V. Bradley. 1958. Complete Counterbalancing of Immediate Sequential Effects in a Latin Square Design. *J. Amer. Statist. Assoc.* 53, 282 (1958), 525–528. https://doi.org/10.1080/01621459.1958.10501456 arXiv:https://amstat.tandfonline.com/doi/pdf/10.1080/01621459.1958.10501456

[19] John Brooke et al. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.

[20] Andy Brown and Simon Harper. 2013. Dynamic injection of WAI-ARIA into web content. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*. 1–4.

[21] Prathik Gadde and Davide Bolchini. 2014. From screen reading to aural glancing: towards instant access to key page sections. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility*. 67–74.

[22] Cole Gleason, Amy Pavel, Himalini Gururaj, Kris Kitani, and Jeffrey P Bigham. 2020. Making GIFs Accessible.. In *ASSETS*. 24–1.

[23] Cole Gleason, Amy Pavel, Emma McCamey, Christina Low, Patrick Carrington, Kris M. Kitani, and Jeffrey P. Bigham. 2020. Twitter A11y: A Browser Extension to Make Twitter Images Accessible. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376728

[24] Darren Guinness, Edward Cutrell, and Meredith Ringel Morris. 2018. Caption crawler: Enabling reusable alternative text descriptions using reverse image search. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–11.

[25] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. North-Holland, 139 – 183. https://doi.org/10.1016/S0166-4115(08)62386-9

[26] Jonathan Lazar, Aaron Allen, Jason Kleinman, and Chris Malarkey. 2007. What frustrates screen reader users on the web: A study of 100 blind users. *International Journal of human-computer interaction* 22, 3 (2007), 247–269.

[27] Gilly Leshed, Eben M Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1719–1728.

[28] Ian Li, Jeffrey Nichols, Tessa Lau, Clemens Drews, and Allen Cypher. 2010. Here's what i did: sharing and reusing web activity with ActionShot. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 723–732.

[29] Letresa McLawhorn. 2001. Leveling the accessibility playing field: Section 508 of the Rehabilitation Act.

[30] Valentyn Melnyk, Vikas Ashok, Yury Puzis, Andrii Soviak, Yevgen Borodin, and IV Ramakrishnan. 2014. Widget classification with applications to web accessibility. In *International Conference on Web Engineering*. Springer, 341–358.

[31] Paula Montoto, Alberto Pan, Juan Raposo, Fernando Bellas, and Javier López. 2009. Automating navigation sequences in AJAX websites. In *International Conference on Web Engineering*. Springer, 166–180.

[32] Christopher Power, André Freire, Helen Petrie, and David Swallow. 2012. 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* (Austin, Texas, USA) *(CHI '12)*. ACM, New York, NY, USA, 433–442. https://doi.org/10.1145/2207676.2207736

[33] Yury Puzis, Yevgen Borodin, Rami Puzis, and IV Ramakrishnan. 2013. Predictive web automation assistant for people with vision impairments. In *Proceedings of the 22nd international conference on World Wide Web*. 1031–1040.

[34] Yury Puzis, Yevgen Borodin, Andrii Soviak, Valentyn Melnyk, and IV Ramakrishnan. 2015. Affordable web accessibility: A case for cheaper ARIA. In *Proceedings of the 12th International Web for All Conference*. 1–4.

[35] Johnny Saldaña. 2021. *The coding manual for qualitative researchers*. sage.

[36] Andrii Soviak. 2015. Haptic Gloves Prototype for Audio-Tactile Web Browsing. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility* (Lisbon, Portugal) *(ASSETS '15)*. Association for Computing Machinery, New York, NY, USA, 363–364. https://doi.org/10.1145/2700648.2811329

[37] W3C. 2018. Web Content Accessibility Guidelines (WCAG) Overview. https://www.w3.org/WAI/standards-guidelines/wcag/.

[38] WebAIM. 2019. WebAIM: Screen Reader User Survey #8 Results. https://webaim.org/projects/screenreadersurvey8/

[39] Shaomei Wu, Jeffrey Wieland, Omid Farivar, and Julie Schiller. 2017. Automatic alt-text: Computer-generated image descriptions for blind users on a social network service. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. 1180–1192.

[40] Hanlu Ye, Meethu Malu, Uran Oh, and Leah Findlater. 2014. Current and Future Mobile and Wearable Device Use by People with Visual Impairments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 3123–3132. https://doi.org/10.1145/2556288.2557085

A DETAILS OF EXTRACTION ALGORITHMS

Auxiliary Segment	**Candidate Tags**	**# Features**	**Classification Model**
Filter options	div, li, section, article, dt, desktop-facet, ul, form, fieldset, button, dl	5	MLP classifier
Sort options	div, ul, select	4	MLP classifier
Search form	form	5	Logistic regression
Multi-page links	div, nav, li, ul, span, section, button, tr, footer, aPage, pagination, bPage	6	Logistic regression

Table 1: Extraction algorithm details for the auxiliary segments.

