Accessibility Evaluation of LLM-Generated Android Code in React Native

Carlos David Ventura[1], **Daniel Mesquita Feijó Rabelo**[2], **Ticianne Darin**[1,2], **Windson Viana**[1,2]

[1]Virtual Institute - Federal University of Ceará (UFC)
ZIP Code 60.440-554 – Fortaleza – CE – Brazil

[2]Master's and Ph.D. in Computer Science - Federal University of Ceará (UFC)
ZIP Code 60.440-900 – Fortaleza – CE – Brazil

{cdavidav19,dmesquita861}@gmail.com, {ticianne,windson}@virtual.ufc.br

Abstract. Introduction*: LLMs usage in interface prototyping raises concerns about their ability to integrate accessibility.* ***Objective:*** *In this study, we evaluated the accessibility of React Native code generated by ChatGPT 4.o mini, DeepSeek V3, and Gemini 2.0 Flash, comparing prompt strategies and languages (English vs. Portuguese).* ***Methodology or Steps****: We analyzed 252 screen samples using the Accessibility Scanner and found 1,159 accessibility errors.* ***Results:*** *Few-shot prompts outperformed zero-shot ones, while the language used had no significant effect. Our results highlight the current technical limitations of LLMs in generating accessible mobile screens and also the critical human factors involved, emphasizing the need for improved methodologies that consider the developer's role in using these tools for inclusive design.*
Keywords *Large Language Models, Mobile Accessibility, React Native, Prompt Engineering*

Resumo. Introdução: *O uso de LLMs na criação de interfaces gera dúvidas sobre sua capacidade de incluir acessibilidade.* ***Objetivo:*** *Neste estudo, a acessibilidade do código React Native gerado pelo ChatGPT 4.o mini, DeepSeek V3 e Gemini 2.0 Flash foi avaliada com a comparação de tipos de prompts e idiomas distintos.* ***Metodologia ou Etapas****: Analisaram-se 252 amostras de telas com o Accessibility Scanner, que encontrou 1.159 erros de acessibilidade.* ***Resultados:*** *Os prompts do tipo few-shot superaram os zero-shot, enquanto o idioma usado não teve efeito significativo. Esses resultados destacam as limitações dos LLMs na geração de telas acessíveis, enfatizando a necessidade do estudo de metodologias que considerem o papel do desenvolvedor no uso dessas ferramentas no design inclusivo.*
Palavras-Chave *Modelos de Linguagem de Grande Escala , Acessibilidade Móvel , React Native, Engenharia de Prompts*

1. Introduction

Recent advances in artificial intelligence (AI) have placed Large Language Models (LLMs), such as ChatGPT, Gemini, and DeepSeek, as influential tools in software design and development. These models are capable of generating source code, addressing

complex tasks, and reducing manual effort, thus reshaping practices in mobile and digital systems engineering [Hou et al. 2024, Chen et al. 2021, Zhao et al. 2025]. While LLMs introduce gains in efficiency and automation, they also reconfigure the roles and interactions of design/development teams, particularly in relation to critical use-oriented concerns such as usability and accessibility of the systems generated. From a Human-Computer Interaction (HCI) perspective, it is essential to examine how these tools mediate design and development workflows, the forms of support they enable, and the frictions they generate with respect to inclusive design practices.

Among the challenges in developing web and mobile applications is the implementation of accessibility. Digital accessibility aims to ensure that websites, tools, and technologies are usable by people with visual, auditory, or motor impairments. However, this concern does not only benefit this group; in fact, accessibility enhances the experience of all users [Henry et al. 2014, Campoverde-Molina et al. 2020]. Standards organizations and platform maintainers have established guidelines to help designers and developers create accessible interfaces. For example, the Web Content Accessibility Guidelines (WCAG) [World Wide Web Consortium (W3C) 2023], ABNT NBR 17060 [1], and the Android Accessibility Developer Guidelines [Android Developers 2025] provide technical criteria to ensure accessible design. However, studies show that many mobile applications do not comply with accessibility guidelines [Vendome et al. 2019, Andrade et al. 2024, Alshayban et al. 2020, Nunes et al. 2023, Mateus et al. 2023]. This often results from limited developer knowledge or poor integration of best accessibility practices into design and development workflows [Leite et al. 2021, Nunes 2025]. These gaps underscore the need for tools that support the automation of accessibility implementation and testing.

