

How Can Copilot Assist in Creating Accessible Websites?

An Empirical Study

Luciano A. Teran¹, Thayssa A. da Rocha^{1,2}, Ingrid M. M. da Silva¹, Rafael B. de Salles², Giselle L. N. Melo¹, Marcelle P. Mota¹, Cleidson R. B. de Souza¹

¹Institute of Exact and Natural Sciences (ICEN) – Federal University of Pará (UFPA)
66075-110 – Belém – Pará – Brazil

²Zup Innovation
38408-262 – Uberlândia – Minas Gerais – Brazil

{luciano.teran, thayssa.rocha, ingrid.silva, giselle.nobre}@icen.ufpa.br
salles.sistemas@gmail.com, mpmota@ufpa.br, cleidson.desouza@acm.org

Abstract. Introduction: Software developers have used intelligent coding assistants to aid their development efforts. **Objective:** We investigated GitHub Copilot's support for creating accessible software applications. We explored its capacity to generate code suggestions that adhere to accessibility best practices during the development of a web application. **Methodology:** We analyzed the results using an inspection tool and the content of development diaries. **Results:** Our results indicate that the assistant provides limited accessibility support, exhibiting three distinct behaviors: in some cases, it generates code that adheres to accessibility criteria autonomously, while in others, it requires detailed prompting. Furthermore, in some cases, it does not even provide accessibility support when explicitly requested by the developer. **Keywords** Digital accessibility, Software development, Intelligent assistant.

1. Introduction

Software Engineering research has, for decades, reflected a concern with developer productivity and the results generated through their excellent performance [Ozkaya 2023]. In recent years, software developers have raised discussions about Artificial Intelligence (AI) tools due to their ability to assist programmers in coding activities, potentially improving productivity and code quality [Bird et al. 2023, de Souza et al. 2024]. While several AI tools can be used to support software developers, in this paper we focus, through GitHub Copilot (Copilot), on intelligent coding assistants, i.e., tools that provide entire blocks of *code* suggestions for developers based on the prompts they enter.

Previous studies have focused on the *advantages* of the coding assistants (see for instance Bird et al. 2023, Ernst and Bavota 2022, and Liang et al. 2024). Meanwhile, other studies have also identified limitations of these tools, including the correctness, safety, and complexity of their code suggestions. However, there is little literature about accessibility and intelligent assistants [Acosta-Vargas et al. 2024, MoghadasNian et al. 2025, Guspian et al. 2024], specifically when addressing the implementation of accessibility guidelines in assistive code tools [Mowar et al. 2024, Mowar et al. 2025].

To increase digital accessibility, researchers have created guidelines that specify what developers should, or should not do when creating an accessible software applications [Kavcic 2005]. Professionals who follow accessibility guidelines, not only contribute to social inclusion by providing equal access and user satisfaction in the digital environment but also fulfill legal requirements in some cases [World Wide Web Consortium 2024a]. Countries like Brazil, the United States, Germany and Portugal have specific laws that guarantee digital accessibility rights to People with Disabilities (PwD) [Brazil 1991, Portugal 2004, Germany 2002, USA 2024].

Through the Web Accessibility Initiative (WAI), the Web Content Accessibility Guidelines (WCAG) were created to provide an internationally recognized reference for individuals, organizations, and governments regarding web content accessibility [World Wide Web Consortium 2024b], being now one of the most widely used sets of digital accessibility guidelines [Mateus et al. 2020]. Studying accessibility supported by intelligent coding assistants is important to determine whether and how these tools provide proper code suggestions and to find out if they can be perceived as a valuable aid for programmers in supporting digital accessibility efficiently and productively [Leite et al. 2021, Teran et al. 2021]. Due to the relevance of digital accessibility and the current lack of knowledge about the limitations of intelligent coding assistants in this matter, this paper aims to answer the following research question: “*How can Copilot assist in creating accessible websites?*”.

To address this question, an empirical study was conducted using GitHub Copilot to develop a web application based on the Next.js framework. The web application was developed by following predefined steps, divided into two phases: (i) in the first phase, we coded the application without requesting accessibility support from Copilot; that is, the researchers’ prompts did *not* express any concern regarding accessibility aspects, while (ii) in the second phase, Copilot was used to debug the code from the first phase in order to implement WCAG guidelines. At the end of each phase, the web application was submitted to an automated accessibility inspection tool. The results were analyzed in conjunction with the developers’ diaries, which were generated during the project phases.

Our results suggest that Copilot does not properly support developers in coding accessible software. For instance, it only automatically provided minor recommendations, such as using alternative text properties for images, without considering other minimum accessibility requirements described in WCAG. Furthermore, it just suggested code that promotes accessibility when the injected prompts specified the accessibility problem.

This paper has two primary contributions: to the best of our knowledge, it is the first paper to assess the accessibility aspects of GitHub Copilot’s code suggestions. Second, it provides a set of lessons that developers and researchers can apply to create accessible software applications with the assistance of GitHub Copilot. We corroborate reflections on the use of AI tools and their future implications in the field of HCI [Duarte et al. 2024]. As with other studies [Freire et al. 2024, da Silva et al. 2024, Borges e Araújo 2024], this research aims to understand the problems and opportunities associated with the support provided by intelligent assistants in professional settings.

The remainder of this paper is structured as follows: Section 2 presents an overview of digital accessibility and intelligent coding assistants. Section 3 explains the

research methods we adopted. This is followed by Sections 4 and 5, which detail our results and discussion. Finally, Section 6 presents our final remarks and future work.

2. Theoretical Framework

This section provides an overview of digital accessibility concepts and technical artifacts. It also discusses the use of intelligent assistants in the software development process.

2.1. Digital Accessibility

Accessibility is fundamental to ensure that people have safe and autonomous access, regardless of ability or circumstance, promoting participation in daily activities and increasing opportunities [Brazil 2009]. There is a wide range of accessibility dimensions; however, we focused on digital accessibility, which is defined as a quality criterion that eliminates barriers to accessing digital services and tools [Kulkarni 2019]. The greatest challenge in achieving digital accessibility lies in the current inability of technology to meet diverse human conditions [Kulkarni 2019]. Some accessibility approaches have been proposed to help designers and developers construct accessible software, such as ATAG [World Wide Web Consortium 2023a], UAAG [World Wide Web Consortium 2016], and Mobile A11y [Whitaker 2016].

WCAG, as one of these initiatives, presents a set of recommendations for making Internet content more accessible and has great relevance due to its use to drive diverse people's digital inclusion [World Wide Web Consortium 2024b, Silva et al. 2019]. There are various ways of evaluating whether developed content adheres to WCAG, such as through manual tests or automated inspection tools. However, few developers have practical accessibility experience and apply it to build and validate this requirement into their products [Antonelli et al. 2018].

Although some tools evaluate software *after* it has been coded, others can perform this accessibility evaluation *during* the coding and implementation phase, such as “ax Accessibility Linter”, “axe-cli”, and “Pa11y¹”. These evaluation tools focus on inspecting accessibility violations, i.e., issues in code that inflict one or more accessibility guidelines [Deque Systems 2024]. Whether the evaluation is performed after or during the software development phase, these tools rely on the users' previous knowledge of digital accessibility concepts. In this paper we will consider WCAG guidelines as our reference to assess web applications accessibility.

2.2. Intelligent Coding Assistants

This Subsection presents an overview of Large Language Models (LLMs) in software development, which has been popularized in code generation using prompts based on natural language, code translation, and automatic code insertion in real time [Ross et al. 2023].

Hou and colleagues [Hou et al. 2024] carried out a systematic review on the application of LLMs in software engineering and, among other vital data collected until 2023 April, they noted that around 56% of the papers documented the use of this technology to improve coding and software development processes. This review

¹Pa11y: open source tool to support designers and developers create web applications more accessible.

also emphasized intelligent coding assistants' central role in code generation tasks. It represented 45% of the papers about these assistants. In contrast, the second most reported activity (11%) supported by LLMs is assistance in finalizing code started by the programmer. In general, assistants improve the efficiency of code writing, helping software developers and promoting greater automation in the code development process [Bird et al. 2023, Ernst e Bavota 2022, Ziegler et al. 2022]. Copilot assisted with these two tasks in our research: code writing and code improvement.

Schmidt [Schmidt 2023] discusses the challenges of Generative AI (Gen AI) in coding and how its usage can accelerate and improve the testing and performance of the code generated. Its use in programming is a consequence of the evolution of how we can program computers. The use of Gen AI involves the prompt elaboration. Some authors reinforce the importance of *“developers' proficiency in prompt engineering to communicate effectively with LLMs and guide them toward desired outputs”* [Terragni et al. 2025] and even suggest templates for better prompting [Rajbhoj et al. 2024].

