All in One Place: Ensuring Usable Access to Online Shopping Items for Blind Users

YASH PRAKASH, Old Dominion University, USA
AKSHAY KOLGAR NAYAK, Old Dominion University, USA
MOHAN SUNKARA, Old Dominion University, USA
SAMPATH JAYARATHNA, Old Dominion University, USA
HAE-NA LEE, Michigan State University, USA
VIKAS ASHOK, Old Dominion University, USA

Perusing web data items such as shopping products is a core online user activity. To prevent information overload, the content associated with data items is typically dispersed across multiple webpage sections over multiple web pages. However, such content distribution manifests an unintended side effect of significantly increasing the interaction burden for blind users, since navigating to-and-fro between different sections in different pages is tedious and cumbersome with their screen readers. While existing works have proposed methods for the context of a single webpage, solutions enabling usable access to content distributed across multiple webpages are few and far between. In this paper, we present InstaFetch, a browser extension that dynamically generates an alternative screen reader-friendly user interface in real-time, which blind users can leverage to almost instantly access different item-related information such as description, full specification, and user reviews, all in one place, without having to tediously navigate to different sections in different webpages. Moreover, InstaFetch also supports natural language queries about any item, a feature blind users can exploit to quickly obtain desired information, thereby avoiding manually trudging through reams of text. In a study with 14 blind users, we observed that the participants needed significantly lesser time to peruse data items with InstaFetch, than with a state-of-the-art solution.

CCS Concepts: • **Human-centered computing** → **Accessibility technologies**; **Empirical studies in accessibility**.

Additional Key Words and Phrases: Web usability, Online shopping, Blind, Visual impairment, Screen reader

ACM Reference Format:
Yash Prakash, Akshay Kolgar Nayak, Mohan Sunkara, Sampath Jayarathna, Hae-Na Lee, and Vikas Ashok. 2024. All in One Place: Ensuring Usable Access to Online Shopping Items for Blind Users. *Proc. ACM Hum.-Comput. Interact.* 8, EICS, Article 257 (June 2024), 25 pages. https://doi.org/10.1145/3664639

1 INTRODUCTION

Web data items on e-commerce websites are commonplace on the internet. Prominent e-commerce platforms like Amazon offer customers a wide array of millions of products for purchase through

Authors' Contact Information: Yash Prakash, Old Dominion University, Department of Computer Science, Norfolk, Virginia, USA, yprak001@odu.edu; Akshay Kolgar Nayak, Old Dominion University, Department of Computer Science, Norfolk, Virginia, USA, anaya001@odu.edu; Mohan Sunkara, Old Dominion University, Department of Computer Science, Norfolk, Virginia, USA, msunk001@odu.edu; Sampath Jayarathna, Old Dominion University, Department of Computer Science, Norfolk, Virginia, USA, sampath@cs.odu.edu; Hae-Na Lee, Michigan State University, Department of Computer Science and Engineering, East Lansing, Michigan, USA, leehaena@msu.edu; Vikas Ashok, Old Dominion University, Department of Computer Science, Norfolk, Virginia, USA, vganjigu@odu.edu.

ACM 2573-0142/2024/6-ART257
https://doi.org/10.1145/3664639

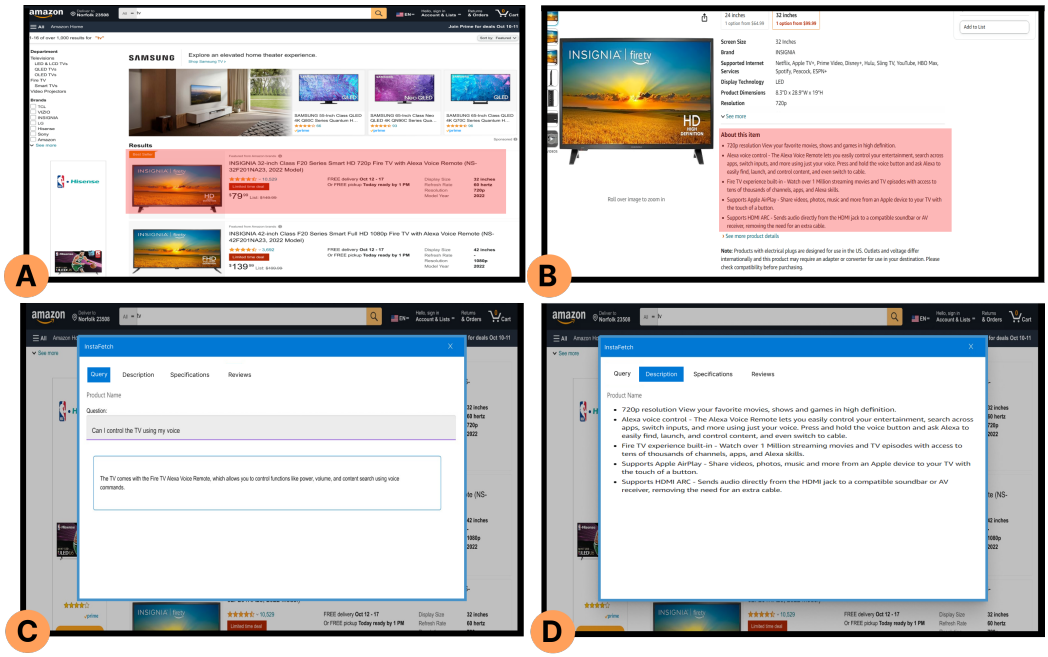

Fig. 1. InstaFetch's all-in-one interface, (A) and (B) respectively display the default 'Query-Results' and the 'Details' page. (C) and (D) showcasing four options: Query, Description, Specifications, and Reviews. The Query feature (C) on the interface enables users to input any product-related queries and receive immediate natural language responses, as detailed in Section 3.5. The other options – Description, Specifications, and Reviews, offer instant access to the content of corresponding sections extracted from the 'Details' page.

their websites. While having plenty of options is desirable, the interaction challenges and the information overload caused by the huge volume of data are also significant [41]. These challenges are even more exacerbated in the case of blind users due to their dependence on screen reader assistive technology that predominantly supports one-dimensional access to web content [20].

A screen reader, fundamentally designed to 'read out' on-screen content, also incorporates specific keyboard shortcuts or gestures to facilitate content navigation (e.g., the 'H' key for the next heading). Within the scope of web browsing, prominent screen readers such as JAWS, NVDA, and VoiceOver allow blind individuals to navigate webpages via a diverse set of keyboard shortcuts, including progression by headings, paragraphs, or hyperlinks. However, content navigation remains predominantly one-dimensional – blind users have to sequentially process extensive textual data before pinpointing their target information on a webpage [9, 33]. Given this interaction mode, blind screen reader users often require a heightened amount of time and interaction effort while doing daily online browsing compared to that required by their sighted counterparts [9].

A typical screen reader-based interaction with web data items is as follows. Upon accessing a shopping website (e.g., Amazon), the screen-reader user behavior pattern for interacting with online data involves initially navigating toward a search bar to enter a query, e.g., "TV". This is commonly achieved by utilizing the 'TAB' key or other designated shortcuts. After submitting the query, users sift through the resulting list of item summaries by employing elementary navigation keys like 'TAB' and arrow keys (Figure 1A). If a particular item piques their interest, they delve deeper by selecting its main link, which takes them to a comprehensive *Details* page (Figure 1B).

On this page, the users explore the chosen item's specifics, attributes, and associated reviews, employing a range of navigational shortcuts. Should they wish to continue shopping or exploring, they return to the previous *Query Results* page containing the list of item summaries. This entire interaction process is too tedious and cumbersome for blind users, mainly due to the sheer volume of content they have to trudge through using a multitude of keyboard screen reader shortcuts [33].

There do exist prior works that have proposed techniques to understand and alleviate the screen-reader usability problems while interacting with e-commerce web data items [20, 35, 45]. However, these efforts have predominantly focused on screen-reader navigation (e.g., enabling faster screen-reader access to the search form and filter options [20], enabling easy navigation between item summaries [34, 35], and facilitating quick navigation to product descriptions [45]). Given the high volume and density of content in e-commerce webpages, quicker screen-reader navigation to the beginning of segments alone is insufficient to ensure usable and efficient access to item data. For instance, the product descriptions are typically large collections of textual data (see Figure 1), so enabling quick navigation to the beginning of a description by itself does not fully solve the usability problem, as the user will still need to navigate and listen to a large body of text within the description to seek the desired product information.

Moreover, when users aim to gather comprehensive information from various segments of a webpage, such as descriptions, specifications, and reviews, they are required not only to locate these sections but also to mentally retain information from each segment across multiple products to facilitate informed decision-making. For instance, consider a scenario where a user is looking to purchase a television with particular attributes, such as smart casting capabilities and a minimum of four HDMI ports. Additionally, if the user is interested in understanding the product's reception, gauging this by the volume of positive reviews, the user would initiate the process by selecting the first product and then proceeding to scrutinize the specifications section to check for the necessary information. Following this, the user would explore the review section to gauge the overall positive feedback. Subsequently, the user must navigate away from the 'Details page' back to the 'Query-Results page' before proceeding to evaluate the next product. This iterative process entails a considerable amount of manual navigation and information synthesis, often making it cumbersome for users to efficiently compare multiple products.

In this paper, we address these limitations of extant research by introducing InstaFetch, a browser extension specifically designed to enhance the online e-commerce shopping experience of blind screen reader users, particularly when they interact with web data items. InstaFetch significantly simplifies the information retrieval process across multiple segments of the 'Details' page for blind users by introducing a direct query feature that allows users to directly input queries concerning any particular data item and receive instant responses, as demonstrated in Figure 1C. Instead of the traditional and often arduous method of sifting through entire sections of content, InstaFetch enables users to bypass extensive navigation by providing a mechanism to extract targeted information instantly. For instance, a user could directly input a query about a product's specifications and user reviews (e.g., "Is the TV equipped with smart cast functionality, and how many HDMI ports does it include? and what is the total count of positive reviews for this TV?") and promptly receive the relevant details. This is particularly beneficial for blind users, for whom navigating complex website structures can be especially challenging. Furthermore, if a user still seeks to examine a data item in detail, InstaFetch offers a solution by providing instantaneous access to all relevant information (such as product details, specifications, and customer reviews) that is scattered across multiple web pages, through a single uniform screen reader-friendly interface as shown in Figure 1D.