The hypothesis is that generative AI emerges as a tool that could increase accessibility in mobile application development. LLMs, with their ability to understand and generate natural language, can assist developers in creating more intuitive and adaptable interfaces, facilitating the navigation and use of applications by people with disabilities. Recent studies have examined this hypothesis in the context of both web page creation [Ahmed et al. 2025] and mobile interface design for native Android applications [Rabelo et al. 2025]. The findings show that, under certain prompt types, accessibility issues in the generated interfaces/screens cannot be overlooked. Within Human-Computer Interaction (HCI), this challenge extends beyond technical automation as discussed in GranDIHC-BR 2025-2035- GC6 [Duarte et al. 2024]. It also involves understanding how such tools shape the design of inclusive digital experiences [Duarte et al. 2024]. Our study investigates this intersection, focusing on how LLMs generate React Native code for accessible mobile interfaces.

Developing applications for mobile devices presents several approaches. The most common uses *native techniques*, which employ languages, tools, and development environments specific to each operating system. *Cross-platform* development, by contrast, is an umbrella term for techniques that seek to circumvent the need for separate codebases, aiming at code reuse across different platforms, such as Android and iOS [Biørn-Hansen et al. 2018]. Within the landscape of mobile development, React Native [Meta Platforms, Inc. 2025] has emerged as a prominent interpreted framework. Data

[1] https://www.abntcolecao.com.br/mpf/norma.aspx?ID=516652

concerning the top 500 most installed applications in the US suggest a significant adoption of React Native, with findings indicating that 16.95% of these apps were developed using this framework, surpassed only by native solutions such as Android Architecture Components and Kotlin [AppBrain 2025].

In this context, our study evaluated the capability of three LLMs (i.e., *ChatGPT 4.o mini*, *DeepSeek V3*, and *Gemini 2.0 Flash*) to generate accessible *React Native* code. Prompt engineering plays a critical role in shaping LLM outputs [Fagadau et al. 2024]. Previous work has shown that prompting techniques such as *zero-shot* and *few-shot* lead to different performance outcomes [Brown et al. 2020, Suh et al. 2025], but their impact on accessibility-aware code generation remains unclear [Suh et al. 2025, Ahmed et al. 2025]. To address this gap, we analyzed how prompt strategies (*zero-shot* vs. *few-shot*) and prompt language (*English* vs. *Portuguese*) affect the accessibility of generated code. We produced 252 screen samples across seven common interface types and evaluated them using the Google *Accessibility Scanner*. The evaluation revealed 1,159 accessibility errors. The *few-shot* approach consistently outperformed *zero-shot* in reducing errors. However, prompt language did not show a statistically significant impact on the number of issues detected.

The remainder of the paper is structured as follows: Section 2 presents a review of the existing literature and studies related to mobile and Web accessibility and LLMs. Following this, Section 3 details the research approach, including the methods and procedures used to screen generation and accessibility analysis. Section 4 then presents the findings obtained from the application of the methodology. Subsequently, Section 5 provides an analysis and interpretation of the results in the context of research questions. Finally, Section 6 presents final considerations, summarizing the key contributions and describing potential future work.

2. Background and Related Work

2.1. Mobile Accessibility

Mobile accessibility ensures that individuals with disabilities can effectively interact with smartphone applications through built-in assistive technologies. For example, Android's TalkBack offers spoken feedback, allowing users with visual impairments to navigate interfaces using gestures rather than standard touch. People with motor impairments can also benefit from alternative input methods, such as external switches and adaptive keyboards. Additional accessibility features include voice control, font scaling, color filters, click delay, and screen magnification. To support these tools, applications must be developed in accordance with established accessibility guidelines.

Accessibility in mobile applications has gained increasing attention in recent research. Studies have examined different dimensions of this topic [Andrade et al. 2024, Vendome et al. 2019, Alshayban et al. 2020, Darvishy 2022, Mateus et al. 2021, Mateus et al. 2023, Nunes et al. 2023], including empirical evaluations of accessibility in existing applications [Vendome et al. 2019, Andrade et al. 2024] and the practical challenges that developers face when applying accessibility guidelines during development [Alshayban et al. 2020, Leite et al. 2021].

