Tab to Autocomplete: The Effects of AI Coding Assistants on Web Accessibility

Peya Mowar
pmowar@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Yi-Hao Peng
yihaop@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Aaron Steinfeld
steinfeld@cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Jeffrey P. Bigham
jbigham@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

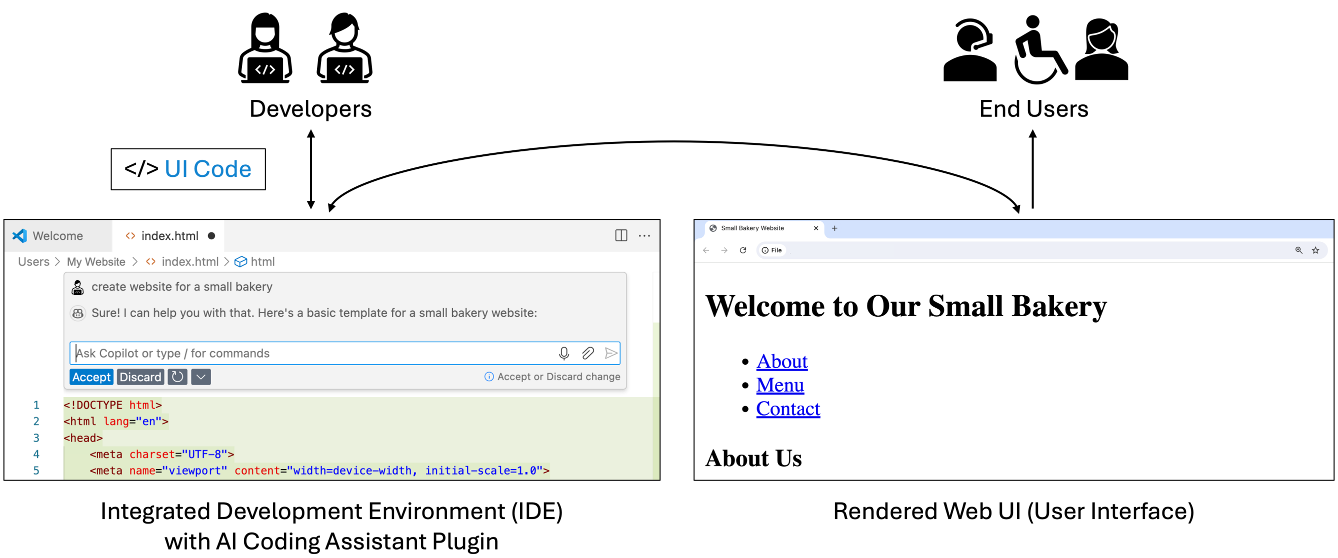

Figure 1: Workflow of AI-assisted Web Development: Developers use AI coding assistants to produce UI code, which is rendered as a web UI and accessed by users with different accessibility needs. We consider the accessibility of the rendered UI as a function of the UI code produced and refer to it as "code accessibility".

ABSTRACT

A long-standing challenge in accessible computing has been to get developers to produce the accessible UI code necessary for assistive technologies to work properly. AI coding assistants (*e.g.*, Github Copilot) potentially offer a new opportunity to make UI code more accessible automatically, but it is unclear how their use impacts code accessibility and what developers need to know in order to use them effectively. In this paper, we report on a study where developers untrained in accessibility were tasked with building web UI components with and without an AI coding assistant. Our findings suggest that while current AI coding assistants show potential for creating more accessible UIs, they currently require accessibility awareness and expertise, limiting their expected impact.

CCS CONCEPTS

• **Human-centered computing → Accessibility design and evaluation methods**; **Interactive systems and tools**; • **Software and its engineering → Development frameworks and environments**.

KEYWORDS

AI Coding Assistants, Web Accessibility, Empirical Studies in HCI

ACM Reference Format:
Peya Mowar, Yi-Hao Peng, Aaron Steinfeld, and Jeffrey P. Bigham. 2024. Tab to Autocomplete: The Effects of AI Coding Assistants on Web Accessibility. In *The 26th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '24), October 27–30, 2024, St. John's, NL, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3663548.3688513

ACM ISBN 979-8-4007-0677-6/24/10
https://doi.org/10.1145/3663548.3688513

1 INTRODUCTION

A long-standing challenge in accessible computing has been to get developers to produce the accessible user interface code necessary for assistive technologies to work properly. The 2024 WebAIM study of the top million web pages found that the average homepage had 56.8 errors, defined as "accessibility barriers having notable end-user impact" [41], a number that has remained stubbornly high despite the substantial effort put into accessibility standards development [5, 6, 11, 28], developer tool integration [4, 21, 38], legal requirements [33, 42], and developer advocacy [12, 35].