InstaFetch mainly relies on two internal frameworks: (1) Item Extractor and (2) Query-Response Generator. The Item Extractor module leverages a mask R-CNN model [28] based extraction framework built using a custom manually annotated dataset comprising 3000 ground-truth examples, to

identify and extract specific information from the 'Details' page such as product description, technical specifications, and customer reviews. The Query-Response Generator makes use of LLaMA [53] language model-based framework to process the information gathered by the Item Extractor and facilitate generating accurate responses to user queries.

In a user-study evaluation involving 14 blind participants, we observed that InstaFetch significantly reduced the frequency at which the blind users accessed the *'Details'* pages of items in comparison to both a state-of-the-art solution (SaIL [6]) as well as their preferred screen reader, thereby reducing overall cumbersome back-and-forth navigation between the 'Details' and 'Query-Results' webpages. There was also a significant improvement in the average time spent and the number of keys pressed per data item while using InstaFetch, compared to that while using either SaIL or their preferred screen reader. Consequently, within a given time duration, the participants could sift through a greater number of data items with the aid of InstaFetch. A majority of the participants also provided explicit feedback that the enhanced browsing experience facilitated by InstaFetch allowed them to reduce interaction fatigue, and also increased their likelihood of securing more "advantageous deals" while shopping online. In sum, our contributions are:

- Design and development of InstaFetch, a web browser extension that provides an instantly accessible *one-stop* interface that allows users to receive immediate responses to desired queries and also facilitates easy and quick access to relevant item information scattered across *'Details'* pages and *'Query-Results'* page in online e-commerce websites.
- Findings of a user study with 14 blind screen-reader users evaluating InstaFetch against both the status-quo as well as a state-of-the-art solution.

2 RELATED WORK

Our work closely relates to the literature on the following topics: (i) Web data usability and its impact on users with visual impairments; (ii) Information extraction from websites; and (iii) Product-driven query-response generation.

2.1 Web Data Usability for Blind Users

Prior studies have extensively explored web accessibility for blind users [8, 25, 55]. However, only a few works have addressed the usability of web content interaction for these users [6, 20, 35, 45]. Moreover, most of these works have focused on improving the usability of accessing content within a single web page [6, 20, 35]; works addressing content distributed across multiple pages are far few between [45]. Aydin et al. [6] introduced the SaIL system that automatically pinpointed the 'hot spots' in webpages. ARIA landmarks were then injected into these hot-spot page segments so that users could quickly access the beginning of these segments using designated screen reader shortcuts. Similarly, Ferdous et al. [20] presented the InSupport system, which offered blind users an accessible proxy interface to directly access the root of auxiliary segments such as product filters and multi-page links in the 'Query-Results' page. Prakash et al. [45] further extended usability support for web data items through their AutoDesc system, which automatically extracted product descriptions from the 'Details' page and then injected these descriptions into the corresponding product summaries on the 'Query-Results' page. All these solutions were found to significantly improve usability for blind users over the status quo screen readers. However, all of these solutions are limited to addressing usability issues mostly within a single webpage, with minimal support for content split across multiple pages. Therefore, these solutions cannot fully handle scenarios where item content is distributed across multiple webpages. Moreover, these solutions only provide facilities to quickly access the beginning of item-related segments (e.g., descriptions, filters); finding the desired

information both within a segment and across segments is still a tedious and cumbersome process for blind screen reader users. In this paper, we fill these exact usability gaps with our InstaFetch.

2.2 Information Extraction from Websites

With the internet evolving rapidly, websites consistently generate extensive amounts of data to meet the diverse needs of their users. Hence, extracting pertinent information from these websites is crucial for enhanced user experience [49]. A plethora of extraction techniques are available to extract various types of data from web pages [3, 5, 13]. This includes data items [4, 19], widgets [40], news articles [32], auxiliary segments [20] and product descriptions [45]. For instance, Alvarez et al. [4] observed that data items (e.g., list of shopping products) in the DOM tree comprises several complete subtrees, and each DOM node corresponding to some item property (e.g., price) has the same path from the root of the DOM tree across all items. Based on these xpath patterns, they employed clustering algorithms to group similar subtrees to detect web data items and then extracted the item properties using a multiple-string alignment-based method. In addition to items, several techniques have been explored for the extraction of diverse kinds of web data, such as widgets [40], news articles [32], and auxiliary segments [20]. For example, Melnyk et al. [40] introduced machine learning models to extract web widgets such as popup menus, chat boxes, and calendars. Their system provided blind users access to chat widgets via a generic accessible interface. To the best of our knowledge, there are currently no datasets or algorithms specifically designed to automatically extract important item-related information such as Product Details, Full Specifications, and Customer Reviews in real-time from corresponding 'Details' pages. We also fill this gap in this paper by building a Mask R-CNN-based extraction model [28] on top of a novel custom dataset that captures all relevant information in an item's 'Details' page.

We chose Mask R-CNN [28], a specialized form of CNN, because of its state-of-the-art performance in page object detection tasks, compared to other contemporary architectures such as Vislnt, SOS, UITVN, and Matiai-ee [56]. Mask R-CNN is particularly effective in our problem scenario because it incorporates semantic segmentation to provide pixel-level detection (masks) of objects on a page [28]. Therefore, we built a custom Mask R-CNN model to extract relevant information from webpages in InstaFetch.

2.3 Product-based Query-Response Systems

The excessive information overload on e-commerce websites has led to an increased popularity of question-answering (QA) systems that help users easily locate specific information of interest [7, 30]. Many of these systems have demonstrated promising results in generating accurate product-aware responses to users' natural language queries [15–17, 21, 22, 58]. The subjectivity and reliability inherent in user queries pose distinct challenges for Product Question Answering (PQA) compared to conventional web-based QA [17]. Chen et al. [15] addressed these challenges through their 'RAGE' system, which employed a convolutional Seq2Seq architecture and recurrent neural networks (RNNs) to create accurate review-driven responses to user queries on e-commerce websites. Gao et al. [22], however, proposed an alternative adversarial learning method that utilized an attention-based review reader to extract question-aware facts from reviews and item attributes to generate responses to user queries. Although these systems significantly reduced the probability of inaccurate answers, they lacked reasoning capabilities and produced 'safe' answers in multi-hop question-answering scenarios. These issues were addressed in a follow-up research work, wherein Gao et al. [21] proposed a 'Meaningful Product Answering Generator' (MPAG). This system incorporated review clustering and instilled reasoning among them through a selective reading mechanism and read-write memory for encoding. To overcome the 'safe' answer problem, they extracted answer skeletons and employed an RNN-based encoder to generate the final response.

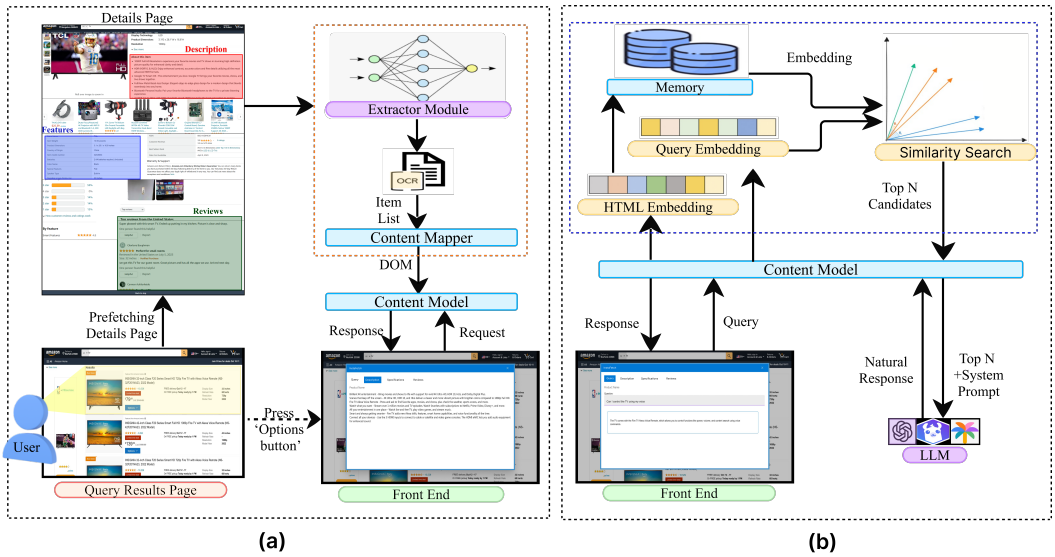

Fig. 2. InstaFetch architectural workflow (a) Info Phase and (b) Query Phase.

In real-time scenarios, however, a QA system must possess the capability to reason across various sections of a webpage to respond to a complex user query. In their work, Yao et al. [57] proposed the 'ReAct,' which equipped Large Language models (LLM) with reasoning, fact-checking, and interactive decision-making abilities. This system presented a state-of-the-art architecture for creating highly efficient and precise multi-hop question-answering systems. InstaFetch novelty lies in employing ReAct prompting along with Chain-of-thought [54] in the RAG (Retrieval Augmented Generation) [36] pipeline for handling multiple sections on a webpage within the context of the QA system. Thus, InstaFetch is capable of reasoning on information distributed across various sections of a webpage and then constructing a coherent and accurate response to user queries.

3 INSTAFETCH DESIGN

3.1 Overview

Figures 2a and b collectively present the architectural design and workflow of InstaFetch. When a user loads the 'Query-Results' webpage, InstaFetch leverages the STEM algorithm [19] to automatically detect item summaries and then embeds an 'Options' button into each data summary.