Despite the benefits that Gen AI has come to improve software engineering's future [Terragni et al. 2025], intelligent coding assistants have different limitations. For instance, they generate insecure code [Pearce et al. 2022] and incorrect code [Nguyen e Nadi 2022]. Another limitation of intelligent coding assistants is non-determinism [Ouyang et al. 2023]. This means that an LLM may propose alternative (code) outputs for the same prompt [Ouyang et al. 2023]. A solution to address this problem is to use the same prompt more than once and evaluate the different code outputs [Ouyang et al. 2023]. Another solution, although less effective [Ouyang et al. 2023], is to set the temperature parameter to zero. This parameter is used to control the randomness of the suggestions generated by the LLM varying between specific (0), balanced (1), or creative (2). In our study, we used temperature “zero” to *minimize* this issue.

Although LLMs trained for code generated (LLM4code) support software development, it is essential to remember that these models must be trained based on a varied dataset to meet the interaction characteristics of different users. The lack of information about the training data of proprietary tools (like GitHub Copilot) and data that considers social and human diversities can encourage a generation of social exclusion mediated by this type of technology [Bender et al. 2021]. In the case of LLM4code tools, this problem was reported by Liu and colleagues [Liu et al. 2023], who identified social bias in these LLMs. Thus, evaluation studies of LLMs must be conducted to verify whether they generate accessible codes and whether their bases reflect exclusionary or inclusive results.

3. Research Methods

We conducted an exploratory study to assess how intelligent programming assistants, specifically GitHub Copilot, generate a web application compliant with WCAG accessibility standards. We adopted a qualitative approach to evaluate our experience with GitHub Copilot by analyzing developers' diaries that were manually filled during programming activities. This was combined with a quantitative analysis aimed to summarize the reports generated by Pa11y, an automatic accessibility inspection tool.

3.1. The Web Application Developed

The web application developed during this study aims to support software developers in adopting inclusive design practices for emergent users of instant payment systems. We selected this application due to the prior need to create a platform that supports designers and developers, including PwD professionals, to design and understand the conditions of emerging users. It was a relevant and practical scenario for evaluating GitHub Copilot's support in writing accessible code aligned with WCAG guidelines. All applications, like this one, need to have accessible features available to include multiple users.

The web application was designed to include five distinct pages. The first page, called **“Home”**, briefly introduces the application's purpose and content with some text and one illustrative image. The second page, called **“Guidance”**, contains the reference guide on implementing accessibility features to emergent users. The guide is organized into textual cards with recommendations that can be filtered by category or searched using a text input field. Meanwhile, the third page, called **“Personas”**, presents cards with text and images that consolidate characteristics of hypothetical emergent users. The fourth page, named **“Quiz”**, was designed as a configurable set of questions and answers about the content presented in the previous pages. The user should be able to choose the number of questions and the subjects that the quiz should cover. Finally, the fifth and last page, **“About”**, presents the web application's complementary documents and the research team involved in its creation. Figure 1 presents a screen example of the web application. All prototype screens are available in our Supplemental Material (in Portuguese)².

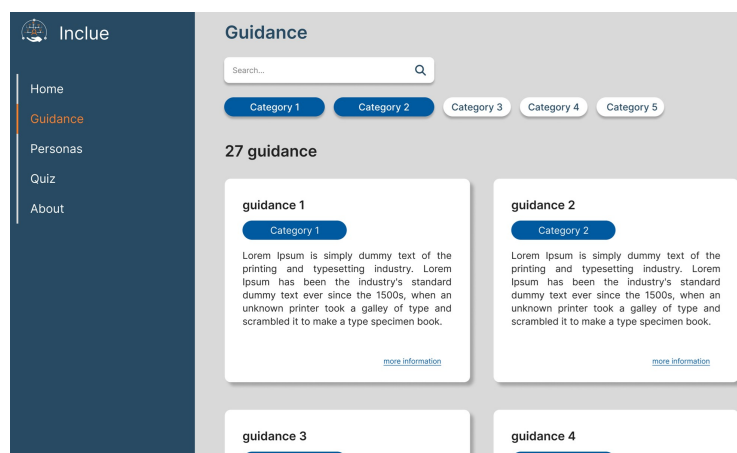


Figure 1. Screen example of the web application developed

3.2. Research Phases

The first and fourth authors conducted our empirical study in two phases (Figure 2), periodically collaborating and occasionally working independently. In the rest of this paper, they will be referenced as *researcher 1* and *researcher 2*.

In the **preliminary phase**, researchers 1 and 2 set up the development environment and defined the diary template, specifying what data would be recorded and when.

²<https://github.com/human-interaction-with-tecnologies/open-science-study-with-copilot-for-accessibility>

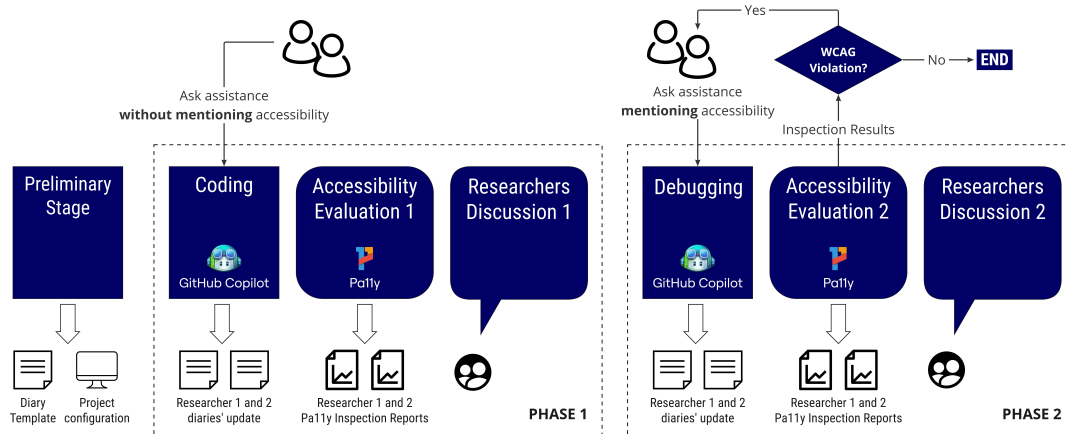


Figure 2. Phases of the empirical study

They structured the project and configured the repository to enable collaborative work. The technical stack, such as **Next.js**, a Framework based on React to develop web applications [Next.js 2024], TypeScript, and Tailwind were selected based on their prior experience. They also agreed to limit the number of prompt refinements to a maximum of four to interact with Copilot, aiming to avoid bias introduced by varying prompting strategies between different prompting approaches. Additionally, Researchers 1 and 2 decided to create a three-section journal (or diary) [Palen e Salzman 2002, Henderson-Summet et al. 2007] to be used while developing the web application.

This organization facilitated a systematic and structured documentation process, enabling the analysis and comparison of results throughout the study. A template was created for each entry in the journal. The structure of the template was as follows: The **First Section** recorded interactions with GitHub Copilot during the web application *codification* (phase one). This section included a subsection for the developed page, which contained: (i) Date, (ii) Prompt used in GitHub Copilot Chat, (iii) Scenario to be developed (behavior and internal structure of page), (iv) GitHub Copilot results in screenshots, (v) Observations about the accessibility support provided by Copilot. This section ended with a personal analysis summarizing the researcher's experience with Copilot's support; The **Second Section** followed the same structure as the first section but dedicated to recording the *debugging* activities (phase two); **Third Section** presented a subjective analysis of the researchers' perceptions of the codification and debugging phases compared, highlighting the differences among the researchers' experiences and main results.

In **phase 1 (coding)**, researchers collaborated in parallel to implement the same features in different branches using Copilot as the intelligent coding assistant. In preparation for subsequent analysis and discussion, Researchers 1 and 2 completed the initial section of their respective diaries during the coding phase. As an illustration, researcher 1 initially used the prompt “*Create 2 divs, 1 for the title and the second for the content. The content must contain 4 paragraphs, being paragraphs 2 and 3 followed by a list.*”. There were no explicit instructions regarding the generation of accessible interfaces in the developers' prompts. The absence of specific accessibility requirements was due to our desire to observe the ‘default’ results of Copilot.

After finishing the coding at phase 1, the researchers individually evaluated the accessibility of their web applications using Pa1ly [Pa1ly 2023], an automated accessibility inspection tool. This tool analyzed the source code and searched for WCAG accessibility violations, considering the AA compliance level. The results were saved in the development branch repositories of each researcher, and they concluded their diary’s first section with a personal evaluation of their experience. Table 1 presents an example of a violation result reported by Pa1ly. After phase 1, researchers 1 and 2 convened to deliberate over the entries recorded in their diaries and the accessibility inspection reports provided by Pally.