AI coding assistants, such as GitHub Copilot, represent a significant breakthrough in programming. The rapid advancement of language models has revolutionized the way individuals explore, interpret, and edit code. By leveraging the power of large language models (LLMs), the programming assistants provide code suggestions and automate routine programming tasks, saving substantial time and making programming more accessible to beginners. The widespread adoption of these assistants underscores their importance in the evolving landscape of software development [18].

Since developers are using AI assistants, if those assistants produce accessible code, then we might presume that the code that developers write would be more accessible. Yet, several aspects of the line of reasoning have been unclear. First, it is unclear whether AI coding assistants have been trained to produce accessible code. Indeed, most assistant are trained on publicly-available user interface code that we know to contain substantial accessibility problems [9]. Second, it is unclear what developers need to know about accessibility to effectively use AI assistants to produce accessible code. For example, if developers have to explicitly instruct AI assistants to generate accessible code (i.e., consider accessibility), the benefit would be limited to those who are already aware of accessibility needs. We know that AI-assisted coding can still require substantial programming knowledge for effective usage, but the level of accessibility expertise required to benefit from any in-built accessibility capabilities in the assistants remains unclear.

To investigate these issues, we designed and ran a study in which 16 developers untrained in accessibility were tasked with building web user interface components with an AI coding assistant. Our results suggest that AI coding assistants can produce accessible code, but developers still need accessibility expertise to make use of them effectively. Otherwise, the accessibility introduced is likely to not be applied comprehensively, advanced features recommended by the assistant are unlikely to be implemented, and accessibility errors introduced by the assistant are unlikely to be caught. These results suggest that future work could usefully engage with how to make AI coding assistants better at producing accessible code and how to reduce the developer awareness and expertise required to benefit from the accessibility coding assistance. Our work provides a first step in understanding the limitations of the current approaches and some approaches for potentially overcoming them.

2 RELATED WORK

Work related to this paper includes *(i)* Web Accessibility *(ii)* Developer Practices in AI-Assisted Programming.

Web Accessibility: Practice, Evaluation, and Improvements. Various efforts have been made to set accessibility standards [6, 11], establish legal requirements [33, 42], and promote education and advocacy among developers [22, 24, 35]. In the research domain, several methods have been developed to assess and enhance web accessibility. These include incorporating feedback into developer tools [4, 38, 39] and automating the creation of accessibility tests and reports for UIs [36, 37]. However, a persistent challenge is that developers need to be aware of these tools to utilize them effectively. With recent advancements in LLMs, developers might now build accessible UIs with less effort using AI assistants. However, the impact of these assistants on the accessibility of their generated code remains unclear. This study aims to investigate these effects.

Developer Practices in AI-Assisted Programming. Recent usability research on AI-assisted development has examined the interaction strategies of developers while using AI Coding Assistants [3]. They observed developers interacted with these assistants in two modes – 1) *acceleration mode*: associated with shorter completions and 2) *exploration mode*: associated with long completions. [18] found that developers are driven to use AI assistants to reduce their keystrokes, finish tasks faster, and recall the syntax of programming languages. On the other hand, developers' reason for rejecting autocomplete suggestions was the need for more consideration of appropriate software requirements. This is because primary research on code generation models has mainly focused on functional correctness while often sidelining non-functional requirements such as latency, maintainability, and security [34]. Consequently, there have been increasing concerns about the security implications of AI-generated code [31]. Similarly, this study focuses on the effectiveness and uptake of code suggestions among developers in mitigating accessibility-related vulnerabilities.

3 METHODOLOGY

To explore the impact of AI coding assistants on code accessibility, we conducted a user study with 16 web developers. This in-person study spanned about 90 minutes and received approval from the Institutional Review Board (IRB). Each participant received a $30 Amazon gift card as reimbursement for their time.

Participants. We recruited 16 participants who have web development experience (7 female and 9 male; ages ranged from 22 to 29) via social media and university mailing groups. Nearly all our participants were students and had multi-year programming experience (except one student who had around one year of experience). 10 of our participants had multi-year *industrial* programming experience (e.g., full-time or intern experiences in the company).