For example, the study [Vendome et al. 2019] investigated Android accessibility by mining 200 apps for API accessibility usage and analyzing 366 developer discussions

on Stack Overflow. They observed limited use of assistive descriptions, even in accessibility-focused applications, and developers struggled with screen readers, navigation, and the correct application of accessibility guidelines. Building on this understanding of accessibility challenges in existing apps, a study conducted a broader analysis of more than 1,000 Android applications [Alshayban et al. 2020]. The authors attested to the prevalence of accessibility problems, finding that nearly all applications presented issues that hindered the use of people with disabilities. They also explored developers' perspectives, revealing a lack of awareness regarding accessibility design principles and analysis tools, coupled with a low prioritization of accessibility within their organizations.

The studies cited previously did not specify the mobile programming approach used (e.g., Android native, cross-platform frameworks), nor did they examine whether this factor affects accessibility. In contrast, the study [Mascetti et al. 2021] focused specifically on implementing accessibility in mobile applications using cross-platform frameworks. The authors investigated the use of Xamarin and React Native. Their analysis revealed that while many accessibility features are shared between native iOS and Android APIs, most of these functionalities are not available in React Native or Xamarin. Another example is the study [Darvishy 2022], which investigated whether the Flutter framework [Google 2025a] supports the development of accessible apps for both iOS and Android using a single codebase. They developed a sample app and conducted usability tests with six blind users. After the initial test, they applied adjustments using Flutter's Semantics classes, including the addition of hints, feedback after interactions, removal of decorative elements, and improvements to the reading flow. Their findings suggest that Flutter offers support for accessible cross-platform development, but achieving full compliance may still require manual platform-specific interventions.

2.2. Accessibility and LLMs

More recently, researchers were interested in understanding the role that generative AIs could play in the accessibility of digital systems. For example, the authors of a case study investigated how ChatGPT-4o can assist in the development of accessible web pages by generating and iteratively correcting a page for a fictitious conference [Ahmed et al. 2025]. They observed that while the code generated by the LLM by default often did not meet the WCAG guidelines, ChatGPT was able to correct many accessibility issues when explicitly instructed, including with the aid of screenshots for visual analysis by the model. However, the study highlighted that the LLM faced difficulties with more complex tasks and that human supervision and multiple iterations were crucial to achieve an acceptable level of compliance.

Another study evaluates the accessibility of LLM-generated code with code originally written by human developers in 10 real-world web projects [Suh et al. 2025]. Their analysis revealed that the LLM-generated code exhibited, on average, fewer common accessibility errors, such as color contrast issues and missing alternative text for images. However, in scenarios considered more complex, especially those requiring the correct use of ARIA attributes and the assurance of adequate semantic and navigation structures, the code originally written by humans tended to be superior, or the LLMs introduced new errors. In [Rabelo et al. 2025], the authors analyzed accessibility aspects in mobile code generated by different versions of ChatGPT and the Brazilian language

model, Sabiá, for native Android applications. The findings indicated that LLMs have difficulty in generating accessible code and, surprisingly, explicitly prompting for accessibility sometimes led to an increase in accessibility errors.

These studies have evaluated the accessibility of web and native mobile applications. However, there is a noticeable gap in research on the level of accessibility in applications developed with cross-platform frameworks. In this context, React Native stands out as an emerging framework focused on cross-platform development[2]. Moreover, expanding the evaluation of the impact of different prompt types can help to understand how to improve LLM suggestions in the process of generating more accessible screens.

Furthermore, broader HCI research explores the challenges and promises of AI in development and design, e.g., automation bias, interpretability of AI suggestions, and the role shift of the developer [Barmer et al. 2021, Zhang et al. 2021]. Our research connects to these discussions through examining the everyday implications and interaction patterns when LLMs are explicitly used to generate accessible code in a cross-platform environment, with the aim of providing empirically grounded insights into this specific human-AI collaboration space.

3. Methodology

The objective of this study was to analyze the ability of LLMs to generate code with accessibility for different screens of cross-platform mobile applications using React Native. To achieve this, a methodology similar to that of other studies [Suh et al. 2025, Rabelo et al. 2025] was followed, although we analyzed different technologies and LLMs. Figure 1 summarizes the steps followed. Three research questions were established:

> **RQ1** - Which prompt strategy proves to be the most efficient with respect to the level of accessibility of the generated code?
> **RQ2** - To what extent does the language used in the prompts (e.g., Portuguese vs. English) affect the accessibility level of code generated by large language models?
> **RQ3** - Which categories of accessibility errors are the most frequent on the screens generated by LLMs?