As shown in Figure 2a, when the user clicks on an 'Options' button for a specific item, InstaFetch triggers the following sequence of internal actions. First, InstaFetch pre-fetches the corresponding 'Details page' and feeds it to an Information Extractor. Next, the Extractor identifies various relevant contents on the details page, such as product descriptions, specifications, and customer reviews, using a custom-built Mask R-CNN based model [28], then extracts the corresponding texts using Optical Character Recognition (OCR) [50], and stores the extracted data within the *Content Model* after pre-processing. This computer vision-based approach was chosen to effectively handle the diverse and often changing DOM structures of various e-commerce webpages, ensuring that InstaFetch can generalize and remain functional across multiple online shopping platforms despite frequent updates.

InstaFetch then leverages the extracted Content Model to support a natural language 'Query' feature[1] (see Figure 2) that allows users to input product-related queries in natural language. For instance, a user might type in a question like "What are the battery life details of this laptop?" (single-hop) or "Does this camera have a warranty?", and "Can I get the highest liked review?" (multi-hop). To answer these queries, InstaFetch leverages a custom LLaMA model [53] refined through prompt engineering for generating product-aware natural language responses. InstaFetch also visualizes the entire Content Model in a screen reader-friendly interface that is seamlessly injected into the 'Query-Results' page in real-time, as illustrated in Figure1D. The user can then navigate the InstaFetch interface to quickly and conveniently view all extracted item information without having to navigate away from the 'Query-Results' page. If the user chooses to close the interface, InstaFetch automatically returns the screen reader's focus to the 'Options' button, thereby allowing the user to continue exploring other items on the page. This streamlined access to relevant information via InstaFetch offers blind users the potential to significantly conserve time and effort. By mitigating the need for repetitive back-and-forth navigation between the 'Query-Results' page and the 'Details' pages, users can more efficiently find and assess the desired information.

3.2 Extracting Item Information

Our goal in this regard was to engineer an Information Extractor that is generalizable across various e-commerce webpages. Current literature in this domain primarily harnesses a combination of visual cues, textual data, and DOM features to meticulously identify and parse every element within e-commerce webpages [26, 31]. However, our objectives diverge; we seek to simplify and speed up this process by focusing exclusively on three critical content segments within e-commerce platforms: *product descriptions*, *specifications*, and *user reviews*, that contain all necessary product-related information. To this end, our methodology is anchored in the application of Object Detection techniques within the realm of computer vision [24, 28] primarily because object detection methods are particularly adept at identifying the specific visual cues that distinguish product descriptions, specifications, and reviews, independent of the varying DOM structures across websites. This approach significantly simplifies the training and scalability aspects of extraction, allowing for efficient development and easier adaptation to the dynamic nature of e-commerce platforms. To extract relevant information from the 'Details' pages, InstaFetch leverages a custom Mask R-CNN model [56]. The training of this model required the creation of a custom dataset, which involved manual annotation of various regions of interest on a variety of 'Details' pages, as explained next.

Dataset. We collected 3,000 images from 1,000 different 'Details' pages. These pages were sourced from diverse domains such as Business and Finance (10%), Home and Garden (20%), Shopping (40%), Style and Fashion (20%), and Technology (10%)[2]. For the 'Details' pages, we manually annotated regions of interest by crafting rectangular masks using the GIMP software [52]. This step was crucial for precisely identifying and segmenting the relevant areas on the webpages. After creating these masks, we then mapped them onto the images, ensuring each image was paired with its corresponding mask. We also compiled metadata for the dataset, which encompassed details such as the name, URL, and creator of each webpage. The complete dataset, along with all relevant information, can be accessed through the anonymous project GitHub folder[3].

[1]The "Query" feature in InstaFetch refers to the LLM-based natural language support, while "Query-Results" denotes the default online page where users search for products.
[2]For detailed information and to access the complete list of webpages utilized in our study, please refer to the GitHub link below.
[3]https://github.com/accessodu/InstaFetch.git

Training. We trained the model utilizing NVIDIA V100 GPU with 128 GB memory per node; the training process spanned across 10 epochs, each containing 500 steps. For our training process, we utilized the Mask R-CNN framework [1], incorporating the ResNet-101 backbone [29] augmented by the Feature Pyramid Network (FPN) [38]. The inherent strength of Feature Pyramid Network is its top-down architecture, which, when integrated with lateral connections, provides an enriched hierarchical feature representation.

The training process began with feature extraction using ResNet-101 (with FPN) to obtain feature maps from the input. These feature maps were then used by the Region Proposal Network (RPN) [60] to generate appropriate webpage object proposals. Following this step, a Region of Interest Align (RoIAlign) technique [28] extracted accurate and precise feature maps. These maps are then fed simultaneously to two separate convolution layers: masked branch convolution layers and regular convolution layers. The masked branch convolution layers then produced the required masks, whereas the regular convolution layers handled the object classification and bounding box delineation. Finally, the masks generated were aligned to the corresponding identified objects within their bounding boxes. The entire training was bifurcated into two distinct phases:

- Initial Adaptation Stage: Emphasis was placed solely on the head layers of the Mask R-CNN model, ensuring the backbone layers remained static. A learning rate of 0.001 was chosen for this segment.
- Refinement Stage: The model underwent a comprehensive fine-tuning. Here, all layers were trained with a revised learning rate of 0.0001.

These specific parameter configurations emerged during optimization efforts on the accuracy of the validation dataset consisting of 300 images. All information about the model is also available in the GitHub folder[4].

Evaluation. On 20 previously unseen websites, we assessed the model using Average Precision (AP) [28] at different Intersections over Union (IoU) thresholds [28], which measure the accuracy of the predicted area against the actual area. Our results showed a Mean Average Precision (MAP)[5] of 75.4% at a 50% IoU threshold and 69.7% at a 75% IoU threshold, indicating the model's effectiveness in accurately identifying regions of interest. We also monitored the total loss, which is the sum of all the losses (loss mask, loss classifier, and loss box regression), and noticed a total loss of 0.529 at convergence point.

3.3 Post-processing

The output from the Mask R-CNN model highlights the regions of interest on the 'Details' page. However, these images often contain some noise, which can affect the output quality when passed through the OCR. The OCR process itself can introduce additional noise, leading to imperfect results. Therefore, to enhance the extraction accuracy, we implemented post-processing as follows: Using the text obtained from OCR as a reference, we employed a simple search algorithm to locate where this text appears within the HTML DOM of the 'Details' page. Once identified, the algorithm traced back through the DOM tree to capture the parent element encompassing the text. The parent node and corresponding children nodes are then extracted and stored by InstaFetch (Figure 2a).

3.4 InstaFetch User Interface

The interactive overlay popup of InstaFetch presents users with four distinct functionality tabs: 'Query', 'Description', 'Specifications', and 'Reviews', as shown in Figure 6 (Appendix A). The

[4]https://github.com/accessodu/InstaFetch.git
[5]The Average of AP for each category - product description, specification, and reviews

interface was meticulously designed for easy navigation with simple 'TAB' and 'ARROW' shortcut keys. The functionalities on this interface were optimized for accessibility through the utilization of tab-index and ARIA (Accessible Rich Internet Applications) attributes. The tab-index attribute determines the order in which elements receive keyboard focus, while the aria-label attribute provides users with additional information about the web element [48]. The 'Query' button enables users to submit product-related questions through an input field, whereas the other functionality tabs in the InstaFetch overlay popup interface showcase specific product sections (i.e., 'Description', 'Specifications', and 'Reviews') while mirroring their structure on the 'Details' page. Initially, when the popup interface is activated, the focus automatically rests on the 'Query' button. If a user presses the key 'ENTER', the user is directed to a form, with the focus shifting seamlessly from the button to the form itself. This form enables users to input and submit their product-related queries by pressing the 'ENTER' key a second time (as illustrated in Figure 6). After query submission, InstaFetch proceeds to retrieve a suitable answer from its back-end system and displays it beneath the form. The screen reader's focus then transitions to the response area once the answer is displayed. In instances where the system cannot produce a relevant response, a standard error message is provided – "Sorry, I do not have an answer to that question." Furthermore, when users press the 'ENTER' key on the other three functionalities, the corresponding content (Description, Specification, or Reviews) from the 'Details' page are fetched and displayed below the navigation section (see Figure 6), while also automatically shifting the focus from the button to the injected content. This interaction design is consistent across the Description, Specifications, and Reviews sections of the InstaFetch interface. The interface also has a close button, providing users with the ability to close the overlay popup and seamlessly return to the exact product they were previously engaging with before accessing the interface. Lastly, note that InstaFetch does not support its own keyboard shortcuts; instead, it extends the current webpage with an additional HTML pop-up interface that is navigable with the standard screen reader shortcuts. We adopted this design to make InstaFetch compatible with any screen reader.

3.5 Dynamic user query handling

The 'Query' feature is the most significant part of InstaFetch as it empowers blind web users to quickly access information scattered across multiple sections on a product's 'Details' page. While prior research has comprehensively delved into question-answering, they primarily address specific sections of a webpage [14, 16, 21], none of the prior research works have explored reasoning collectively across different sections of a webpage and that too in the context of easing web navigation for blind users. InstaFetch addresses this research gap by providing an interface for blind users to seek responses to complex product-related queries. This process of generating accurate product-aware responses to user queries in InstaFetch involves the following sequence of steps: (i) Constructing a contextual knowledge base utilizing data extracted by the Extractor module; (ii) Retrieving information that is pertinent to the user query; and (iii) Generating coherent natural language response to the user query. Details for each of these steps are provided next.