Materials and Tasks. We selected two real-world websites, Kubernetes [29] and BBC Simorgh [23], as our study materials. These websites receive over 2 million monthly visits worldwide [2], belong to different categories in the IAB Content Taxonomy [1], and differ in how accessible they are. To design our tasks, we sampled actual issues from each website's repository on Github. One task involved a general feature request, while the other focused on enhancing the user interface for improved accessibility. Performing our tasks involved consideration of several common web accessibility issues (e.g., color contrast, alternative text, form labeling) [41].

Study Procedure. Participants were assigned tasks related to the two selected websites, with a total of four tasks to complete. To

Table 1: Manual Evaluation Criteria for Web Accessibility

Task Category	Evaluation Criteria
Adding alt-text	*Unacceptable*: Missing or uninformative [27] alt-text *Needs Improvement*: Added alt-text with < 3 required descriptors [19] *Good*: Added alt-text with >= 3 out of 4 required descriptors
Button colour contrast	*Unacceptable*: contrast ratio of < 4.5:1 for normal text and < 3:1 for large text *Needs Improvement*: WCAG level AA: minimum contrast ratio of 4.5:1 for normal and 3:1 for large text *Good*: WCAG level AAA: minimum contrast ratio of 7:1 for normal text and 4.5:1 for large text
Form labeling	*Unacceptable*: Missing form labels and keyboard navigation *Needs Improvement*: One of form labels and keyboard navigation *Good*: Both form labels and keyboard navigation
Link labeling	*Unacceptable*: Missing or uninformative [30] link descriptions *Needs Improvement*: Somewhat descriptive links [27] *Good*: Descriptive link labels

replicate real-world scenarios where web developers often prioritize functional requirements over accessibility unless explicitly required [17], the study's true purpose was not disclosed. Participants were informed that the study was about the usability of AI pair programmers in web development tasks but were not explicitly instructed to make their web components accessible. The study followed a within-subject design. To counterbalance the order effect, participants were assigned to one of four orderings, covering all possible combinations of website order and Copilot usage. Participants were also allowed access the web for task exploration or code documentation. After completing the tasks, they were asked to complete a post-task survey inquiring about their development expertise, experience in web accessibility, and open-ended feedback.

Data Collection and Analysis. We captured the entire study sessions through screen recordings, resulting in about 19 hours of video data. We complemented this with observational notes taken during the sessions, documenting verbal comments made by participants. The participants' interactions with Copilot Chat were also recorded for further analysis between prompts and the final code. We also collected AI usage, programming languages and framework preferences, and expertise in web accessibility via a post-task survey. We manually inspected the websites created during the study and evaluated their accessibility on a qualitative scale of 'Unacceptable', 'Needs Improvement', and 'Good' adopted from prior research published in CHI and ASSETS, detailed further in Table 1.

4 RESULTS

We present the findings by showing participants' previous experiences and the overall accessibility of the revised code.

Prior Experience with AI Coding Assistants and Accessible UI Development. From the post-task survey, we found that nearly all participants (except one) had previously used AI coding assistants, with GitHub Copilot and OpenAI ChatGPT being the most popular choices among 10 participants. Other assistants the participants had used were Tabnine (N = 6) and AWS CodeWhisperer (N = 2). As for web development skills, 12 participants had substantial experience with HTML and CSS, 10 were proficient in JavaScript and 7 were proficient in React.js. Despite this expertise, the majority (N = 14) were unfamiliar with the Web Content Accessibility Guidelines (WCAG). Only 2 participants knew about these guidelines, but even they had not actively engaged in creating accessible web user interfaces or received formal training on the subject.