3.1. LLMs Selection

The selection of LLMs was driven by convenience. Our intent was to explore both widely adopted systems from major AI companies and a prominent Chinese startup that has recently gained media attention.

ChatGPT 4.o mini, developed by OpenAI [OpenAI 2025], was included due to its popularity; it is essential to evaluate its capabilities and limitations in the context of accessibility. Furthermore, previous studies [Rabelo et al. 2025, Suh et al. 2025, Delnevo et al. 2024, Aljedaani et al. 2024, Othman et al. 2023] have already investigated ChatGPT in relation to the accessibility of web and mobile code, providing a useful point

[2]The importance of React Native is further highlighted by the [Stack Overflow 2024] survey, which identified it as the second most widely used cross-platform framework among professional developers in 2024, slightly surpassed by Flutter.

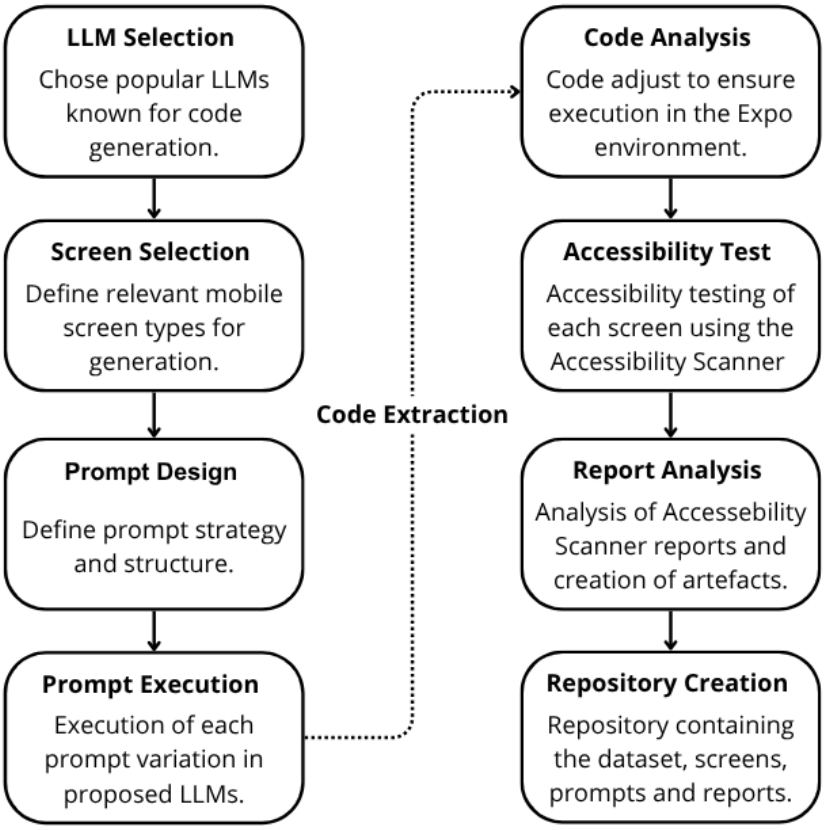

Figure 1. Methodology Workflow

of comparison for this research. **Gemini 2.0 Flash** [Google 2025b], developed by Google, was selected for its multimodal approach and integration with Google's infrastructure. Evaluating Gemini allows us to explore whether models designed with a focus on enhanced human-machine interaction offer unique advantages in generating accessible code. **DeepSeek V3** [DeepSeek 2025] represents a more recent LLM, which allows us to examine whether newer models present advances in the generation of accessible code compared to more established models such as ChatGPT.

3.2. Screens Selection

The selection of the evaluated screens was based on the study [Rabelo et al. 2025], in which the authors performed an exploratory visual examination of the 50 most popular apps in the Google Play Store to identify the most common screen types. The **seven** chosen screens were: Login, Registration, Product Details, Profile, Music Player, E-Commerce Side Menu, and To-Do List. They chose them because of their frequency in mobile apps and the potential interest of developers in generating these screens using LLMs.