Table 1. Violation Example Reported in the Pa1ly

Violation’s Evidence
“error”, “WCAG2AA.Principle1.Guideline1_4_1_4_3.G18.Fail”, “This element has insufficient contrast at this conformance level. Expected a contrast ratio of at least 4.5:1, but text in this element has a contrast ratio of 3.71:1. Recommendation: change background to #2b71e6.”, “<div> Technology</div>”, “#__next > div > div:nth-child(2) > div:nth-child(2) > div:nth-child(3) > div:nth-child(1) > div”

During **phase 2 (debugging)**, the researchers looked for errors in the software and, consequently, found ways to resolve them [McCauley et al. 2008]. Thus, each researcher debugged its own software application independently, using the accessibility suggestions provided by Copilot, obtained through prompts enriched with explicit instructions on generating an accessible application, considering the learnings from the previous phase. Examples of the Researcher’s prompts include “*Refactor interface to support WCAG2AA norm Principle1 Guideline1_4_1_4_3 G18_Fail*” and “*Resolve accessibility problem for WCAG2AA 1.4.3 G18 contrast ratio*”.

Following the same method that was used in phase 1, both researchers wrote individual entries in their diaries, including their prompts and results from GitHub Copilot. Subsequently, the researchers conducted an individual review of the debugged code using Pa1ly, the same inspection tool that was employed in phase 1. The results were documented in the diaries and utilized to establish new input to continue debugging the code until all accessibility violations had been resolved. After finalizing debugging the code, the researchers consolidated their experiences using GitHub Copilot. To facilitate the reproducibility of the research (open science), the inspection reports³ and research diaries are available⁴ are available to consult.

At the end of the process, Pa1ly reported no accessibility violations in the web application. The researchers either manually resolved accessibility violations or accepted GitHub Copilot’s suggestions during the code debugging. The source code generated through all these research phases is also available in our open science repository.

3.3. Tools versions and configuration

We used **GitHub Copilot X**, with GPT-4 integrated and same features still used nowadays [Dohmke 2023], via an integrated development environment (IDE) extension to support our software development (coding and debugging) efforts. The Copilot Chat

³<https://bit.ly/454E2CB>

⁴<https://bit.ly/4kVxHz2>

feature was used to generate and enhance the web application code through prompts specially designed for each task. Copilot was set with the temperature 0 to refine its results and reduce the influence of its non-deterministic [Ouyang et al. 2023] characteristic. This configuration makes the Copilot's answers more specific and reliable for later analysis.

Following Barke et al's [Barke et al. 2023] GitHub Copilot's interaction modes, the researchers used both exploratory and accelerated modes to develop the web application. In phase 1, the assistant was asked to generate code suggestions via exploratory mode, and then accelerated mode was used to finalize the development. Meanwhile, in phase 2 the code was debugged using the exploration mode.

The programming language used was **Next.js** (version 14.0.3) based on Typescript (version 5). Syntactically Awesome Style Sheets (SCSS on the 1.69.7 version) and Tailwind library were added in the project settings and used in the web application front-end (version 3.3.0). The inspection tool selected for this research was **Pally** (version 3.0.3), which is frequently updated and extensively utilized by the software development community, with an estimated "156.174" downloads per week [npm, Inc. 2024].

Data analysis was conducted in two rounds, after phase 1 and phase 2 execution. The qualitative data of the diaries and the Pally report were analyzed by researchers 1 and 2 in the initial round. The analysis was initiated individually and subsequently conducted collaboratively. They categorized the results based on the WCAG violations of Pally reports, the prompts used by the researchers, and the GitHub Copilot code suggestions recorded in their diaries.

Results were presented to all other authors in the second round for further analysis. All authors verified and made adjustments in the initial classification and grouped the results into three major categories, namely: (i) accessibility features *automatically* suggested by Copilot (Subsection 4.1), (ii) accessibility features suggested by Copilot *during the debugging* (see Subsection 4.2), and (iii) accessibility features *never* addressed by Copilot (discussed in Subsection 4.3). The results (Section 4) were grouped this way to facilitate the understanding of the support provided by Copilot, avoiding redundancies, as in some cases, accessibility was not renewed in the commitment or debugging stages.

It is important to mention that some inspection results were not categorized because they were influenced by external factors unrelated to GitHub Copilot. For instance, the *Resize Text* feature (WCAG success criterion 1.4.4) was implemented by default through Next.js settings rather than due to GitHub Copilot's code suggestions. Therefore, this case is not reported later because it does not contribute to this study.

4. Results

This section presents the results of the empirical study conducted by researchers 1 and 2. During data analysis, we organized the information into three categories based on Copilot's ability to address WCAG requirements. It is worth noting that most of the generated prompts were originally written in Portuguese. In this section, we have translated them into English; however, when presented in figures, they may be in Portuguese.

4.1. Accessibility features *automatically* suggested by GitHub Copilot

This subsection presents the WCAG-compliant code generated with Copilot assistance during phase 1 of this research, specifically during the coding activities where researchers did not explicitly request accessibility support. Upon analyzing the researchers' diaries, it was observed that Copilot occasionally incorporated accessibility features under WCAG without explicitly asking for it. For instance, GitHub Copilot automatically suggested the alternative text property for image resources presented in the menus of pages “Guidance”, “Personas”, and “About” of the web application.

Figure 3 shows a code snippet implemented by researcher 1 in which the HTML tag “img” with an alternative text property was generated using the prompt “*create component for image*”. Likewise, Figure 4 displays the code snippet built by researcher 2 with the alternative text property for the “Image” element of the React framework. In this case, the prompt provided by researcher 2 to GitHub Copilot was: “*Implement the same card pattern as a tailwind in the div*”. It is important to note that the researchers did not request an alternative text in their prompts at any point. Instead, Copilot autonomously suggested the implementation of this property for image accessibility in accordance with WCAG Principle 1, Guideline 1.1, and Success Criterion 1.1.1 Non-text Content.

```
<p className={mu-4} {consideration.content}>/p>
<div className="mb-4">
  <ul>
    {consideration.images.map((image, index) => (
      <li key={index}>
        //criar um componente para imagem
        <img src={image.url} alt={image.alt} />
      </li>
    ))}
  </ul>
</div>
</div>
```

Figure 3. Researcher 1 Code snippet

```
<ul className="flex space-x-4">
  {consideration?.images.map((image) => {
    return (
      <li>
        <Image src={image.url}
          width={250}
          height={250}
          alt={image.text}
        />
      </li>
    );
  })}
</ul>
```

Figure 4. Researcher 2 Code snippet

Copilot automatically proposed the alternative text implementation; however, we noted that it did not occur with more advanced accessibility features. For instance, researcher 1 highlighted the following observation (subsection 1.12 of researcher 1's diary [Teran et al. 2025a]) in the conclusion of phase 1: “*I observed that GitHub Copilot generated alternative text whenever a code suggestion incorporated an image into its*

structure. However, GitHub Copilot did not generate other required resources, such as the name of Accessible Rich Internet inputs or attributes Applications (Aria)”.

ARIA is a set of properties that strengthens the web accessibility to people with disabilities [Mozilla 2024]. Specifically, in this case, the divs or inputs of the web application did not instance ARIA properties to facilitate communication with assistive technologies.

Besides alternative text, minimum contrast (success criteria 1.4.3) was another WCAG criterion automatically suggested by Copilot. It indicates that texts and backgrounds must have a contrast ratio of at least 4.5:1, except when the images are large scale (with a contrast ratio of at least 3:1), are not visible to anyone on the interface, or are logos and brand names [World Wide Web Consortium 2023c]. In this case, Copilot’s suggestions aligned with best practice interface design for presenting content to people with low vision, utilizing the Tailwind library as a base. We observed that the GitHub Copilot generated tailwind properties like “bg-white” and “text-gray-500”, “text-gray-600” or “text-gray-700” for researchers 1 and 2 (records can be found in subsections 1.5 and 1.7 of researcher 1’s diary [Teran et al. 2025a] and subsection 1.5 of researcher 2’s diary [Teran et al. 2025b]).

However, contrast was not successfully implemented for every code generated by Github Copilot, as presented in Subsection 4.3. In summary, Copilot automatically suggested simple accessibility features, such as alternative text for images and contrast, but it did not support more sophisticated accessibility features. Those features are discussed in the following sections.

4.2. Accessibility features suggested by GitHub Copilot during the debugging

This subsection presents the accessibility standards Copilot delivered through the debugging activity (phase 2), when researchers explicitly prompted the generation of WCAG-compliant code. Debugging was applied to the code generated at the end of this research’s phase 1, since Copilot delivered a web application not fully compliant with WCAG guidelines.

Table 2 lists the inspection results at the end of phase 1, identified by its WCAG success criteria code and grouped by each page where it was found. A full description of each violation is provided next.