Impact of AI Assistants on Website Accessibility: Observations and Insights. The use of AI coding assistants has shown mixed results in resolving some common accessibility issues, as illustrated in Figure 2. Notably, Copilot can effectively generate accessible components by utilizing patterns from other parts of a website. For example, it might automatically include proper labels for form fields, such as `<label for="email">Email:</label>` in a signup form. However, the effectiveness of these tools can be limited by the need for more sophisticated accessibility knowledge. Our study revealed that while developers generally rely on Copilot for syntax assistance and accelerated coding through its autocomplete features, their focus tends to be on productivity and efficiency rather than the accessibility of the output. This reliance sometimes leads to oversight of necessary accessibility considerations. For instance, despite prompts from Copilot to manually adjust image descriptions as highlighted in Table 2, participants frequently overlooked these suggestions, resulting in code submissions with empty `alt` attributes. Moreover, there were instances where Copilot inadvertently introduced new accessibility issues. For example, when adding new button components with hover effects, it failed to ensure adequate contrast between the button's hover color and its background. Some errors, such as determining whether an image is decorative or informative (and if an image is informative, alt-texts should be added), cannot be detected by automated accessibility checkers, indicating certain limitations in current automated evaluations. These findings underscore the dual role of AI coding assistants: while they can generate some accessible code, they can also overlook or introduce accessibility issues when the focus is primarily on feature development.

5 DISCUSSION

Enhancing Language Models for Generating Accessible UI Code. Our research shows that current AI programming assistants produce user interface code at varying levels of accessibility. To

Table 2: The (partial) chat history revealed that directly copying Copilot's suggestions would be incomplete, as developers overlooked the additional recommended steps.

User:	*"User Query"*
Copilot:	`Generated Code`
	Associated Text Explanation
User 1:	*"I want to label the images or add relevant image descriptors"*
Copilot:	`alt=imgAlt // Add this line`
	Remember to replace imageAlt, <...> with your actual image attributes. In this example, imageAlt is the alt text for the image, and <...>. You will need to replace these with the actual values.
User 2:	*"Can you optimise the images on this article page for search engine optimization. Refactor the code accordingly by labeling the images or adding relevant image descriptions."*
Copilot:	`imgAlt=imageDescription // Add this line`
	Sure, to optimize images for SEO, you can add alt attributes to your img tags. The alt attribute provides alternative information for an image if a user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

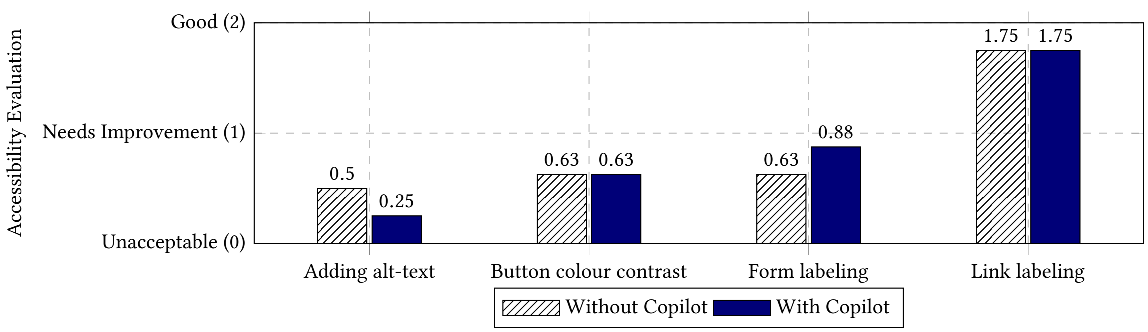

Figure 2: Mean Accessibility Evaluation Scores by Tasks and Copilot Usage: Higher scores indicate that participants were successful.

lower the barriers for novice developers in implementing accessible user interfaces, it's crucial to improve the underlying models of these coding assistants. Recent developments in self-refinement techniques for language models [8, 15, 20] offer a pathway to enhance these models. By fine-tuning them with accessible user interface examples gathered from extensive web crawling data [13, 16], along with employing a robust reward mechanism [10, 14] and additional visual (and other modality) understanding modules [32, 40], we can align and steer these models to more effectively generate accessible user interface code (e.g., code with proper aria-labels; images with high-quality alt-texts).

Building Programming Tools with Accessibility in Mind. Although automatically generating accessible user interfacesthrough computational models can simplify the process for creators, a key aspect of making digital content accessible involves increasing creators' awareness of accessibility from the start [25, 26, 38]. A model that produces accessible code can also provide feedback or guidance tool for humans in an AI-assisted co-programming environment. For example, instead of delivering fully accessible user interface code all at once, incorporating interactive elements with customizable attributes into current programming tools could help users learn during the process. This may increase their awareness of accessibility issues and also help them address potential flaws or enable more personalization in AI-generated code (e.g., customizing the color contrast for buttons with hover effects). This feedback may also be applied in the opposite direction [7], enabling models to improve through ongoing interaction and input.