By replicating the screen selection of that study, we sought to facilitate future comparisons of results and the analysis of accessibility on screens generated by LLMs in different mobile development approaches.

3.3. Prompt Design and Execution

The prompts used in this study were designed using two distinct strategies: **zero-shot** and **few-shot**. Table 1 shows the general idea of these prompts. Both approaches shared a foundational element, a textual description of the interface screen under consideration. Building on this base and informed by existing literature on common accessibility violations [Alshayban et al. 2020, Andrade et al. 2024, Suh et al. 2025, Muniz et al. 2024], we specifically targeted touch target size, text contrast, and content labeling as key areas of investigation.

The primary distinction between the zero-shot and few-shot approaches lies in the provision of examples [Brown et al. 2020]. In the zero-shot method, we instructed the LLM to adhere to accessibility guidelines and cautioned against the categories most frequently violated. In contrast, the few-shot strategy involved providing LLMs with illustrative examples of both effective and ineffective implementations, based on the React Native accessibility API documentation[3], for these same accessibility categories.

The following example presents a zero-shot prompt, providing the language model only with a task description without prior examples.

> Provide a React Native code snippet for a music player screen that adheres to accessibility best practices.
> The player should include standard playback controls, such as buttons for play/pause, skip forward, rewind, and a progress bar. It also includes a playlist and displays information about the current song.
> Ensure the code addresses the following accessibility considerations:
> - Sufficient text contrast.
> - Proper labeling for screen readers.
> - Adequate touch target sizes for all interactive elements.

The subsequent example illustrates an accessibility category for a few-shot prompt, which precedes the task description.

> Touch Target Size (using Touchable components and padding/min dimensions):
> Bad:
> - Component: `<TouchableOpacity style={{ width: 16, height: 16 }}>...</TouchableOpacity>`
> - Description: The touch target is too small for easy interaction.
>
> Good:
> - Component: `<TouchableOpacity style={{ minWidth: 40, minHeight: 40, padding: 4 }}>...</TouchableOpacity>`
> - Description: The touch target meets the recommended minimum size of 48x48 pixels, with added padding for visual spacing and easier interaction.

Each unique prompt variation was executed three times to assess the consistency and variability of the generated code when the same prompt was applied repeatedly. Figure 2 illustrates the structured execution process of the prompts for the music player

[3]https://reactnative.dev/docs/accessibility

Table 1. Prompt types and their structures.

Type	Structure
Base prompt	Screen description detailing the required elements that must be present in each one.
Zero-shot	React Native screen generation request + **Base prompt** + accessibility request without examples.
Few-shot	Good and bad accessibility examples of the most violated accessibility categories + React Native screen generation request + **Base prompt**.

screen; this procedure was then replicated for the remaining screens in each LLM, totaling 252 screens.

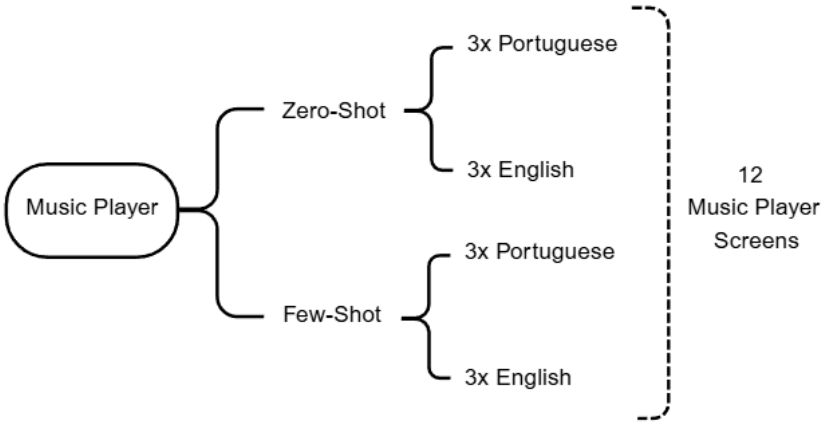

Figure 2. Prompt execution structure example for Music Player screen.