3.5.1 Constructing contextual knowledge base. The 'Query' feature of InstaFetch relies on a custom knowledge base to provide necessary information for constructing valid responses to user queries. This knowledge base supplies product-specific information to the LLM [36] and ensures that responses are generated within the context of the product details available on the 'Details' page, thereby preventing model hallucinations. The Extractor module (Section 3.2) provides relevant information related to a specific product in the form of HTML data for each respective data section. The large size and complexity of the HTML data obtained from the extractor module could limit the LLM's ability to comprehend the entire contents due to its limited contextual window [37, 39]. To

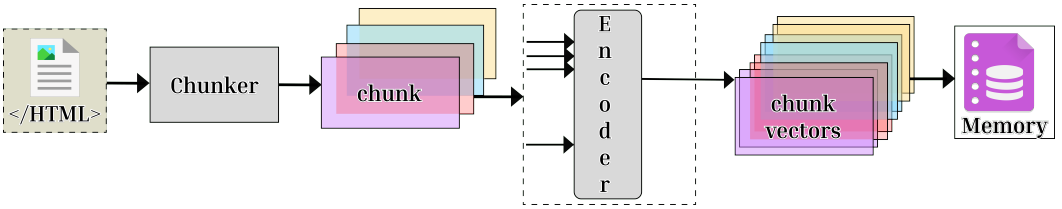

Fig. 3. Construction of knowledge base for response generation

address this, we fragment the document into smaller, semantically relevant chunks using a structure-aware chunking strategy [44]. Additionally, we reduce the likelihood of missing pertinent details during information retrieval by incorporating sufficient overlap between adjacent chunks [18]. These chunks of HTML data are then converted into dense vector representations using an OpenAI embedder [42] and cached in the system's memory for subsequent reference as shown in Figure 3.

3.5.2 Retrieving relevant information. In the answer-generation process, not all the information available in the knowledge base may be relevant. Hence, we selectively retrieve only the information pertinent to the query and pass it to the Large Language Model (LLM) for reasoning and response generation. InstaFetch employs the Retrieval-Augmented Generation (RAG) framework that includes a retriever and a generator [36]. When a user query is received via an HTTPS POST request, it is translated into dense vector representations using the OpenAI embedder [42]. The retriever then executes a cosine similarity of these query embeddings against the cached HTML embeddings to retrieve the top N chunks that exhibit the highest degree of similarity. This retrieved group of relevant chunks can include information of diverse nature and format, such as table elements, description lists, and user reviews. This information is then passed to the generation module, along with the query, mounted on a well-engineered prompt to construct a response to the user query, as shown in Figure 2b.

3.5.3 Generating query response. We employed 'Prompt-Engineering' to guide a pre-trained LLaMA Large Language Model (LLM) [53] to construct accurate product-aware responses to user queries by utilizing information of diverse formats from multiple sections of a web page. This prompt is vital to instruct the LLM to comprehend the user query and reason on the pertinent information available from the retriever module to generate a natural-sounding response.

To achieve this, we employ Chain-of-thought (CoT) [54] and ReAct [57] techniques to empower the LLM to reason across multiple data sections dynamically. With CoT, when a user asks a complex question, the system breaks down its reasoning process step by step before providing an answer. This is like the system showing its "thought process" behind reaching a conclusion or solving a problem. ReAct takes this a step further by not just reasoning out loud but also taking "actions" – for example, fetching the latest prices, or checking user reviews to provide a more informed, accurate answer. For example,

(1) User Question: "What's the battery life and is it good?"
(2) CoT: "Battery life is usually mentioned in the specifications or reviews."
(3) ReAct: Check the product's specifications and also summarizes positive customer reviews mentioning battery life.

By combining CoT and ReAct, the system not only understands user questions in depth but also actively seeks out and verifies the information needed to give the user accurate, helpful answers.

For improved understanding and reasoning, we structure prompts to include examples of reasoning (CoT) and action (ReAct), thereby guiding the LLM to not only produce answers but also

follow logical steps in fetching and incorporating external information. For this, we leverage a few-shot prompt template that was constructed by providing demonstrative task examples [12, 46]. These examples were defined manually for diverse questions that could be potentially raised on the 'Details' page. Additionally, the prompt also ensured that the model refrained from generating generic responses. In cases where a suitable match was not found, it delivered a standardized "Response not found" error message, enhancing the model's reliability and user experience. All details are available on the anonymous GitHub project folder[6].

Note that off-the-shelf LLMs cannot be used as-is for our problem, as they lack the context and domain-specific knowledge required for real-time product-related queries. Moreover, re-training or fine-tuning these models is not always feasible due to high computational costs [59]. We address these issues in our LLM customization by integrating Retrieval Augmented Generation [36] and Prompt engineering techniques [53]. Specifically, we incorporated CoT (known for its detailed reasoning) [54] and ReAct (known for its factual grounding) [57] prompting strategies that guided the LLM through a series of logical steps leading to the final response.

Evaluation. To evaluate the quality of responses generated by InstaFetch, we employed BLEU [43] score, which measures the lexical unit overlap between the generated response and the ground truth. The BLEU score for the sampled set of questions was 0.78, indicating high similarity and good performance of the system. Additionally, we conducted manual evaluations of response accuracy, as automatic evaluation metrics could be misleading [51]. We invited 10 annotators from diverse academic and professional backgrounds to evaluate the quality of the responses generated on multiple web pages for a total of 50 randomly sampled questions for which the response was generated by InstaFetch.

The annotators rated the answers generated by InstaFetch on a scale of 1 to 10 for the following metrics: (i) Factuality (1 for least accurate and 10 for highly accurate): determining if the facts mentioned in the response were correct or not; (ii) Relevance (1 for least relevant and 10 for highly relevant): measuring the degree to which the answer was relevant to the question and did not deviate from its context; and (iii) Grammaticality (1 for grammatically incorrect and 10 for no grammatical errors): determining if the answer generated was grammatically correct.

The sampled set of questions included: (i) simple questions that could be answered by a direct one-shot reference to any content on the 'Details' page (e.g., "How many HDMI ports are available on this TV?"); (ii) "Single-hop questions that required reasoning (e.g., "Can I connect my laptop to the TV using a USB cable?"); and (iii) multi-hop questions that required reasoning across multiple pieces of content spanning different sections on the web page (e.g., "Does the TV support surround sound, and what do past buyers think about the sound quality of the response generated?").

The mean rating for all questions was calculated for different metrics. On average, the responses generated by InstaFetch were rated at 8 for factuality, 6.6 for relevance, and 9.2 for grammaticality. Inaccuracies in relevance primarily stemmed from conflicting information extracted from different sections. Notably, the user-generated review sections on webpages posed a challenge due to their inherent ambiguity, resulting in a higher frequency of conflicting data within the generated responses. For example, a participant queried about the quality of a certain TV on Amazon. While the product description might boast of high picture quality, users in the review section observed a lack of clarity and sharpness. In such cases, InstaFetch produced responses that were not relevant to the users' expected response.

[6] https://github.com/accessodu/InstaFetch.git

3.6 Implementation Details

This section describes the implementation challenges and engineering efforts involved in implementing various modules and integrating them into InstaFetch.

We implemented InstaFetch as a web browser extension, following the open-source guidelines provided by Google[7]. When the browser extension is enabled, the service worker is activated, which listens and responds to browser events such as loading of a new page or closing of an existing page[8]. Once the 'Query-Results' page is loaded, the content scripts[9] are injected into the webpage. These Javascript files communicate with the parent extension code and have access to modify the standard DOM of the webpage. When the 'Query-results' page loads, InstaFetch leverages the STEM algorithm [19] (whose code is available at GitHub[10]) to identify data items and inject the UI code for the popup overlay interface into the DOM tree including the 'Options' button for every data item listed on the webpage. The injected HTML code included special labels and screen reader-focus modifications to make the elements accessible to blind users. The UI was implemented using HTML, CSS, and JavaScript.

Once the users selected a data item of choice, we employed the Selenium driver [23] that captures comprehensive snapshots of multiple windows of the entire 'Details' page. These snapshots serve as the preliminary input for the subsequent extraction process executed by the Mask R-CNN model. To build the Mask R-CNN model for extracting the item description from the 'Details' page, we leveraged the publicly available Matterport code on GitHub[11]. For post-processing of the Mask R-CNN output, we used the Tesseract OCR engine [50]. We specifically used the raw text output from the OCR to feed a simple DOM search algorithm to retrieve the respective DOM subtrees for each of the data sections (i.e., Reviews, Description, Specification) from the webpage's HTML code and stored them in the Content Model (Section 3.2).

After the DOM data for the web elements was extracted and stored, we leveraged the HTMLHeaderTextSplitter class provided by the Langchain framework[12] to fragment them into semantically grouped smaller chunks that included metadata providing information about where that chunk came from based on the HTML. We then employed the OpenAI embedder [42] to translate these fragments of code to dense vectors. These vector representations of the webpage were then cached in the system memory using a Python dictionary. These stored vectors of web data elements served as the knowledge base for the Q&A functionality of InstaFetch.

Whenever a user submits a query from the InstaFetch user interface, the server receives a POST request with the query in the request body. This query was then extracted and embedded using OpenAI 'text-embedding-ada-002' embedder. A custom retriever module was then utilized to execute a similarity search of the query embeddings against cached DOM vectors to shortlist the documents that exhibited the highest degree of similarity. These documents were then sent to a Llama LLM with a well-engineered prompt to generate a natural-sounding response, which was then relayed back to the user interface. In contrast, whenever a user hits either the Description, Technical Specifications, or Review button, the backend server receives a GET request. The request is then handled, and the respective data element is sent back to the front end for rendering[13].

[7] https://developer.chrome.com/docs/extensions/mv3/devguide/
[8] https://developer.chrome.com/docs/extensions/mv3/service_workers/
[9] https://developer.chrome.com/docs/extensions/mv3/content_scripts/
[10] https://github.com/accessodu/InstaFetch.git
[11] https://github.com/matterport/Mask_RCNN
[12] https://python.langchain.com/docs/get_started/introduction
[13] https://developer.mozilla.org/en-US/docs/Web/HTML

The backend server was developed using Django REST Framework (DRF)[14] and Python. This server was responsible for managing incoming requests, executing essential functions, and providing responses to the front-end user interface. All the code and demo videos associated with InstaFetch is available on GitHub[15].