Study Limitations. The primary limitation of our study is that our student participants were mostly recruited from the same university and may not capture the full spectrum of developer experiences. Additionally, the brief duration of our study may not accurately represent long-term real-world interactions with AI Coding Assistants; extended study periods could potentially unveil more comprehensive insights into users' ongoing engagement and challenges. While our study provides important insights into the accessibility and awareness of AI Coding Assistants, caution should be exercised when extending these findings to the broader developer community.

6 CONCLUSION

In this paper, we have presented the results of a study of the effects of AI coding assistance on web accessibility. Our results suggest that AI coding assistants have potential to improve the accessibility of code that developers produce, but a remaining challenge is that developers still need to have expertise in accessibility to use these tools effectively.

REFERENCES

[1] 2024. IAB Website Categories. https://docs.webshrinker.com/v3/iab-website-categories.html#iab-categories. Accessed: 2024-04-22.

[2] 2024. SimilarWeb - Website Traffic & Market Intelligence. http://similarweb.com. Accessed: 2024-04-22.

[3] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.

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

[5] Giorgio Brajnik, Yeliz Yesilada, and Simon Harper. 2010. Testability and validity of WCAG 2.0: the expertise effect. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*. 43–50.

[6] Ben Caldwell, Michael Cooper, Loretta Guarino Reid, Gregg Vanderheiden, Wendy Chisholm, John Slatin, and Jason White. 2008. Web content accessibility guidelines (WCAG) 2.0. *WWW Consortium (W3C)* 290, 1-34 (2008), 5–12.

[7] Sabrina Caldwell, Penny Sweetser, Nicholas O'Donnell, Matthew J Knight, Matthew Aitchison, Tom Gedeon, Daniel Johnson, Margot Brereton, Marcus Gallagher, and David Conroy. 2022. An agile new research framework for hybrid human-AI teaming: Trust, transparency, and transferability. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 12, 3 (2022), 1–36.

[8] Angelica Chen, Jérémy Scheurer, Tomasz Korbak, Jon Ander Campos, Jun Shern Chan, Samuel R Bowman, Kyunghyun Cho, and Ethan Perez. 2023. Improving code generation by training with natural language feedback. *arXiv preprint arXiv:2303.16749* (2023).

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[10] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128* (2023).

[11] Wendy Chisholm, Gregg Vanderheiden, and Ian Jacobs. 2001. Web content accessibility guidelines 1.0. *Interactions* 8, 4 (2001), 35–54.

[12] Martyn Cooper, David Sloan, Brian Kelly, and Sarah Lewthwaite. 2012. A challenge to web accessibility metrics and guidelines: putting people and processes first. In *Proceedings of the international cross-disciplinary conference on Web accessibility*. 1–4.

[13] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. 2021. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758* (2021).

[14] Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. 2023. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998* (2023).

[15] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2024. Language models can solve computer tasks. *Advances in Neural Information Processing Systems* 36 (2024).

[16] Hugo Laurençon, Lucile Saulnier, Léo Tronchon, Stas Bekman, Amanpreet Singh, Anton Lozhkov, Thomas Wang, Siddharth Karamcheti, Alexander M. Rush, Douwe Kiela, Matthieu Cord, and Victor Sanh. 2023. OBELICS: An Open Web-Scale Filtered Dataset of Interleaved Image-Text Documents. arXiv:2306.16527 [cs.IR]

[17] Jonathan Lazar, Alfreda Dudley-Sponaugle, and Kisha-Dawn Greenidge. 2004. Improving web accessibility: a study of webmaster perceptions. *Computers in human behavior* 20, 2 (2004), 269–288.

[18] Jenny T Liang, Chenyang Yang, and Brad A Myers. 2024. A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[19] Kelly Mack, Edward Cutrell, Bongshin Lee, and Meredith Ringel Morris. 2021. Designing Tools for High-Quality Alt Text Authoring. In *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility* (<conf-loc>, <city>Virtual Event</city>, <country>USA</country>, </conf-loc>) *(ASSETS '21)*. Association for Computing Machinery, New York, NY, USA, Article 23, 14 pages. https://doi.org/10.1145/3441852.3471207

[20] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2024).

[21] Jennifer Mankoff, Holly Fait, and Tu Tran. 2005. Is your web page accessible? A comparative study of methods for assessing web page accessibility for the blind. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 41–50.