3.4. Materials and Methods

3.4.1. Code analysis - Expo

After the code generation phase, each output was exported to a development environment using Expo [Expo 2025], a framework and platform built on top of React Native that simplifies the development, building, and deployment of React Native applications. It is worth noting that for some screens, the LLMs incorporated external libraries, requiring their installation for the code to run correctly. Furthermore, instances were observed where the LLMs added non-existent props to certain components, and these invalid props were subsequently omitted. In other scenarios, while the LLMs correctly passed necessary data as props to the screen components, the Expo environment did not inherently expect these props in the page components. Consequently, LLMs were instructed to code the required data directly within those components instead of passing them as props. This step aimed to identify and resolve any potential errors or issues

that could hinder the successful execution and rendering of the generated React Native components.

3.4.2. Accessibility Test

The accessibility of each generated screen was evaluated using the Google Accessibility Scanner [Google LLC 2025]. This tool analyzes the visual and auditory accessibility of Android applications by identifying potential issues related to content labeling, touch target size, contrast, and more. The Accessibility Scanner works by inspecting the UI hierarchy and properties of on-screen elements, providing feedback and suggestions for improvement based on established accessibility guidelines.

The tests were carried out by a single evaluator using a Samsung Galaxy M52 5G device. React Native screens were run through the Expo Go application [Expo Project 2025], which allows developers to preview React Native projects directly on a physical device without building standalone applications, as shown in Figure 3. Importantly, errors related to image contrast were intentionally ignored, as the images displayed were merely placeholders generated by LLMs and did not represent actual UI elements relevant to the app's functionality. As such, they were not treated as genuine accessibility violations.

4. Results

A summary of accessibility error results by prompt type is presented in Table 2. The table also indicates the number of errors identified, the average number of errors, and the total standard deviation for each prompt. Among the 252 screens evaluated, a total of 1159 accessibility errors were identified, with an average of 4.6 errors per screen. The zero-shot prompts accounted for more errors (705) than the few-shot prompts (454).

To analyze the difference between the two approaches, we compared the 126 prompts generated with few-shot with their corresponding zero-shot versions. Applying a paired Student's *t*-test to these comparisons, we obtained the following results: $t = -4.821, p = 0.000004$. The result ($p < 0.05$) indicates that the differences are statistically significant.

Table 2. Accessibility Errors by Prompt Strategy.

Prompt	Errors	Mean Errors per Screen	Standard Deviation
Few-Shot	454	3.60	3.53
Zero-Shot	705	5.60	5.67
Total	1159	4.60	4.82

Figure 4 presents a boxplot that illustrates the number of accessibility errors found in all attempts for each investigated LLM, separating the results by prompt strategy. The graph allows for a visual comparison of the variation and consistency of performance in the results between the different LLMs and strategies. DeepSeek V3 tended to exhibit the