4 EVALUATION

To evaluate InstaFetch, we conducted an IRB-approved user study with blind screen-reader users as explained next.

4.1 Participants

We recruited a total of 14 blind participants[16] through email lists and word-of-mouth snowball sampling. These participants were selected based on the following specific inclusion criteria: (i) experience with web browsing using a screen reader in a desktop/laptop environment; (ii) familiarity with the Chrome web browser; and (iii) ability to communicate in English. Gender representation was approximately balanced, with 6 female and 8 male participants, and the average age of the participants was 31.14 (with a Median of 31.5, a Minimum of 20, and a Maximum of 43). None of the participants reported any other difficulties, such as hearing or motor control issues, that interfered with their ability to complete the study tasks. Table 1 presents the full self-reported participant demographic information.

4.2 Design

In a within-subject experimental design, influenced by previous research [20, 34], participants were asked to do a representative online shopping task under the following three distinct study conditions or treatments:

- Screen Reader – In this status-quo condition, the participants performed the task using their preferred screen reader.
- SaIL – In this condition, the participants did the task with assistance from a state-of-the-art solution, namely SaIL [6]. SaIL automatically detects salient segments (e.g., menu, search form, item summaries, filters) in the current webpage and then injects special ARIA landmarks into the detected segments, so that users can quickly navigate to these segments using special screen reader shortcuts (e.g., 'R' in JAWS screen reader).
- InstaFetch – In this condition, the participants did the task with the assistance of InstaFetch browser extension.

In each of these conditions, the participants were asked to browse a list of products on an e-commerce website and choose a product that most closely aligned with their individual preferences. We chose this task to simulate real-world scenarios where individuals typically review lists of items, compare their features, and ultimately select their preferred item from the list. To minimize the potential impact of a learning effect, we ensured that the same website was not used more than once when performing the task under different conditions. Instead, we selected three different shopping websites for the three conditions: Amazon, Etsy, and eBay. We further ensured that the product types explored in the three conditions were also different; specifically, we used the following three product types: Television, Furniture, and shoes. The exact assignment of product types to websites, websites to conditions, and the ordering of conditions were all counterbalanced across

[14] https://www.django-rest-framework.org/
[15] https://github.com/accessodu/InstaFetch.git
[16] This is the typical sample size for research in this area, due to the difficulty in recruiting participants belonging to this disadvantaged community [20, 35].

Table 1. Demographics of blind participants in the InstaFetch evaluation study. All information was self-reported by the participants.

ID	Age	Gender	Age of Vision Loss	Occupation	Preferred Screen Reader	Web Experience	E-Commerce Use (times/week)
P1	43	M	Since birth	Teacher	JAWS	7 years	4-6
P2	36	M	Age 8	Unemployed	JAWS	4 years	2-3
P3	28	M	Since birth	Student	NVDA	2 years	1-2
P4	23	M	Age 10	Student	NVDA	1 year	3-5
P5	36	F	Since birth	Social worker	JAWS	5 years	5
P6	32	M	Age 6	Teacher	JAWS	4 years	3-4
P7	25	F	Cannot remember	Unemployed	NVDA	2 years	2-3
P8	38	M	Cannot remember	Social-worker	JAWS	6 years	7-8
P9	31	F	Since birth	Teacher	JAWS	5 years	3-4
P10	22	F	Age 8	Student	NVDA	1 year	2
P11	27	F	Since birth	Unemployed	NVDA	3 years	5-6
P12	34	M	Cannot remember	IT employee	JAWS	7 years	4-5
P13	20	F	Since birth	Unemployed	NVDA	1 year	1-2
P14	41	M	Cannot remember	Teacher	JAWS	8 years	6-8

the study participants to the best extent possible using the well-known Latin-square method [10]. A maximum of 20 minutes was allotted for each study task.

4.3 Apparatus and Procedure

Depending on the participant's availability and location, the user study was conducted either in-person or virtually via Zoom. Participants used their own computers/laptops with their preferred screen readers to do the study tasks. All web pages related to the tasks were pre-processed, cached, and securely hosted on a web server. During the pre-processing stage, the SaIL landmarks were proactively computed and injected into the task webpages before caching them on the web server. The InstaFetch 'Options' button too was injected a priori into the task webpages before caching, and moreover all InstaFetch functionalities were setup on the web server. This setup ensured that all participants (both in-person and remote) could do all the tasks in all the conditions without having to install anything on their computers. This cached setup also helped avoid any confounding effects of frequent website changes.

The study commenced with the experimenter obtaining formal consent from the participant and providing a brief explanation of the study's purpose. Next, the experimenter introduced the different study conditions, and allowed the participant to practice for 20 minutes in order to get comfortable using both the SaIL landmarks and the InstaFetch interface. The experimenter then administered the tasks one-by-one in the predetermined counterbalanced order. For each task, a maximum of 20 minutes was allocated, however, this time limit was not explicitly conveyed to the participant in advance, so as to prevent any confounding influence on their natural interaction behavior with web data items.

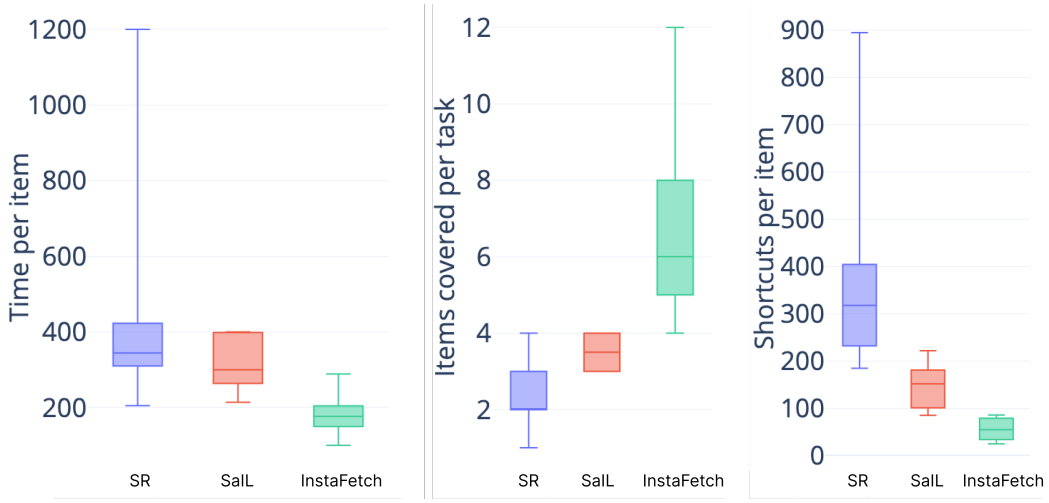

Fig. 4. Box plots for the Average time spent per item, the Average number of items covered during the task, and the Average number of Shortcut Presses per item in all three study conditions.

Upon completing the study tasks, the experimenter administered the System Usability Scale (SUS) [11] and NASA Task Load Index (NASA-TLX) [27] subjective questionnaires. These questionnaires were intended to assess the perceived usability and interaction effort for each of the study conditions. This was following by an exit interview, where the experimenter engaged in an open-ended discussion with the participant to gather qualitative feedback, including feature requests and suggestions for improvement. Upon the participant's consent, screen-sharing and recording features were kept active throughout the study to capture all interaction activities for post-study analysis. All conversations were conducted in English, and participants received compensation in the form of an Amazon gift card.

4.4 Data Collection and Analysis

From the study data, we computed the following metrics for each participant: (i) Average time spent per item; (ii) Average number of shortcuts pressed per item (including navigating the *Details* page) while doing a task; (iii) Number of items covered while doing a task; (iv) SUS usability scores; (v) NASA-TLX workload scores; and (vi) Qualitative feedback. These metrics and data allowed us to compare the various study conditions and assess whether InstaFetch had a significant positive impact on the overall user experience when interacting with web data items. We report our findings in the next subsection.

4.5 Results

4.5.1 Average time spent per item. The average time spent by a participant per data item was determined by dividing the total time spent on the task (irrespective of its completion status) by the number of unique items they interacted with during the task. Figure 4 presents the statistics for this metric for all three study conditions. In general, participants spent an average of 478 seconds (with Median = 344, Minimum = 205, Maximum = 1200) when using screen readers, 310 seconds (with Median = 299, Minimum = 214, Maximum = 400) with SaIL, and 182 seconds (with Median = 176, Minimum = 100, Maximum = 289) with InstaFetch. This 182 seconds per item with InstaFetch included a 2.1-second average time delay for fetching the 'Description,' 'Specifications' and 'Reviews'

data in the back-end and a 3.8-second average time overhead for obtaining product-related responses from the customized LLM service to address user queries. A Friedman test demonstrated that there was a significant[17] effect of the study condition on the average time spent per item ($\chi^2 = 12.6$, $p = 0.0018$). Subsequently, a post-hoc Conover's test with Benjaminyi-Hochberg FDR adjustment revealed that InstaFetch significantly outperformed both the screen reader ($p = 0.01$) and the SaIL ($p = 0.022$) study conditions.