The accessibility violations found at this phase were: **Page Title (2.4.2)** indicates that pages must have titles that describe their purposes [World Wide Web Consortium 2023d]; **Name, Role, Value (4.1.2)** describes that interface elements must have names, functions, or states defined during programming to facilitate reading and notification to be carried out by assistive technologies to users [World Wide Web Consortium 2023e]; **Info and Relationships (1.3.1)** indicates that the information or relationships of the pages will be defined via programming to maintain their integrity when there is a change in the content presented to users [World Wide Web Consortium 2023b]; **Bypass Blocks (2.4.1)** is about mechanism is available to bypass blocks of content that are repeated on multiple pages [World Wide Web Consortium 2023f]; **Minimum Contrast (1.4.3)** highlights that the visual presentation of text and images of text need to be adequate contrast ratio [World Wide Web Consortium 2023c].

Table 2. Inspection results after phase 1

Page	Researcher 1	Researcher 2
Home	Page Title (2.4.2); Minimum Contrast (1.4.3).	Page Title (2.4.2); Minimum Contrast (1.4.3).
Guidance	Page Title (2.4.2); Name, Role, Value (4.1.2); Info and Relationships (1.3.1); Minimum Contrast (1.4.3).	Page Title (2.4.2); Name, Role, Value (4.1.2); and Info and Relationships (1.3.1); Minimum Contrast (1.4.3).
Personas	Page Title (2.4.2); Minimum Contrast (1.4.3).	Page Title (2.4.2); Minimum Contrast (1.4.3).
Quiz	Page Title (2.4.2); Minimum Contrast (1.4.3).	Page Title (2.4.2); Info and Relationships (1.3.1); and Name, Role, Value (4.1.2); Minimum Contrast (1.4.3).
About	Page Title (2.4.2); and Bypass Blocks (2.4.1); Minimum Contrast (1.4.3).	Page Title (2.4.2); and Bypass Blocks (2.4.1); Minimum Contrast (1.4.3).

To resolve the violations, each researcher refactored their code using GitHub Copilot. It was necessary to increasingly add more information to the prompts to get the expected result. Below we present the reported cases:

Case #1: Researcher 1 used the following prompts (subsection 2.1 of researcher 1’s diary [Teran et al. 2025a]) to fulfill success criteria 2.4.2 while attempting, without success, to get a Copilot suggestion that addressed the violation: “*Refactor the code to comply with the WCAG accessibility standard*”; “*Refactor the code to comply with the WCAG accessibility standard: A title should be provided for the document, using a non-empty title element in the head section.*”; “*Refactor the code to meet the WCAG accessibility standard to meet the criteria of success 2.4.2 in react*”. However, these prompts did not generate the expected result. To address the violation, researcher 1 prompted: “*Refactor the code to correct the accessibility problem ‘A title should be provided for the document, using a non-empty title element in the head section’*”.

Meanwhile, to correct the title issue (subsection 2.1 of researcher 2’s diary [Teran et al. 2025b]), researcher 2 used the command below, interpreting the description provided by the inspection report: “*A title should be provided for the document, using a non-empty title element in the head section.*”. With this prompt, he updated successfully in the first attempt: “*Refactor to use head title instead of h1*”.

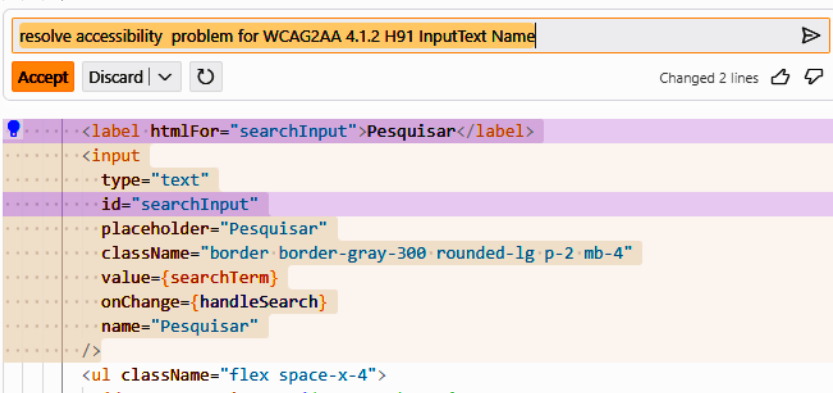
Case #2: To attend to success criteria 4.1.2, researcher 1 created a debugging prompt based on the previous prompt’s structure, i.e., using the description provided by the inspection report since it was the solution that worked before. Thus, the following prompt (subsection 2.3 of researcher 1’s diary [Teran et al. 2025a]) was successfully used to solve this violation: “*Refactor the code to fix the accessibility problem ‘This textinput element does not have a name available to an accessibility API. Valid names are: label element, title undefined, aria-label undefined, aria-labelled by undefined’*”.

With this approach, GitHub Copilot provided the suggestion to insert the “aria-label” and “name” properties in the “input” tag, as shown in Figure 5, to facilitate the identification of the data entry field by assistive technologies. The Pa11y description report was a critical component that guided the researcher while prompting, as it stipulated the actions that the assistant should take to a specific accessibility error in the code.

```
<div className="flex gap-3 mt-7 mb-5">
  <div className="relative">
    <input
      type="text"
      placeholder="Buscar"
      className="border border-gray-300 rounded-md px-2 py-1 pr-8"
      value={searchText}
      onChange={handleSearchTextChange}
      name="searchText" // Add name attribute for accessibility
      aria-label="Buscar" // Add aria-label attribute for accessibility
    />
    <div className="absolute right-2 top-1/2 transform -translate-y-1/2">
      <svg width="18" height="19" viewBox="0 0 18 19" fill="none" xmlns="http://www.w3.org/2000/svg">
        <path d="M14.625 8.0625C14.6016 9.70312 14.1328 11.1445 13.2188 12.3" />
      </svg>
    </div>
  </div>
</div>
```

Figure 5. Code Suggested for researcher 1 in Data Input Violation

In turn, researcher 2 used the violation identification code provided by the inspection report to the *Name*, *Role*, *Value* success criteria 4.1.2, as shown in Figure 6, so he entered the prompt (reported in subsection 2.3 of researcher 2’s diary [Teran et al. 2025b]) that solved the accessibility violation: “*Resolve accessibility problem for WCAG2AA 4.1.2 H91 InputText Name*”.



The screenshot shows a code editor interface. At the top, a search bar contains the prompt: "resolve accessibility problem for WCAG2AA 4.1.2 H91 InputText Name". Below the search bar are buttons for "Accept", "Discard", and a refresh icon. To the right of the buttons, it says "Changed 2 lines". Below the search bar, a code snippet is displayed, showing the suggested fix for the accessibility issue. The code is as follows:

```
<label htmlFor="searchInput">Pesquisar</label>
<input
  type="text"
  id="searchInput"
  placeholder="Pesquisar"
  className="border border-gray-300 rounded-lg p-2 mb-4"
  value={searchTerm}
  onChange={handleSearch}
  name="Pesquisar"
/>
<ul className="flex space-x-4">
```

Figure 6. Code Suggested for Researcher 2 in Input Violation

However, interestingly, the same prompt had to be run twice by researcher 2, and only on the second attempt did the GitHub Copilot suggestion successfully fix the accessibility issue. As discussed in Subsection 2.2, this is an example of the non-deterministic nature of LLMs. Indeed, researcher 2 highlighted in his diary that “[...] *non-determinism was a factor that got in the way when generating this debugging because I had to try to execute the command at the prompt more times than expected to fix a violation [...]*”.

As mentioned in Subsection 6.3, the researchers had limited experience with the inner works of LLMs. However, one of them was already familiar with the LLM’s limitation regarding non-determinism.

Comparing the researcher’s interaction with GitHub Copilot, researcher 1 received a suggestion to insert a property in the input, and researcher 2 received a suggestion to insert the “label” tag attached to the Input identifier. Furthermore, the prompts used by

researcher 1 or 2 to resolve “*Name, Role, Value*” violation, *simultaneously*, also fixed the success criteria 1.3.1: “*Info and Relationships*”. This violation of criterion 1.3.1 was due to the lack of information identifying particular content. In simpler terms, label could be read by screen readers when it was inserted into the data input.

Case #3: Finally, the “Bypass Blocks” violation (related to success criteria 2.4.1) was solved by researcher 1 using the prompt strategy of including the message presented in the inspection report as documented in subsection 2.4 of his diary [Teran et al. 2025a, Teran et al. 2025b]. The following prompt solved the issue in the first attempt: “*Refactor the code to fix the accessibility issue ‘Iframe element requires a non-empty title attribute that identifies the frame.’*”.

Researcher 2 used a similar prompt, describing the request and including the message indicated by the inspection report, having also successfully resolved the accessibility violation (subsection 2.6 of researcher 2’s diary [Teran et al. 2025b]). The prompt used was: “*Resolve Iframe element requires a non-empty title attribute that identifies the frame*”.