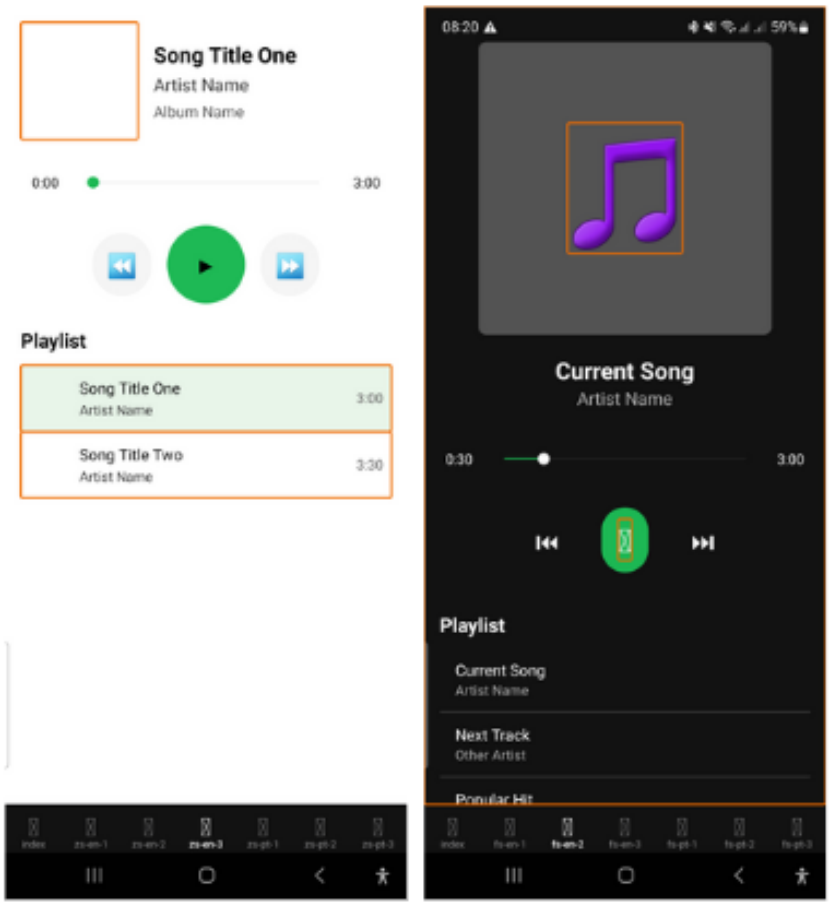

Figure 3. Example of Accessibility Scanner test for Music Player screens generated by DeepSeek V3 with zero-shot (left) and few-shot (right) approaches.

lowest number of errors and fewer variation in results for both strategies, while ChatGPT 4.o mini and Gemini 2.0 Flash displayed a higher number of errors and greater variability with the zero-shot strategy, but demonstrated fewer errors and more consistency with the few-shot strategy.

Table 3 details the total number of accessibility errors detected on all generated screens for each model, as well as the mean number of errors per screen and the standard deviation. DeepSeek V3 exhibited the lowest number of errors (289). ChatGPT 4.o mini recorded the highest number of errors (487). Gemini 2.0 Flash showed intermediate results with 383 errors.

Table 3. Accessibility Errors by LLM.

LLM	Errors	Mean Errors per Screen	Standard Deviation
ChatGPT 4.o mini	487	5.80	5.61
DeepSeek V3	289	3.44	3.68
Gemini 2.0 Flash	383	4.56	4.59

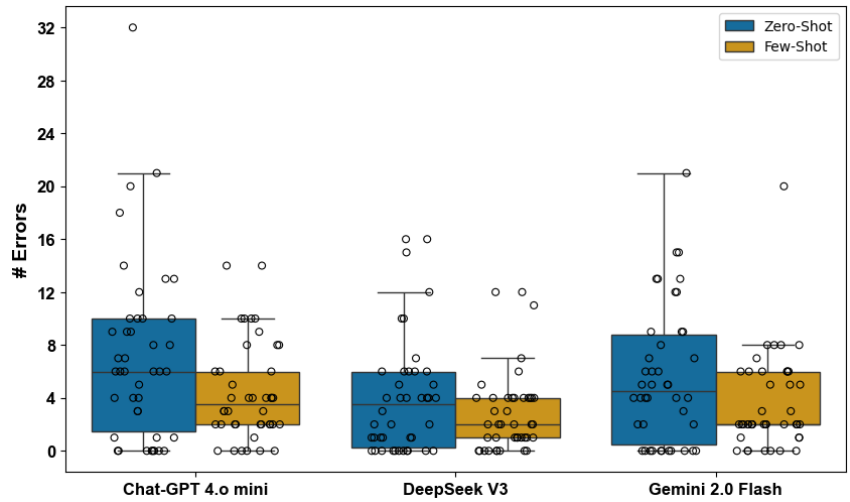

Figure 4. Boxplot showing the number of errors for each of the 252 screens categorized by prompt strategy and LLM model.

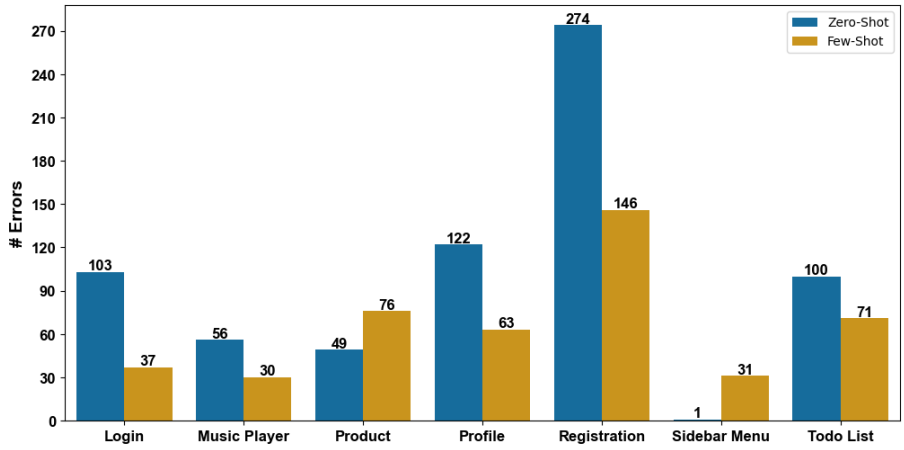

Figure 5. Accessibility errors by type of screen and prompt strategy.