A comprehensive analysis of the screen reader log data showed that the average time spent per item was greatly influenced by two main task activities: (i) Finding the webpage segment of interest, i.e., *Description*, *Technical Specification*, and *User Reviews*; and (ii) Navigating back to the item summary on the 'Query Results' page after exploring the 'Details' page for additional item-related information. The time spent on these activities was significantly lower in the InstaFetch condition, the gain was attributed to the fact that 70% of the users had pre-existing knowledge about the specific information they sought from the products, i.e., since the participants already knew the 'type' of products (i.e., television, furniture, and shoes) they were perusing, they each had a good idea of what features or attributes (e.g., shoe material, TV size) mattered to them the most for making a final selection. Consequently, these users directly interacted with the interface by inputting their queries and were generally satisfied with the information provided in response. Their satisfaction was evidenced by their behavior; they either submitted a follow-up query for additional details or proceeded to evaluate another product. Notably, when users exhibited a deeper interest in a particular item, they could access related information seamlessly through the interface. This feature significantly curtailed the need for users to visit the 'Details' page of the product, indicating the effectiveness of InstaFetch in streamlining the information retrieval process. The time spent was highest in the Screen Reader condition, where the users frequently navigated back and forth between the 'Query Results' and 'Details' pages. In the SaIL condition, the participants spent relatively less time per item compared to that in the screen reader condition, likely because SaIL-injected ARIA landmarks helped them skip much irrelevant content. However, a few participants forgot their shortcuts for navigating ARIA landmarks during the task, leading them to revert to standard one-dimensional screen reader navigation, which in turn increased their time spent per data item.

4.5.2 Number of Shortcut Presses per Item. The number of keyboard shortcuts is directly proportional to the effort exerted by the participants to do the tasks, as determined in prior studies [20]. Therefore, fewer shortcuts mean reduced task effort and, hence, better usability. Similar to *average time spent per item*, the average number of keyboard shortcuts per item was calculated by dividing the total number of shortcuts pressed by the participant throughout the task (regardless of task completion) by the number of unique items explored during the task. Figure 4c presents the statistics for this metric for all three study conditions. In sum, participants averaged 397 shortcuts (with Median = 317, Minimum = 185, Maximum = 895) while using screen readers, 145 shortcuts (with Median = 151, Minimum = 85, Maximum = 222) with SaIL, and 57 shortcuts (with Median = 55, Minimum = 25, Maximum = 85) with InstaFetch. These differences in the average number of keyboard shortcuts between the study conditions was found to be statistically significant (Friedman's test, $\chi^2 = 20.0$, $p < 0.001$). A post-hoc Conover's test with Benjaminyi-Hochberg FDR adjustment demonstrated that InstaFetch significantly outperformed both screen reader ($p < 0.001$) and SaIL ($p = 0.03$).

Analysis of the screen reader shortcut log data revealed that the main cause underlying the differences in the observed values between conditions was the navigation behavior adopted by the user on the task webpages in different conditions, which in-turn affected the amount of content navigated during the course of locating desired item-related information. In the Screen-Reader

[17]By 'significant', we mean statistical significance.

condition, a majority of the participants used only a handful of basic navigation shortcuts such as UP/DOWN Arrow keys, TAB key, and H key (on average 6 unique keys per participant), therefore they had to sift through a lot of irrelevant content in the task webpages before reaching the desired content of interest. This burden was somewhat reduced in case of the SaIL condition, where the participants could avoid much of the irrelevant content by leveraging the special landmark shortcut. However, in the SaIL condition, the participants still had to navigate back-and-forth between the Query Results page and the Details pages, and also reposition themselves within each webpage, which involved considerable shortcut presses. In the InstaFetch condition, as most of the item-relevant data was accessible through the InstaFetch interface in the Query Results page itself, there was a significant reduction in the one dimensional navigation effort, thereby resulting in fewer key shortcut presses.

4.5.3 Number of items covered during task. This metric measured the count of unique items that a participant interacted with during a task, excluding any revisits to items already explored. This metric is significant as it quantifies the breadth of user engagement with different items within a task, offering insights into the diversity of user exploration and interest. Figure 4b displays the statistics for this metric for all three study conditions. Overall, the participants explored an average of 2.2 items (Median = 2, Minimum = 1, Maximum = 4) while using a screen reader, 3.5 items (Median = 3.5, Minimum = 3, Maximum = 4) with SaIL, and 6.8 items (Median = 6, Minimum = 4, Maximum = 12) with InstaFetch. An inferential analysis using the Friedman test revealed a statistically significant difference in the number of items explored between the three study conditions ($\chi^2 = 18.2$, $p < 0.001$). Similar to the previous metrics, a post-hoc Conover's test with Benjaminyi-Hochberg FDR adjustment demonstrated that InstaFetch significantly outperformed both the screen reader ($p = 0.001$) and the SaIL ($p = 0.036$) study conditions.

The average number of unique items explored during a task was again influenced by the participant's navigation behavior especially regarding revisits to the previously-explored items. The number of revisits were significantly higher in the Screen Reader and SaIL conditions compared to the InstaFetch condition. Each revisit added a significant time overhead, thereby affecting the number of items covered in the allotted task time of 20 minutes. As explained later in the paper, many participants also attributed the reduced coverage in the Screen Reader and SaIL conditions to interaction fatigue, specifically, they stated that listening to a lot of irrelevant content during navigation increased frustration and reduced motivation to further explore the list of items. Since InstaFetch provided direct access to almost all item-related information on the 'Query-Results' page itself, the interaction fatigue was substantially less compared to that in other conditions, thereby enabling the participants to explore more items.

4.5.4 Use of Natural Language Queries and Errors. Overall, 12 out of 14 participants used natural language queries while doing tasks in the InstaFetch condition. These 12 participants issued a total of 40 queries, with an average of 3.3 queries (Standard Deviation: 2.53) per participant. Based on the participants' behavior and utterances while using the query feature, the experimenter noticed that 5 of these 12 participants were issuing a higher number of queries purely out of curiosity and for 'fun', and it was not clear if their intention in these scenarios was to deeply examine the product item details. The remaining 7 participants were more specific in their querying and did not issue back-to-back queries. The experimenter observed that 5 of 7 participants repeatedly asked the same question for multiple items ("What is the total number of good reviews?"), most likely to compare these items based on common aspect (e.g., number of good reviews). However, the questions asked by the two remaining participants steadily evolved as they perused the data items one by one; for example, Participant 3 only asked about sound quality for the first three TV data records ("How is the sound quality of this TV?"), and from the fourth record, the questions started

eliciting more details ("Does this TV support Dolby Atmos, and if so, how would you describe its overall sound quality?"). This observation indicates that as users gained more confidence in InstaFetch, they started exploring more of its capabilities by diversifying their queries.

We examined the responses generated by InstaFetch for the 40 queries and manually computed the error rate[18]. Specifically, two authors independently examined the query-response pairs along with the corresponding webpage context and determined if the response was correct or wrong. The annotations of the two authors were then examined for inter-annotator agreement, which was 0.81 (Cohen Kappa), thereby indicating high accordance. All authors then debated the mismatching annotations and settled on a final annotation through majority voting. The overall accuracy of InstaFetch in generating responses to participants' queries was 48.4%. From the collected recordings, we then analyzed the participants' reactions to the responses that were deemed incorrect. We observed that in 84.8% of these cases, the participants simply shrugged it off, as best expressed by P3: "*It is just like my Siri saying something weird at times, let me try asking in a different way*". In the remaining 15.2% cases, the involved participants were visibly annoyed by the incorrect responses. In 64.7% of the inaccurate cases, the participants re-issued their queries in a slightly different manner, whereas in the remaining 35.3% of cases, the participants simply navigated back to the other segments on the InstaFetch interface to manually search for the desired information on their own.

4.6 Queries vs. Interface Access

Through analyzing screen-reader logs and video recordings, we discerned the significant influence of core components of InstaFetch on user interaction. The capability for direct query input stood out as a notably impactful feature, utilized by 12 out of 14 participants. While only 2 participants exclusively used the InstaFetch's instant access feature for checking out detailed product information, preferring it over the query option to meticulously review products, a majority of those who employed the query feature also engaged with the instant access feature.

Overall, the study revealed a unanimous agreement among most participants that unfamiliarity with new websites typically leads to navigational challenges, resulting in a preference for a limited number of familiar online shopping platforms. This restricts their exposure to potentially attractive deals and products available on other sites. However, with the introduction of InstaFetch, users reported the ability to freely navigate through various web pages, obtaining the necessary information without the burden of extensive searching.

4.6.1 System Usability Scale (SUS) and Perceived Workload. The System Usability Scale (SUS) questionnaire [11] involved the participants rating a series of alternating positive and negative Likert items on a scale from 1 to 5, with 1 indicating strong disagreement, 3 representing a neutral response, and 5 signifying strong agreement. These responses are then aggregated into a single usability score between 0 to 100, with higher scores indicate more favorable usability ratings. Figure 5 displays the SUS statistics for the three study conditions. The SUS ratings for the InstaFetch condition (Average = 80.25, Standard Deviation = 12.52) were significantly higher than those for both the screen reader (Average = 47.5, Standard Deviation = 7.98) and SaIL (Average = 62, Standard Deviation = 13.07) conditions (one-way Anova test, $F = 18.57$, $p < 0.001$).

The NASA-TLX (NASA Task Load Index) questionnaire [27] was used to evaluate the perceived workload as expressed by the participants in their responses to the questionnaire. NASA-TLX too generates a score between 0 and 100, however, lower TLX ratings indicate reduced workloads and therefore better performance. In our study, we observed a significant influence of the study conditions on the NASA-TLX scores, as confirmed by the results of the Anova test ($F = 19.26$,

[18]Examples and discussion on errors are provided in GitHub https://github.com/accessodu/InstaFetch.git

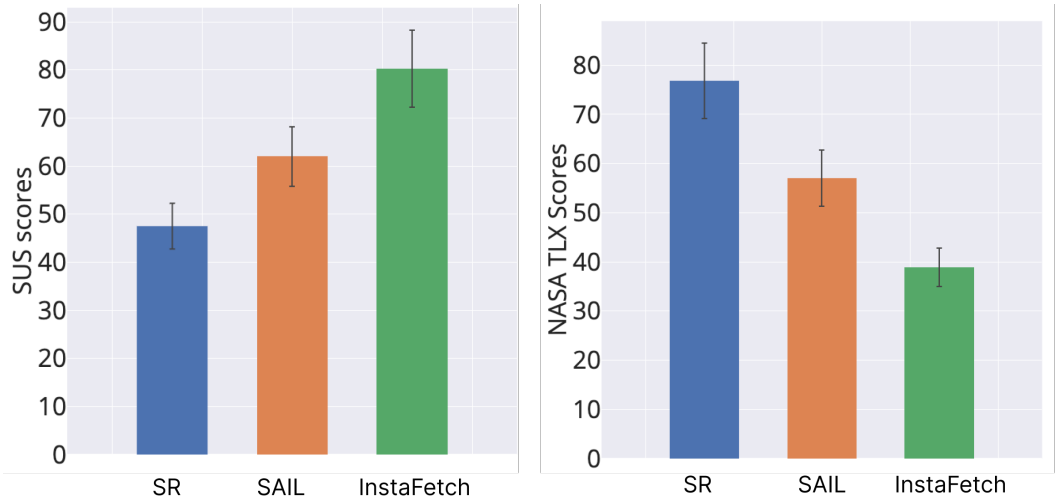

Fig. 5. Perceived usability (SUS) and task workload (NASA-TLX) for all three study conditions.