4.3. Accessibility features never fully addressed by GitHub Copilot

Case #4: Our investigation revealed that Copilot did not support addressing certain accessibility violations. Researchers 1 and 2 attempted to enhance code snippets to provide sufficient contrast (criterion success 1.4.3 by WCAG). However, they were unable to resolve this issue even after testing different prompts (subsection 2.5 of researcher 1’s diary [Teran et al. 2025a] and subsection 2.8 of researcher 2’s diary [Teran et al. 2025b]).

It is important to mention that the informative text located in the “**About**” page had the contrast success criteria automatically addressed during phase 1, coding (see Subsection 4.1). However, the suggestions did not work consistently in the menu of the pages.

Table 3 presents all requested Prompts⁵ used specifically by researcher 1 in attempting to solve the contrast violation (subsection 2.5 of researcher 1’s diary [Teran et al. 2025a]). In P1, the researcher constructed a prompt that combined the task request and the message from the inspection report, using the same strategy that had worked in previous issues. However, it did not work this time. In P2 and P3, he adapted the prompt approach, considering both the inspection report message and the problem’s recommendation.

After testing the commands presented in Table 3 without success, researcher 1 performed the adjustment manually. In his diary, he highlighted the following observation: “[...] *the contrast error was not resolved due to the problem being caused by a property that left the menu text gray. Therefore, tests were carried out with commands that had color recommendations or problem locations in their structure, but the Copilot was unable to interpret and apply an appropriate adjustment. Thus, I made the manual adjustment by removing the class that caused this contrast problem.*”.

Researcher 2 indicated that GitHub Copilot did not provide necessary adjustments to contrast issues (see subsection 2.8 of researcher 2’s diary [Teran et al. 2025b]). In this

⁵Prompts are requests or questions made by users to GitHub Copilot Chat to generate code, fix bugs, write tests, and document code [GitHub 2025].

Table 3. Prompts Generated by Researcher 1 to Fix Contrast Problem

Code	Prompts
P1	<i>Refactor the code to fix the accessibility issue “This element has insufficient contrast at this compliance level. A contrast ratio of at least 4.5:1 was expected, but the text in this element has a contrast ratio of 4.38:1. Recommendation: Change the background to #f7f8fa.”</i>
P2	<i>Refactor the code to fix accessibility issue “This element has insufficient contrast at this conformance level. Expected a contrast ratio of at least 4.5:1, but text in this element has a contrast ratio of 4.38:1.” located in “Questions and Answers” following the recommendation “change background to #f7f8fa !important.”</i>
P3	<i>Refactor the code to fix the accessibility issue “This element has insufficient contrast at this conformance level. Expected a contrast ratio of at least 4.5:1, but text in this element has a contrast ratio of 4.38:1.” located in “Questions and Answers” following the “change recommendation background to #f7f8fa and remove other property if necessary.”</i>

context, this researcher used the following prompt, which had no effect in correcting the contrast: “To solve the problem for insufficient contrast, change background to #2b71e6 in tailwind”.

Additionally, manual adjustments were made because Copilot changed other elements that were in compliance before testing this prompt. In his diary observations, this researcher wrote: “When executing command one, it did not solve the problem and even ‘broke’ the color, I needed to act manually to solve the problem of buttons with blue color outside the accessibility standards’ contrast pattern.”.

In summary, Copilot’s debugging suggestions received by researcher 1 and researcher 2 did *not* produce a suitable suggestion for addressing the contrast violation. Consequently, the researchers manually made this adjustment after testing various prompts. These results suggest that the problem is related to design standards and accessibility information, which is likely outside the scope of Copilot’s training data. We will discuss this further in the next Section.

4.4. Summary of the Results

Table 4 summarizes quantitatively the following results:

- Some success criteria were implemented during phase 1 (coding), in which researcher 1 (R1) and researcher 2 (R2) noted that GitHub Copilot suggested code with the minimum level of contrast (success criterion 1.4.3) and alternative text (totalling four implementations in favour of accessible code);
- Five success criteria were not implemented (success criterion 2.4.2, 4.1.2, 1.3.1, 2.4.1, and 1.4.3), resulting in a total of 72 accessibility violations (15 present in the code developed by researcher 1 and 57 present in the code developed by researcher 2);
- Researchers 1 and 2 resolved accessibility violations with the assistance of GitHub Copilot and the Pa11y tool (42 in total, nine after debugging the code by researcher 1 and 33 after code debugging by researcher 2);
- Many contrast minimum level (success criterion 1.4.3) issues were not resolved with the help of GitHub Copilot (32 in total, 6 in the debugged code by researcher 1, and 24 in the debugged code by researcher 2);

- Specifically about the contrast minimum level, it was implemented automatically when GitHub Copilot a code with Tailwind property (see more details in Subsection 4.1). In phase 2, the same success criteria were impossible to implement with GitHub Copilot or Pa1ly support, resulting in remaining violations, even after debugging, by design matters (see more details in Subsection 4.3).

Table 4. Quantitative summary

Success Criteria	Coding (Phase 1)				Debugging (Phase 2)			
	Implemented		Violations		Fixed		Violations	
	R1	R2	R1	R2	R1	R2	R1	R2
2.4.2 Page Titled	-	-	5	5	5	5	-	-
4.1.2 Name, Role, Value	-	-	2	15	2	15	-	-
1.3.1 Info and Relationships	-	-	1	1	1	1	-	-
2.4.1 Bypass Blocks	-	-	1	12	1	12	-	-
1.4.3 Contrast (Minimum level)	1	1	6	24	-	-	6	24
1.1.1 Non-text Content (Alternative Text)	1	1	-	-	-	-	-	-
Total	2	2	15	57	9	33	6	24

5. Discussion

Our experience in developing and debugging web applications to support WCAG accessibility guidelines enables us to answer our research question “*How can Copilot assist in creating accessible websites?*”. We concluded that Copilot suggests code with accessibility features that are primarily simple to implement, and in this sense, are likely to be found in its training data. Examples include the image alternative text and the minimum contrast level specified in the Tailwind library. However, Copilot also suggests code that is *not* compliant with accessibility, requiring, in the best case, an automated code inspection tool or, in the worst case, manual code changes by a developer. These results are corroborated by the researchers’ diaries and the Pa1ly inspection tool, which revealed several accessibility violations in the GitHub Copilot-suggested code.

It is important to note that the contrast violation occurred while both researchers 1 and 2 were coding, as observed in phase 1. Furthermore, both researchers attempted to improve the code with different prompts, but neither was able to resolve the issue with Copilot’s assistance. This situation suggests that more specific problems, such as those related to design skills, are not easily and quickly solved with Copilot’s support. It also highlights the importance of software developers’ awareness of accessibility guidelines and patterns [Freire et al. 2007, Freire et al. 2008, Antonelli et al. 2018, Leite et al. 2021]. Since Phase 1, it has been considered a simulation of real software development behavior, aided by Copilot, which has become increasingly integrated into developers’ practices. When Copilot was used in practice, it became apparent that it does not effectively help developers gain experience in the sociotechnical aspects of building inclusive software, particularly for beginners or those unfamiliar with best practices for software accessibility.

Copilot should be used as a tool that supports the programmer; therefore, it is important to adopt a human perspective on the assistant's suggested code, as humans possess the ultimate knowledge and critical ability to evaluate the suggestions [Ernst e Bavota 2022]. In the context of accessibility, we contend that human-AI collaboration is more crucial than the replacement of humans by AI-based tools. Through our experiences, it was observed that developers need to refine their prompts to ensure they provide source code that adheres to accessibility best practices. Therefore, it is important to evaluate the potential role that prompt literacy and AI-assisted development training may play in achieving better results when creating accessible applications with Copilot-like tools.

Human-AI interaction is a critical area of study for software engineering and HCI researchers, as LLMs are becoming a standard tool in software development. These models can support and adapt to various scenarios across the software development process and team functions, such as back-end development and testing, by enabling effective interaction, code generation, assistance, and optimization [Hamza et al. 2024]. Given that programming knowledge, accessibility design, and prompt engineering are inherently complex human activities, it is unsurprising that researchers adopted different ways of interacting with GitHub Copilot to create an accessible web application. For instance, researcher 1 used multiple prompts to address an accessibility violation (subsection 2.1 of researcher 1's diary [Teran et al. 2025a]), where he had to "negotiate" with the intelligent assistant by increasingly adding more information to the prompts. In contrast, researcher 2 sometimes achieved similar results from GitHub Copilot with a single prompt (subsection of researcher 2's diary [Teran et al. 2025b]).