B FEATURES FOR AUXILIARY SEGMENT CLASSIFIERS

Search Form	
Feature	***Description***
Inner text present	Checks whether the candidate node has inner text in its subtree - binary
Search keyword	Checks for the keyword "search" in the inner text of candidate's subtree - binary
no. of search keyword	Number of "search" keyword matches with in the candidate's subtree - integer
Button present	Check whether the candidate has a button in its subtree - binary
Search attribute value	Checks if any attribute in any node within the candidate's subtree has "search" keyword in it - binary
Filter Options	
Feature	***Description***
Checkbox List	Checks if the candidate's subtree contains a list of check boxes - binary
Number of links	Counts the number of links in the candidate's subtree - integer
Number of inputs	Counts the number of input tags in the candidate's subtree - integer
URL valid	Check if all the links in the candidate's subtree contain valid URLs - binary
Button list	Checks if the candidate's subtree contains a list of buttons - binary
Sort Options	
Feature	***Description***
Keyword match	Checks inner text of nodes in candidate's subtree for keywords such as *price*, *recommended*, *ratings*, *distance*, *time*, and so on - binary
Keyword count	Counts the number of keyword matches using the same keywords as the previous option - integer
Sort keyword	Check if inner text values of nodes in candidate's subtrees have the keyword *sort* - binary
Option tag count	Counts the number of option tags (if any) in the candidate's subtree - integer
Multi-Page Links	
Feature	***Description***
Number of buttons	Counts the number of buttons in the candidate's subtree - integer
Number of links	Counts the number of links in the candidate's subtree - integer
Common URL	Counts the number of links that have the same domain and subdomain URL - integer
Number of values	Counts the number of nodes in the subtree that have only numeric text such as 1, 2, and 3 - Integer
Keyword present	Checks if inner text of subtree nodes contains keywords such as *page*, *show*, *next*, *previous*, and so on - binary
Keyword count	Counts the number of occurrences of select keywords in the candidate's subtree - integer

Table 2: Features for each auxiliary segment, along with their descriptions.

C CLASSIFICATION PERFORMANCE OF EXTRACTION ALGORITHMS FOR AUXILIARY SEGMENTS

Segment Type	Classifier	Precision (%)		Recall (%)		F-score (%)	
		Negative	Positive	Negative	Positive	Negative	Positive
Search Form	Logistic Regression	90.4	94.5	94.7	90.0	92.5	92.2
	Multi-Layer Perceptron	90.7	93.2	93.3	90.4	93.5	91.7
Multi-page Links	Logistic Regression	83.4	100	100	79.9	90.9	88.8
	Multi-Layer Perceptron	83.4	100	100	79.9	90.9	88.8
Sort Options	Logistic Regression	91.9	100	100	90.8	95.7	95.1
	Multi-Layer Perceptron	92.3	100	100	91.3	95.9	95.4
Filter Options	Logistic Regression	99	100	100	99	99.5	99.5
	Multi-Layer Perceptron	99	100	100	99	99.5	99.5

Table 3: Classification performance of machine learning algorithms.

D PARTICIPANT DEMOGRAPHIC DETAILS

ID	Age	Gender	Age of Vision Loss	Preferred Screen Reader	Hours Per Day	Computer Type
P1	39	M	Since birth	JAWS	5-6	Laptop
P2	26	M	Age 3	JAWS	5-6	Laptop
P3	52	F	Age 5	JAWS	2-3	Laptop
P4	45	M	Age 6	JAWS	3-4	Desktop
P5	34	F	Age 11	JAWS	5-6	Laptop
P6	48	F	Cannot remember	JAWS	4-5	Laptop
P7	34	M	Cannot remember	JAWS	3-4	Desktop
P8	54	F	Cannot remember	JAWS	1-2	Laptop
P9	57	F	Since birth	JAWS	2-3	Desktop
P10	28	M	Age 2	JAWS	5-6	Desktop
P11	64	F	Since birth	JAWS	1-2	Desktop
P12	51	F	Since birth	JAWS	3-4	Laptop
P13	35	M	Since birth	JAWS	5-6	Desktop
P14	43	M	Since birth	JAWS	3-4	Laptop

Table 4: Participant demographics for the user study. All information was self-reported by the participants. Hours per day indicates the average time a participant spent per day on web browsing.

E SIGNIFICANCE TEST RESULTS FOR USER STUDY

Task	**Completion Time**	**Number of User Actions**
Task T1	$H = 34.207$, $df = 2$, $p < 0.001$	$H = 34.465$, $df = 2$, $p < 0.001$
Task T2	$H = 34.369$, $df = 2$, $p < 0.001$	$H = 33.576$, $df = 2$, $p < 0.001$

Table 5: Kruskal-Wallis test for statistical significance between study conditions.