$p < 0.001$). Specifically, the TLX scores for the InstaFetch condition (Average = 38.96, Standard Deviation = 12.001) were significantly lower than those for both the SaIL condition (Average = 57.03, Standard Deviation = 11.69) and the Screen Reader condition (Average = 76.83, Standard Deviation = 14.89). This significant difference in TLX scores was further validated through pairwise comparisons using the post-hoc Tukey's HSD test, which showed that InstaFetch outperformed both screen reader ($Q = 8.78$, $p < 0.001$) and SaIL ($Q = 4.19$, $p = 0.016$).

The reasons underlying the participants' SUS and NASA-TLX ratings for the different study conditions were uncovered during the analysis of the participants' feedback in the open-ended exit interviews as explained next.

4.6.2 Qualitative feedback. The participants' feedback in the exit interviews were qualitatively analyzed using the standard open coding followed by axial coding method [47] where we iteratively went over the transcripts to identify recurring themes and insights that were common across multiple participants. A few notable ones are presented next.

To-and-fro navigation between pages is frustrating. Almost all (9) participants mentioned that navigating between different pages while searching for desired information was time-consuming and frustrating. Five participants further stated that having all item information in a single place, as facilitated by InstaFetch, substantially reduced the amount of listening, so it helped better retain item-related information in their memories while perusing the list of items. This explains why there were fewer revisits to items in the InstaFetch condition.

Navigation within an e-commerce webpage is also tedious and frustrating. A majority (7) of the participants also mentioned that locating different pieces of item-related information scattered within a webpage is often an arduous ordeal that involves plenty of listening to irrelevant content. These participants appreciated the SaIL's ability to avoid a bulk of the irrelevant content while navigating a webpage, however, they mentioned that they still had to navigate through at considerable amount of unrelated content such as menu, search form, company logo image, and quick links, all of which were determined as salient by SaIL. The participants mentioned that no such problem existed in InstaFetch, as only item-related information was present in its interface.

Searching for information within a page segment can sometimes be cumbersome. Half of the participants recommended adding additional navigation support in InstaFetch to tackle voluminous item-related segments. These participants mentioned that often item-related segments are too long, especially reviews (4 participants), and that they would prefer to listen to some kind of summaries, instead of navigating through the whole segment. The participants however did mention that InstaFetch support for natural language queries alleviated this burden to a large extent, but explained that as they usually look for different pieces of information within a segment, typing multiple queries, one for each desired information piece, can be a bit cumbersome.

Missing out on better deals due to interaction fatigue. All participants mentioned that while shopping online, they always experienced interaction fatigue due to which they could not cover enough items in the available list, thereby missing out on 'good deals'. The participants attributed this fatigue to a variety of factors including numerous shortcut presses, constant back-and-forth between webpages, and listening to too much content during navigation. Seven participants explicitly stated that InstaFetch mitigated these interaction burdens to a substantial extent, which enabled them to consider more items before making a decision.

Request for voice-based input and intelligent assistants. All participants suggested inclusion of voice-based querying support in InstaFetch. A few (3) participants even suggested extending natural language-querying to full-fledged conversations, where they could get engage in a dialog with InstaFetch to get all the desired information about any item in one exchange, as opposed to typing multiple queries one-by-one. Nonetheless, most participants stated that they would like to "keep" InstaFetch on their computers and a few of the participants also asked if InstaFetch was "free" to use.

5 DISCUSSION

The empirical findings from our user study demonstrate the capability of InstaFetch in notably enhancing both the browsing efficiency and overall user experience for blind screen reader users during their interactions with web data items on e-commerce platforms. However, the study also highlighted certain limitations, which paves the way for subsequent investigative endeavors in this field. A few notable ones are discussed next.

5.1 Limitations

A limitation of our study was that we focused on websites where both SaIL and InstaFetch algorithms demonstrated accurate extraction of segments from the 'Details' pages. While such a design choice was instrumental in avoiding confounding variables, it inadvertently prevented us from uncovering the repercussions of algorithmic inaccuracies on the user experience. The question of how blind participants would react and adapt to potential algorithmic errors in InstaFetch is the scope of future work. The second limitation of InstaFetch is that it presently supports only typed natural language queries. As mentioned earlier, several participants expressed a desire for voice-based querying and a few even suggested incorporating intelligent dialog agents in InstaFetch. The third limitation of our work is that InstaFetch currently supports only English-language websites and queries. Integrating regional language support is one of our future research objectives, and we aspire to attract a broader and more diverse user base.

Another inherent limitation of InstaFetch is that it was designed and tested only for e-commerce shopping websites, its usefulness in other types of websites is yet to be explored. Given the modular and easily generalizable architecture of InstaFetch, future research could focus on expanding its capabilities to different genres of websites, such as news portals, blogs, and educational sites. Moreover, when a user's query is ambiguous, the LLM-based module in InstaFetch may find it challenging to deliver an accurate response despite sophisticated prompt engineering. Addressing

such ambiguities in queries will be crucial moving forward. Additionally, while we mitigated the bias while evaluating the customized LLM through independent annotations and reviews by the entire research team, we admit that there may be a small possibility of unforeseen bias due to the lack of external personnel in the evaluation. Validating by findings by using external evaluators is in scope of our future work on this topic.

Lastly, to drive broader acceptance of InstaFetch, it is imperative to ensure compatibility with prominent screen readers and internet browsers. A potential way to achieve this is to formulate screen-reader plugins or multi-browser extensions to smoothly fuse InstaFetch's features with all mainstream browsing platforms. Currently, InstaFetch is tailored for desktop and laptop settings, leaving the mobile user segment unsupported. Given the pervasive nature of smartphones and the increasing trend of mobile-based e-commerce activities, facilitating efficient non-visual web interactions becomes crucial on smart mobile devices. Considering these considerations, we look forward to creating InstaFetch alternatives for mobile browsers.

5.2 Enabling Skimming-based Navigation.

While InstaFetch provided instant access to item-related segments, such as reviews, product specifications, and shipping details, navigating the content with these segments can sometimes be challenging and tedious, especially user reviews that can be extremely long. Sighted individuals have the advantage of swiftly skimming visual data, thereby capturing the essence of the vast content in reviews. In contrast, blind users have to navigate content serially via auditory interfaces using keyboard shortcuts, which can be very tedious and burdensome, as shown in our results. Therefore, we are considering extending InstaFetch to include advanced skimming support (e.g., [2]). Our objective is to enable blind users to swiftly gauge the relevance of content, enabling them to judiciously decide on whether to delve deeper into the content.

5.3 Societal Impact.

The usability of web interactions is critical for visually impaired individuals, yet most web designs cater to sighted users, imposing extra burdens on blind users. This imbalance creates a usability gap that limits the web's benefits for those with visual disabilities, unlike their sighted counterparts. Typically, sighted users leverage visual cues on a webpage to swiftly locate specific product information, utilizing spatial orientation to inform their content search. Conversely, blind users are required to sequentially traverse the DOM to access the information they need using keyboard shortcuts. In this paper, we address this disparity by enabling users to directly input queries and receive immediate responses, thereby streamlining the information retrieval process for visually impaired users. Moreover, we enable a more streamlined, effortless, and user-friendly interaction with information about web data items, thereby empowering blind users to explore a greater number of data items within a shorter timeframe and with reduced effort, thereby enhancing their ability to secure advantageous shopping *deals* comparable to those achieved by their sighted peers in shopping websites.

6 CONCLUSION

The prevailing distributed visualization of information related to web data items over multiple segments and multiple webpages predominantly cater to the preferences and convenience of sighted individuals. For blind users, this configuration induces a tedious and frustrating navigation exercise, marked by frequent toggling between multiple pages and inevitably encountering a myriad of irrelevant content. Addressing this glaring usability divide, we introduced InstaFetch, a novel browser extension tailored specifically for visually impaired users, aiming to centralize access to critical item information. This extension seamlessly aggregates essential data like product descriptions,

specifications, and reviews into a single screen reader-friendly interface that is accessible with a single user action. Furthermore, InstaFetch enables users to pose queries for specific item-related information and obtain relevant answers excavated from the webpage as the context. In a user study involving 14 blind participants, InstaFetch significantly outperformed both the status quo screen reader and a state-of-the-art solution.

REFERENCES

[1] Waleed Abdulla. 2017. Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. https://github.com/matterport/Mask_RCNN.

[2] Faisal Ahmed, Yevgen Borodin, Andrii Soviak, Muhammad Islam, IV Ramakrishnan, and Terri Hedgpeth. 2012. Accessible skimming: faster screen reading of web pages. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. 367–378.

[3] Julian Alarte, David Insa, and Josep Silva. 2017. Webpage menu detection based on DOM. In *International Conference on Current Trends in Theory and Practice of Informatics*. Springer, 411–422.