Non-determinism affected GitHub Copilot's suggestions for code accessibility, despite it being set to temperature 0. This concept was highlighted by researcher 2, who needed to run the same prompt multiple times to improve the code, ensuring it addressed the accessibility violations of the web application. This behavior reinforces previous argument that while an LLM can suggest a set of outputs from a specific prompt, developers must have the proper knowledge to decide which, if any, one to accept [Ouyang et al. 2023].

The combination of Pa11y and Copilot generated interesting accessibility solutions. Both tools supported researchers in resolving accessibility issues in the source code they developed. Researcher 1 used the violation messages provided by Pa11y to improve his code. Meanwhile, researcher 2 interpreted the violation messages presented by Pa11y to create the prompts for Copilot or to describe the debugging task to fix accessibility issues. Both researchers used data from violation messages from the Pa11y inspection report as input to address the accessibility violations. As we will discuss in our lessons learned (Section 5.1), combining an automatic accessibility inspection tool with Copilot might be useful for developers interested in creating inclusive web applications.

It is important to emphasize that not all accessibility principles are testable by inspection tools or can be fully addressed with the sole support of an intelligent assistant. As we presented, during the empirical study, Copilot did not always provide suggestions that adhered to accessible design, as in the case of solving contrast problems for the menu, present in all pages of the application (Table 4 - success criteria 1.4.3) presented in Subsection 4.3. As demonstrated in Subsection 4.1, the contrast of the **"About"** menu

adhered to the project's design pattern and best practices for interface design. Certain accessibility features *cannot* be evaluated using automated accessibility inspection tools. These include live subtitles (success criterion 1.2.4), pre-recorded audio description (success criterion 1.2.5), and sign language (success criterion 1.2.6). Therefore, further research is needed to assess and enhance LLM4 code tools, such as Copilot, to improve their ability to support the development of accessible software applications.

The lack of support in code generation adherent to accessibility guidelines corroborates the seminal study conducted by Bender et al. [Bender et al. 2021], which indicates that the LLMs used by intelligent coding assistants are trained in hegemonic data. Thus, lack of representation of social diversity in LLMs can lead to exclusions within the population. As a result, the lack of diversity in the data used to train LLMs reflects social biases in the code suggested by these tools [Liu et al. 2023] that affect the generation of accessible software applications for various people, such as people with disabilities. Future research should address these aspects to promote a more just and egalitarian society.

5.1. Using GitHub Copilot for Accessibility

Based on our results, we present four lessons learned for using GitHub Copilot to assist the development of accessible software applications:

- **Lesson 1:** Automatic accessibility inspection tools, such as Pa1ly, combined with intelligent assistants, such as GitHub Copilot, can help evaluate and correct accessibility problems. *Evidence 1:* In the second phase of our empirical study, GitHub Copilot received information from the Pa1ly tool and assisted the developers in evaluating and correcting the accessibility issues of the web application;
- **Lesson 2:** The elimination of accessibility violations is achieved through the implementation of a prompt that specifies the necessary modifications in conjunction with the problem description. A suggested input command is: *Refactor the code to fix the accessibility problem "<accessibility error description>"*. *Evidence 2:* Researchers 1 and 2 used the description of the accessibility error from the Pa1ly report. To do this, researcher 1 developed a small standardized request for accessible code, stating what he wanted to do and what the problem was that GitHub Copilot should fix;
- **Lesson 3:** If applying lesson 2 is not enough, the developer should either select the location of the problem using a mouse or keyboard, or indicate the location of the accessibility violation in the prompt. *Evidence 3:* researcher 2 indicated the location of the accessibility problem so that Copilot could solve it: *"resolve Iframe element requires a non-empty title attribute that identifies the frame"*;
- **Lesson 4:** Understanding how to implement and evaluate accessibility in web applications was essential for attempting GitHub Copilot's support and defining strategies to address accessibility barriers that hindered the WAG conformance. *Evidence 4:* When observing researchers 1 and 2's diaries, we noticed that the minimum contrast level was not implemented even after the debugging with Copilot support. In this sense, the researchers' knowledge of accessible design and software development was necessary to make the necessary adjustments.

Our results highlight the opportunities and challenges in implementing and evaluating the accessibility of an application built using GitHub Copilot and automatic accessibility inspection tools. We hope our results generate technical and conceptual reflections on using GitHub Copilot and its influence on software quality, developer productivity, and the digital inclusion of diverse people. Finally, it is worth noting that this study was directed at GitHub Copilot. Changes to the model or the use of other intelligent assistants may impact the results, discussion, and lessons learned.

6. Final Remarks

This article presents an empirical study conducted using GitHub Copilot to assess its support in developing and evaluating accessible software applications. The study was divided into two main phases: (i) Coding, where the web application was developed with GitHub Copilot and automatically inspected using the accessibility inspection tool Pa11y; and (ii) Debugging, in which the accessibility errors were fixed with the support of GitHub Copilot and reports from Pa11y.

Our findings indicate that GitHub Copilot did not widely suggest code that adhered to WCAG standards during the development of the web application. Although it offered alternative text properties for images, it neglected to address other accessibility violations that were more critical. During debugging, we observed that the results from Pa11y, when combined with Copilot, helped programmers address most, but not all, accessibility violations.

Our involvement encompasses both technical and social aspects. In terms of technical aspects, the results underscore the need for advances in Copilot and Pa11y to assist programmers in producing accessible software that is both higher-quality and accessible, while also reducing cognitive load. Our lesson learned for using Copilot, presented in the Section 5.1, corroborates this assertion. In addition, as a social contribution, we highlight the importance of code suggestions that prioritize accessibility to promote digital inclusion for diverse individuals. In other words, Copilot has the potential to make web applications more accessible, as it is used by more than 1 million developers [Cui et al. 2024]. However, it is necessary to recognize that not all code suggestions from this intelligent assistant can incorporate the accessible features. In an effort to satisfy this interaction quality criterion, developers must be aware of digital accessibility and the ability to analyze their codes [Alshayban et al. 2020].

Finally, we emphasize that our research is consistent with the actions proposed by the grand challenges of HCI research agenda in Brazil [Duarte et al. 2024]. Specifically, concerning immediate action, our experience of use corroborates the understanding of AI assistance in the development of accessible software, which is in AI compliance with international standards for digital inclusion (WCAG), and the (non-)help provided to software development professionals. Consequently, our findings and the lessons we have acquired are intended to facilitate the discussion and exchange of reflections on the use of AI from the perspective of its transparency, visions, and challenges for social diversity.

6.1. Limitations

The purpose of this study was to understand the support provided by GitHub Copilot in developing accessible software. In this context, the web application was developed

to collect evidence on the accessibility of the interface built using Copilot's suggested code. However, considering that the web application developed has a medium level of complexity and only five sections of pages for interaction, it would be helpful to conduct additional case studies while developing more complex interactive systems to assess the influence of the intelligent assistant on the accessibility of the resources developed. These systems could even employ user interface pages and features that were not present in our web application. In summary, one of the work's limitations is the application's easy to medium complexity. Consequently, the accuracy of the results can be significantly enhanced by increasing the quantity of data obtained, including prompts, generated codes, and suggestions provided by GitHub Copilot. In addition, performing multiple case studies [Runeson e Höst 2009] with diverse scenarios and developers is necessary to further understand how GitHub Copilot supports accessibility.

An additional limitation of this investigation was the small number of developers who used GitHub Copilot, specifically two in our instance: researcher 1 and researcher 2. Increasing the number of developers in the research may improve the data gathered regarding the code's accessibility, as recommended by Copilot. Additionally, the developers are also authors of this paper, which may have contributed to biases. However, it is worth noting that one of the researchers is a professional software developer, while the other has years of industry experience as a developer. It was not possible to analyze the impact that the developer's experience with GitHub Copilot and prompt engineering would have on the results; therefore, this could be explored in further research. Furthermore, their empirical studies underscore the necessity of examining this quality criterion and, consequently, the definition of techniques that corroborate the construction of accessible software assisted by intelligent assistants.

Pally did not present accessibility compliance data, it only reports *lack* of compliance. We also did not inspect the code to identify additional situations that met the WCAG success criteria, in addition to the alternative text and contrast observed in the researcher's diaries during the coding phase (phase 1). Therefore, Copilot might provide more accessibility support than what we reported. The reader must consider that Pally reports identified numerous accessibility violations in the code generated with Copilot's recommendations (Table 4). In any case, we argue that we provide initial evidence about the accessibility support provided by GitHub Copilot, especially in the context of accessibility features that required debugging or were not supported at all.

Finally, our findings may not apply to other coding assistants beyond GitHub Copilot, due to the unique characteristics of each tool. These differences stem from variations in the underlying large language models (LLMs) used by these tools, as well as their training parameters, the volume and sources of training data, and the ease of access for users. Consequently, this limits the generalizability of our results.