[22] Lilu Martin, Catherine Baker, Kristen Shinohara, and Yasmine N Elglaly. 2022. The Landscape of Accessibility Skill Set in the Software Industry Positions. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*. 1–4.

[23] BBC News. 2024. BBC Home - Breaking News, World News, US News, Sports ... https://www.bbc.com/

[24] Maulishree Pandey and Tao Dong. 2023. Blending Accessibility in UI Framework Documentation to Build Awareness. In *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility*. 1–12.

[25] Amy Pavel, Gabriel Reyes, and Jeffrey P Bigham. 2020. Rescribe: Authoring and automatically editing audio descriptions. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 747–759.

[26] Yi-Hao Peng, JiWoong Jang, Jeffrey P Bigham, and Amy Pavel. 2021. Say it all: Feedback for improving non-visual presentation accessibility. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–12.

[27] Athira Pillai, Kristen Shinohara, and Garreth W Tigwell. 2022. Website builders still contribute to inaccessible web design. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*. 1–4.

[28] Christopher Power, André Freire, Helen Petrie, and David Swallow. 2012. Guidelines are only half of the story: accessibility problems encountered by blind users on the web. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 433–442.

[29] Kubernetes Project. 2024. Kubernetes. https://kubernetes.io/

[30] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O Wobbrock. 2018. Examining image-based button labeling for accessibility in Android apps through large-scale analysis. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*. 119–130.

[31] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. 2023. Lost at c: A user study on the security implications of large language model code assistants. *arXiv preprint arXiv:2208.09727* (2023).

[32] Pratyusha Sharma, Tamar Rott Shaham, Manel Baradad, Stephanie Fu, Adrian Rodriguez-Munoz, Shivam Duggal, Phillip Isola, and Antonio Torralba. 2024. A Vision Check-up for Language Models. *arXiv preprint arXiv:2401.01862* (2024).

[33] Brian Sierkowski. 2002. Achieving web accessibility. In *Proceedings of the 30th annual ACM SIGUCCS conference on User services*. 288–291.

[34] Manav Singhal, Tushar Aggarwal, Abhijeet Awasthi, Nagarajan Natarajan, and Aditya Kanade. 2024. NoFunEval: Funny How Code LMs Falter on Requirements Beyond Functional Correctness. *arXiv preprint arXiv:2401.15963* (2024).

[35] David Sloan, Andy Heath, Fraser Hamilton, Brian Kelly, Helen Petrie, and Lawrie Phipps. 2006. Contextual web accessibility-maximizing the benefit of accessibility guidelines. In *Proceedings of the 2006 international cross-disciplinary workshop on Web accessibility (W4A): Building the mobile web: rediscovering accessibility?* 121–131.

[36] Amanda Swearngin, Jason Wu, Xiaoyi Zhang, Esteban Gomez, Jen Coughenour, Rachel Stukenborg, Bhavya Garg, Greg Hughes, Adriana Hilliard, Jeffrey P Bigham, et al. 2023. Towards Automated Accessibility Report Generation for Mobile Apps. *arXiv preprint arXiv:2310.00091* (2023).

[37] Maryam Taeb, Amanda Swearngin, Eldon School, Ruijia Cheng, Yue Jiang, and Jeffrey Nichols. 2023. Axnav: Replaying accessibility tests from natural language. *arXiv preprint arXiv:2310.02424* (2023).

[38] Hironobu Takagi, Chieko Asakawa, Kentarou Fukuda, and Junji Maeda. 2003. Accessibility designer: visualizing usability for the blind. *ACM SIGACCESS accessibility and computing* 77-78 (2003), 177–184.

[39] Hironobu Takagi, Chieko Asakawa, Kentarou Fukuda, and Junji Maeda. 2003. Accessibility designer: visualizing usability for the blind. *SIGACCESS Access. Comput.* 77–78 (sep 2003), 177–184. https://doi.org/10.1145/1029014.1028662

[40] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2016. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE transactions on pattern analysis and machine intelligence* 39, 4 (2016), 652–663.

[41] WebAIM. 2024. The WebAIM Million - The 2024 report on the accessibility of the top 1,000,000 home pages. https://webaim.org/projects/million/. Accessed: 2024-04-22.

[42] Yeliz Yesilada, Giorgio Brajnik, Markel Vigo, and Simon Harper. 2012. Understanding web accessibility and its drivers. In *Proceedings of the international cross-disciplinary conference on web accessibility*. 1–9.