[4] Manuel Álvarez, Alberto Pan, Juan Raposo, Fernando Bellas, and Fidel Cacheda. 2010. Finding and extracting data records from web pages. *Journal of Signal Processing Systems* 59, 1 (2010), 123–137.

[5] 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.

[6] 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.

[7] Shrabastee Banerjee, Chrysanthos Dellarocas, and Georgios Zervas. 2021. Interacting user-generated content technologies: How questions and answers affect consumer reviews. *Journal of Marketing Research* 58, 4 (2021), 742–761.

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

[9] 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)*. Association for Computing Machinery, New York, NY, USA, Article 13, 10 pages. https://doi.org/10.1145/1805986.1806005

[10] 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

[11] John Brooke. 1996. Sus: a "quick and dirty'usability. *Usability evaluation in industry* 189, 3 (1996).

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[13] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2003. Vips: a vision-based page segmentation algorithm. (2003).

[14] Kaushik Chakrabarti, Zhimin Chen, Siamak Shakeri, and Guihong Cao. 2020. Open domain question answering using web tables. *arXiv preprint arXiv:2001.03272* (2020).

[15] Shiqian Chen, Chenliang Li, Feng Ji, Wei Zhou, and Haiqing Chen. 2019. Driven answer generation for product-related questions in e-commerce. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 411–419.

[16] Yang Deng, Wenxuan Zhang, and Wai Lam. 2020. Opinion-aware answer generation for review-driven question answering in e-commerce. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 255–264.

[17] Yang Deng, Wenxuan Zhang, Qian Yu, and Wai Lam. 2023. Product Question Answering in E-Commerce: A Survey. *arXiv preprint arXiv:2302.08092* (2023).

[18] Prasad M Deshpande, Karthikeyan Ramasamy, Amit Shukla, and Jeffrey F Naughton. 1998. Caching multidimensional queries using chunks. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. 259–270.

[19] Yixiang Fang, Xiaoqin Xie, Xiaofeng Zhang, Reynold Cheng, and Zhiqiang Zhang. 2018. STEM: a suffix tree-based method for web data records extraction. *Knowledge and Information Systems* 55, 2 (2018), 305–331.

[20] 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*. 49–62.

[21] Shen Gao, Xiuying Chen, Zhaochun Ren, Dongyan Zhao, and Rui Yan. 2021. Meaningful answer generation of e-commerce question-answering. *ACM Transactions on Information Systems (TOIS)* 39, 2 (2021), 1–26.

[22] Shen Gao, Zhaochun Ren, Yihong Zhao, Dongyan Zhao, Dawei Yin, and Rui Yan. 2019. Product-aware answer generation in e-commerce question-answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 429–437.
[23] Boni García, Mario Munoz-Organero, Carlos Alario-Hoyos, and Carlos Delgado Kloos. 2021. Automated driver management for selenium WebDriver. *Empirical Software Engineering* 26, 5 (2021), 1–51.
[24] Ross Girshick. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 1440–1448.
[25] 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*. 1–12.
[26] Tomas Gogar, Ondrej Hubacek, and Jan Sedivy. 2016. Deep neural networks for web page information extraction. In *Artificial Intelligence Applications and Innovations: 12th IFIP WG 12.5 International Conference and Workshops, AIAI 2016, Thessaloniki, Greece, September 16-18, 2016, Proceedings 12*. Springer, 154–163.
[27] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.
[28] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 2961–2969.
[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[30] Warut Khern-am nuai, Hossein Ghasemkhani, Dandan Qiao, and Karthik Kannan. 2023. The impact of online Q&As on product sales: The case of Amazon answer. *Information Systems Research* (2023).
[31] Anurendra Kumar, Keval Morabia, Jingjin Wang, Kevin Chen-Chuan Chang, and Alexander Schwing. 2021. CoVA: context-aware visual attention for webpage information extraction. *arXiv preprint arXiv:2110.12320* (2021).
[32] Eduardo Sany Laber, Críston Pereira de Souza, Iam Vita Jabour, Evelin Carvalho Freire de Amorim, Eduardo Teixeira Cardoso, Raúl Pierre Rentería, Lúcio Cunha Tinoco, and Caio Dias Valentim. 2009. A fast and simple method for extracting relevant content from news webpages. In *Proceedings of the 18th ACM conference on Information and knowledge management*. 1685–1688.
[33] 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.
[34] Hae-Na Lee and Vikas Ashok. 2022. Customizable Tabular Access to Web Data Records for Convenient Low-Vision Screen Magnifier Interaction. *ACM Transactions on Accessible Computing (TACCESS)* (2022).
[35] Hae-Na Lee, Sami Uddin, and Vikas Ashok. 2020. TableView: Enabling Efficient Access to Web Data Records for Screen-Magnifier Users. In *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility*. 1–12.
[36] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.
[37] Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023. How Long Can Context Length of Open-Source LLMs truly Promise?. In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.
[38] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2117–2125.
[39] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172* (2023).
[40] 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.
[41] Carol Moser, Chanda Phelan, Paul Resnick, Sarita Y Schoenebeck, and Katharina Reinecke. 2017. No such thing as too much chocolate: evidence against choice overload in e-commerce. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 4358–4369.
[42] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. 2022. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005* (2022).
[43] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.
[44] Pinecone. 2023. LangChain Unleashed. https://www.pinecone.io/learn/chunking-strategies/.
[45] Yash Prakash, Mohan Sunkara, Hae-Na Lee, Sampath Jayarathna, and Vikas Ashok. 2023. AutoDesc: Facilitating Convenient Perusal of Web Data Items for Blind Users. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. 32–45.

[46] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446* (2021).
[47] Johnny Saldaña. 2021. *The coding manual for qualitative researchers*. sage.
[48] Weishi Shi, Heather Moses, Qi Yu, Samuel Malachowsky, and Daniel E Krutz. 2023. ALL: Supporting Experiential Accessibility Education and Inclusive Software Development. *ACM Transactions on Software Engineering and Methodology* (2023).
[49] Brijendra Singh and Hemant Kumar Singh. 2010. Web data mining research: a survey. In *2010 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE, 1–10.
[50] Ray Smith. 2007. An overview of the Tesseract OCR engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, Vol. 2. IEEE, 629–633.
[51] Amanda Stent, Matthew Marge, and Mohit Singhai. 2005. Evaluating evaluation methods for generation in the presence of variation. In *International conference on intelligent text processing and computational linguistics*. Springer, 341–351.
[52] The GIMP Development Team. 1998. GNU Image Manipulation Program. https://www.gimp.org
[53] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
[54] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.
[55] 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.
[56] Canhui Xu, Cao Shi, Hengyue Bi, Chuanqi Liu, Yongfeng Yuan, Haoyan Guo, and Yinong Chen. 2021. A Page Object Detection Method Based on Mask R-CNN. *IEEE Access* 9 (2021), 143448–143457.
[57] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).
[58] Qian Yu, Wai Lam, and Zihao Wang. 2018. Responding e-commerce product questions via exploiting qa collections and reviews. In *Proceedings of the 27th International Conference on Computational Linguistics*. 2192–2203.
[59] Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Li Yun, Hejie Cui, Zhang Xuchao, Tianjiao Zhao, et al. 2023. Domain specialization as the key to make large language models disruptive: A comprehensive survey. *arXiv preprint arXiv:2305.18703* (2023).
[60] Zhuoyao Zhong, Lei Sun, and Qiang Huo. 2019. An anchor-free region proposal network for Faster R-CNN-based text detection approaches. *International Journal on Document Analysis and Recognition (IJDAR)* 22, 3 (2019), 315–327.

A INSTAFETCH ILLUSTRATION

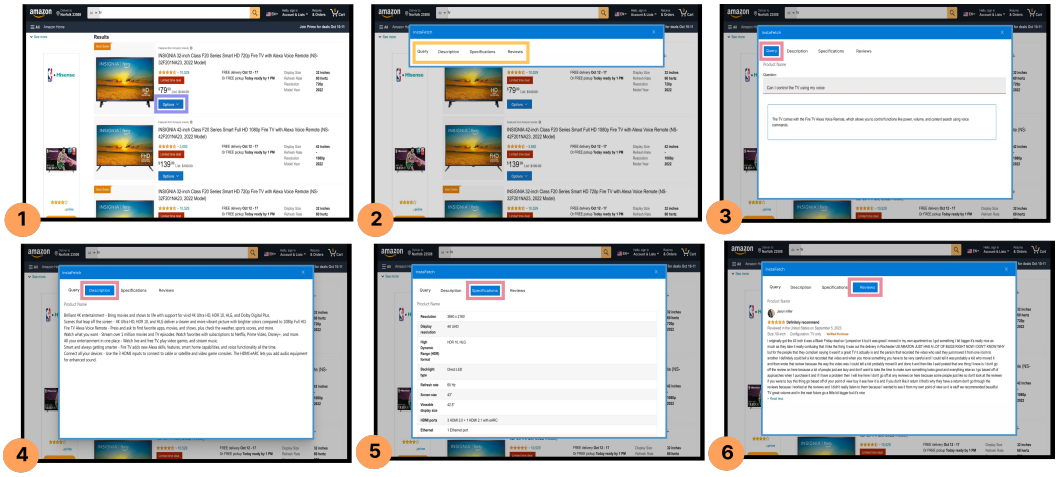

Fig. 6. InstaFetch illustration. Upon selection of the 'options' button accompanying any data record (1), users can activate a quartet of choices displayed via a pop-up interface (2). The primary feature of this pop-up is a 'query' function, which provides users with the capability to submit a question and receive an immediate answer pertinent to their inquiry (3). Complementing this feature are three additional options—'description' (4), 'specifications' (5), and 'reviews' (6)—which collectively furnish users with a comprehensive synopsis of the product's detail page. This functionality affords users the convenience of an expedited, in-situ overview of product specifics, circumventing the necessity for page redirection to obtain this information.

Received February 2024; revised April 2024; accepted April 2024