6.2. Future Work

We plan to validate the web application with people with disabilities to determine whether the code debugging has promoted the accessibility we expected, documenting these results and any remaining violations. This approach also enables triangulation of results among automatic inspection tools, developers' experiences, and interaction by individuals who rely on accessibility features. Specifically, a new study will be conducted using

a different intelligent coding assistant to expand the breadth of the data obtained and, more importantly, to investigate whether the problem of code generation that adheres to accessibility is a recurring issue among other intelligent assistants. We plan to utilize a code assistant that enables developers to incorporate additional context, domain-specific knowledge, or ensembles of various models to enhance its LLM's ability to handle ambiguity and generate accurate code [Hou et al. 2024]. To achieve this objective, we plan to utilize Retrieval-Augmented Generation (RAG). This methodology integrates information from external databases to generate outcomes that satisfy user inquiries that are not accommodated by the LLM database [Gao et al. 2023]. Therefore, in a further study, we will develop a curated dataset containing code techniques supported by WCAG to observe whether this new knowledge successfully empowers the assistant to generate coding suggestions that allow developers to implement accessible resources in a preventive manner. Our ultimate goal is to utilize RAG to mitigate some inherent biases [Bender et al. 2021] in LLMs' assistance in building accessible applications.

6.3. Ethical Aspects and Researcher's Positionality

This research was based on the researchers' professional experience, and, according to CNS Resolution 510/2016, it does not require approval by an ethics committee. Next, we highlight our social, cultural, and academic background that motivated and drove our perspective in this research.

About the two researchers and co-authors who developed the web application (researcher 1 and researcher 2): Researcher 1 is an autistic person and a Ph.D. candidate in computer science. He has been working in software development since 2015 and has a 4 years of experience as a full-stack and front-end developer in the industry. His research focuses on Human-Computer Interaction (HCI), specifically on digital accessibility and he expects accessibility to be increasingly incorporated into software development projects. Researcher 2 holds a degree in Information Systems and works as the technical leader at a large software company. With over 13 years of experience in programming, he interacts daily with quality analysts, including some individuals with disabilities. He supports junior developers entering programming careers and believes that utilizing LLM-based tools, such as GitHub Copilot, can ease this process.

The remaining authors are also involved in research projects and professional experiences engaged with people with disabilities. We are motivated and positioned by social inclusion and the right to access [Bardzell e Bardzell 2011, Erete et al. 2018, Keyes et al. 2020, Liang et al. 2021], specifically to digital technologies. Based on our social experiences and technical skills, we hope to shorten the challenges in creating accessible software applications mediated by GitHub Copilot, and similar tools, promoting the inclusion of all potential users.

Acknowledgement

We would like to thank Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - grant numbers 420406/2023-9 and 442779/2023-2 - for partially funding this study, and to Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Finance Code 001 - for providing scholarship funding to the first, second, and fifth authors. We thank GitHub Copilot (Microsoft) for the empirical study and Grammarly for the review of our writing.

References

- Acosta-Vargas, P., Salvador-Acosta, B., Novillo-Villegas, S., Sarantis, D., e Salvador-Ullauri, L. (2024). Generative artificial intelligence and web accessibility: Towards an inclusive and sustainable future. *Emerging Science Journal*, 8(4):1602–1621.
- Alshayban, A., Ahmed, I., e Malek, S. (2020). Accessibility issues in android apps: state of affairs, sentiments, and ways forward. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20*, page 1323–1334, New York, NY, USA. Association for Computing Machinery.
- Antonelli, H. L., Rodrigues, S. S., Watanabe, W. M., e de Mattos Fortes, R. P. (2018). A survey on accessibility awareness of brazilian web developers. In *Proceedings of the 8th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-Exclusion, DSAI '18*, page 71–79, New York, NY, USA. Association for Computing Machinery.
- Bardzell, S. e Bardzell, J. (2011). Towards a feminist hci methodology: social science, feminism, and hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11*, page 675–684, New York, NY, USA. Association for Computing Machinery.
- Barke, S., James, M. B., e Polikarpova, N. (2023). Grounded copilot: How programmers interact with code-generating models. *Proc. ACM Program. Lang.*, 7(OOPSLA1).
- Bender, E. M., Gebru, T., McMillan-Major, A., e Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, FAccT '21*, page 610–623, New York, NY, USA. Association for Computing Machinery.
- Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., e Gazit, I. (2023). Taking flight with copilot: Early insights and opportunities of ai-powered pair-programming tools. *Queue*, 20(6):35–57.
- Borges, J. M. e Araújo, R. D. (2024). Experiences and challenges of a redesign process with the support of an ai assistant on an educational platform. In *Proceedings of the XXIII Brazilian Symposium on Human Factors in Computing Systems, IHC '24*, New York, NY, USA. Association for Computing Machinery.
- Brazil (1991). Law 8,213, dated july 24, 1991. http://www.planalto.gov.br/ccivil_03/leis/l8213cons.htm%7D%7D. Accessed on August 08, 2025.
- Brazil (2009). Decree No. 6,949 of August 25, 2009. https://www.planalto.gov.br/ccivil_03/_ato2007-2010/2009/decreto/d6949.htm. Accessed on May 06, 2024.
- Cui, Z., Demirer, M., Jaffe, S., Musolff, L., Peng, S., e Salz, T. (2024). The Effects of Generative AI on High Skilled Work: Evidence from Three Field Experiments with Software Developers. *SSRN eLibrary*.
- da Silva, I., Silva, M. P., Teran, L. A., e Mota, M. P. (2024). Desafios da inteligência artificial generativa na construção de sistemas computacionais acessíveis. In *Anais do XV Workshop sobre Aspectos da Interação Humano-Computador para a Web Social*, pages 19–28, Porto Alegre, RS, Brasil. SBC.

- de Souza, C. R. B., Rodríguez-Pérez, G., Basha, M., Yoon, D., e Beschastnikh, I. (2024). The fine balance between helping with your job and taking it: Ai code assistants come to the fore. *IEEE Software*, 41(6):111–118.
- Deque Systems (2024). Glossary | Deque Docs — docs.deque.com. <https://docs.deque.com/devtools-for-web/4/en/glossary#violation>. Accessed on April 23, 2024.
- Dohmke, T. (2023). Github copilot x: The ai-powered developer experience. <https://github.blog/news-insights/product-news/github-copilot-x-the-ai-powered-developer-experience/>. Accessed on May 29, 2025.
- Duarte, E. F., Toledo Palomino, P., Pontual Falcão, T., Porto, G. L. P. M. B., Portela, C. d. S., Ribeiro, D. F., Nascimento, A., Costa Aguiar, Y. P., Souza, M., Moutin Segoria Gasparotto, A., e Maciel Toda, A. (2024). Grandihc-br 2025-2035 - gc6: Implications of artificial intelligence in hci: A discussion on paradigms ethics and diversity equity and inclusion. In *Proceedings of the XXIII Brazilian Symposium on Human Factors in Computing Systems*, IHC '24, New York, NY, USA. Association for Computing Machinery.
- Erete, S., Israni, A., e Dillahunt, T. (2018). An intersectional approach to designing in the margins. *Interactions*, 25(3):66–69.
- Ernst, N. A. e Bavota, G. (2022). Ai-driven development is here: Should you worry? *IEEE Software*, 39(2):106–110.
- Freire, A. P., Cardoso, P. C. F., e Salgado, A. d. L. (2024). May we consult chatgpt in our human-computer interaction written exam? an experience report after a professor answered yes. In *Proceedings of the XXII Brazilian Symposium on Human Factors in Computing Systems*, IHC '23, New York, NY, USA. Association for Computing Machinery.
- Freire, A. P., Goularte, R., e de Mattos Fortes, R. P. (2007). Techniques for developing more accessible web applications: A survey towards a process classification. In *Proceedings of the 25th Annual ACM International Conference on Design of Communication*, SIGDOC '07, page 162–169, New York, NY, USA. ACM.
- Freire, A. P., Russo, C. M., e Fortes, R. P. (2008). A survey on the accessibility awareness of people involved in web development projects in brazil. In *Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A)*, pages 87–96.
- Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., e Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.
- Germany (2002). Law on equality for persons with disabilities. <https://www.gesetze-im-internet.de/bgg/index.html>. Accessed on February 03, 2025.
- GitHub (2025). Getting started with prompts for Copilot Chat - GitHub Docs. <https://docs.github.com/en/copilot/using-github-copilot/guides-on-using-github-copilot/getting-started-with-prompts-for-copilot-chat>. Accessed on Feb 02, 2025.

- Guspian, I., Lin, C.-C., Huang, A. Y., e Yang, S. J. (2024). Transforming outreach effects: The impact of generative ai on inclusive education branding and sustainability. *Journal of Education For Sustainable Innovation*, 2(2):170–179.
- Hamza, M., Siemon, D., Akbar, M. A., e Rahman, T. (2024). Human-ai collaboration in software engineering: Lessons learned from a hands-on workshop. In *Proceedings of the 7th ACM/IEEE International Workshop on Software-Intensive Business*, IWSiB '24, page 7–14, New York, NY, USA. Association for Computing Machinery.
- Henderson-Summet, V., Grinter, R. E., Carroll, J., e Starner, T. (2007). Electronic communication: Themes from a case study of the deaf community. In *Human-Computer Interaction–INTERACT 2007: 11th IFIP TC 13 International Conference, Rio de Janeiro, Brazil, September 10-14, 2007, Proceedings, Part I 11*, pages 347–360. Springer.
- Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., e Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*, 33(8).
- Kavcic, A. (2005). Software accessibility: Recommendations and guidelines. In *EUROCON 2005 - The International Conference on "Computer as a Tool"*, volume 2, pages 1024–1027.
- Keyes, O., Peil, B., Williams, R. M., e Spiel, K. (2020). Reimagining (women's) health: Hci, gender and essentialised embodiment. *ACM Trans. Comput.-Hum. Interact.*, 27(4).
- Kulkarni, M. (2019). Digital accessibility: Challenges and opportunities. *IIMB Management Review*, 31(1):91–98.
- Leite, M. V. R., Scatalon, L. P., Freire, A. P., e Eler, M. M. (2021). Accessibility in the mobile development industry in brazil: Awareness, knowledge, adoption, motivations and barriers. *Journal of Systems and Software*, 177:110942.
- Liang, C. A., Munson, S. A., e Kientz, J. A. (2021). Embracing four tensions in human-computer interaction research with marginalized people. *ACM Trans. Comput.-Hum. Interact.*, 28(2).
- Liang, J. T., Yang, C., e Myers, B. A. (2024). A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA. Association for Computing Machinery.
- Liu, Y., Chen, X., Gao, Y., Su, Z., Zhang, F., Zan, D., Lou, J.-G., Chen, P.-Y., e Ho, T.-Y. (2023). Uncovering and quantifying social biases in code generation. NIPS '23, Red Hook, NY, USA. Curran Associates Inc.
- Mateus, D. A., Silva, C. A., Eler, M. M., e Freire, A. P. (2020). Accessibility of mobile applications: evaluation by users with visual impairment and by automated tools. In *Proceedings of the 19th Brazilian Symposium on Human Factors in Computing Systems*, IHC '20, New York, NY, USA. Association for Computing Machinery.

- McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., e Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education*, 18.
- MoghadasNian, S., MoghadasNian, S., e MoghadasNian, A. (2025). Generative ai in airline tourism: Enhancing personalization with equity and accessibility. In *Proceedings of the 4th International Congress on Management, Economy, Humanities and Business Development*. Tabriz Islamic Art University, Tabriz, Iran.: English.
- Mowar, P., Peng, Y.-H., Steinfeld, A., e Bigham, J. P. (2024). Tab to autocomplete: The effects of ai coding assistants on web accessibility. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 1–6.
- Mowar, P., Peng, Y.-H., Wu, J., Steinfeld, A., e Bigham, J. P. (2025). Codea11y: Making ai coding assistants useful for accessible web development. *arXiv preprint arXiv:2502.10884*.
- Mozilla (2024). Aria - acessibilidade - mdn web docs. <https://developer.mozilla.org/pt-BR/docs/Web/Accessibility/ARIA>. Accessed on April 23, 2024.
- Next.js (2024). Next.js by vercel - the react framework. <https://nextjs.org/>. Accessed on April 23, 2024.
- Nguyen, N. e Nadi, S. (2022). An empirical evaluation of github copilot's code suggestions. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 1–5.
- npm, Inc. (2024). pally — npmjs.com. <https://www.npmjs.com/package/pally>. Accessed on May 06, 2024.
- Ouyang, S., Zhang, J. M., Harman, M., e Wang, M. (2023). Llm is like a box of chocolates: the non-determinism of chatgpt in code generation. *arXiv preprint arXiv:2308.02828*.
- Ozkaya, I. (2023). The next frontier in software development: Ai-augmented software development processes. *IEEE Software*, 40(4):4–9.
- Pally (2023). Pally — pally.org. <https://pally.org/>. Accessed on May 06, 2024.
- Palen, L. e Salzman, M. (2002). Voice-mail diary studies for naturalistic data capture under mobile conditions. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 87–95.
- Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., e Karri, R. (2022). Asleep at the keyboard? assessing the security of github copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 754–768. IEEE.
- Portugal (2004). Law 38/2004, dated august 18, 2004. <https://diariodarepublica.pt/dr/detalhe/lei/38-2004-480708>. Accessed on May 06, 2024.
- Rajbhoj, A., Somase, A., Kulkarni, P., e Kulkarni, V. (2024). Accelerating software development using generative ai: Chatgpt case study. In *Proceedings of the 17th Innovations in Software Engineering Conference, ISEC '24*, New York, NY, USA. Association for Computing Machinery.

- Ross, S. I., Martinez, F., Houde, S., Muller, M., e Weisz, J. D. (2023). The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, IUI '23, page 491–514, New York, NY, USA. Association for Computing Machinery.
- Runeson, P. e Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14:131–164.
- Schmidt, A. (2023). Speeding up the engineering of interactive systems with generative ai. In *Companion Proceedings of the 2023 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '23 Companion, page 7–8, New York, NY, USA. Association for Computing Machinery.
- Silva, J. S. e., Gonçalves, R., Branco, F., Pereira, A., Au-Yong-Oliveira, M., e Martins, J. (2019). Accessible software development: a conceptual model proposal. *Universal Access in the Information Society*, 18:703–716.
- Teran, L. A., da Rocha, T. A., da Silva, I. M. M., de Salles, R. B., Melo, G. L. N., Mota, M. P., e de Souza, C. R. B. (2025a). Github copilot diary - researcher 1. <https://bit.ly/githubcopilot-diary-research-1>. Accessed on August 08, 2025.
- Teran, L. A., da Rocha, T. A., da Silva, I. M. M., de Salles, R. B., Melo, G. L. N., Mota, M. P., e de Souza, C. R. B. (2025b). Github copilot diary - researcher 2. <https://bit.ly/githubcopilot-diary-research-2>. Accessed on August 08, 2025.
- Teran, L. A., de Almeida Silva, A. T., Melo, G. L. N., e Mota, M. P. (2021). A study on discovering accessibility issues in the software development process. In *CONTECSI International Conference on Information Systems and Technology Management*. TECSI.
- Terragni, V., Vella, A., Roop, P., e Blincoe, K. (2025). The future of ai-driven software engineering. *ACM Trans. Softw. Eng. Methodol.* Just Accepted.
- USA (2024). Ada.gov: The americans with disabilities act. <https://www.ada.gov>. Accessed on May 06, 2024.
- Whitaker, R. (2016). Mobile a11y. <https://mobilea11y.com/>. Accessed on April 21, 2024.
- World Wide Web Consortium (2016). User agent accessibility guidelines (uaag) overview. <https://www.w3.org/WAI/standards-guidelines/uaag/>. Accessed on April 21, 2024.
- World Wide Web Consortium (2023a). Authoring tool accessibility guidelines (atag) overview. <https://www.w3.org/WAI/standards-guidelines/atag/>. Accessed on April 21, 2024.
- World Wide Web Consortium (2023b). Understanding SC 1.3.1: Info and Relationships. <https://www.w3.org/WAI/WCAG21/Understanding/info-and-relationships>. Accessed on May 05, 2024.
- World Wide Web Consortium (2023c). Understanding SC 1.4.3: Contrast (Minimum). <https://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum>. Accessed on May 05, 2024.

- World Wide Web Consortium (2023d). Understanding SC 2.4.2: Page Titled. <https://www.w3.org/WAI/WCAG21/Understanding/page-titled.html>. Accessed on May 05, 2024.
- World Wide Web Consortium (2023e). Understanding SC 4.1.2: Name, Role, Value. <https://www.w3.org/WAI/WCAG21/Understanding/name-role-value>. Accessed on May 05, 2024.
- World Wide Web Consortium (2023f). Understanding understanding sc 2.4.1: Bypass blocks (level a). <https://www.w3.org/WAI/WCAG21/Understanding/bypass-blocks>. Accessed on February 14, 2025.
- World Wide Web Consortium (2024a). Introduction to Web Accessibility. <https://www.w3.org/WAI/intro/accessibility.php>. Accessed on May 06, 2024.
- World Wide Web Consortium (2024b). Wcag 2 overview. <https://www.w3.org/WAI/standards-guidelines/wcag/>. Accessed on May 06, 2024.
- Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., e Aftandilian, E. (2022). Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, MAPS 2022, page 21–29, New York, NY, USA. Association for Computing Machinery.