We applied the T-Student test by pairing the evaluation results two by two. In all three combinations, the difference in the number of errors was significant ($p < 0.05$). For ChatGPT 4.o mini and DeepSeek V3, the t-value was 4.373 and the p-value was 0.00004. For ChatGPT 4.o mini and Gemini 2.0 Flash, the t-value was 2.449 and the p-value was 0.01641. And finally, for DeepSeek V3 and Gemini 2.0 Flash, the t-value was -2.759 and the p-value was 0.00713.

Figure 5 illustrates the total number of accessibility errors detected in the seven evaluated screen types, contrasting the results of the zero-shot and few-shot prompt strategies. Considerable variation in error count is evident among the different screen types. The “Registration” screen exhibited the highest number of errors for zero-shot (274) and few-shot (146), while the “Music Player” screen recorded the fewest errors with the few-shot approach (30), and the “Sidebar Menu” screen had the lowest error count with zero-shot (1). In particular, the few-shot strategy resulted in a higher number of errors compared to zero-shot only for the “Product” and “Sidebar Menu” screens.

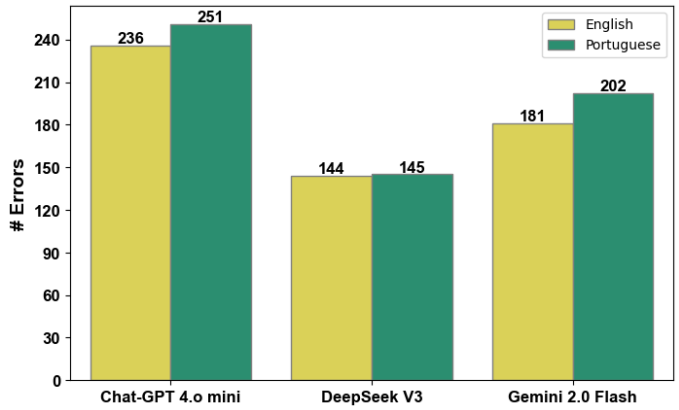

Figure 6. Total accessibility errors by LLM and language used in prompt.

Table 4. Accessibility Errors by Language.

Language	Errors	Mean Errors per Screen	Standard Deviation
Portuguese	598	4.75	5.17
English	561	4.45	4.46

Figure 6 presents a comparison of the total number of accessibility errors for each LLM based on the language used in the prompts. For ChatGPT 4.o mini, 236 errors were recorded with English prompts and 251 with Portuguese prompts. DeepSeek V3 exhibited the lowest totals, with 144 errors for English and 145 for Portuguese. Finally, Gemini 2.0 Flash accumulated 181 errors with English prompts and 202 with Portuguese prompts. Across the three LLMs evaluated, the use of Portuguese prompts resulted in a slightly higher total number of errors compared to that observed with English prompts.

In a complementary way, Table 4 displays the total number of errors across all generated screens as well as the average errors and standard deviation based on the language used on the prompts, with Portuguese totaling 598 errors and an average of 4.75 errors per screen, and English totaling 561 errors with an average of 4.45 errors per screen.

To analyze the difference between the two languages, we compared the 126 prompts generated with Portuguese prompts with their corresponding English prompts. Applying a paired Student's *t*-test to these comparisons, we obtained the following results: $t = 0.824, p = 0.41126$. The result ($p > 0.05$) indicates that the differences are not statistically significant between languages. Figure 7 presents a comparative analysis of the performance of LLMs on the seven types of screens. For each screen, the results of the three individual attempts are displayed. Accessibility varies between prompts for each screen. For example, in "Profile" in Gemini 2.0 Flash, one prompt resulted in 5 errors, another had 12, and the third had 8. Also, in "Registration" in ChatGPT 4.o mini, one prompt resulted in 10 errors, another had 17, and the third had 26.

Figure 8 displays the aggregated number of accessibility errors per category for each of the evaluated screens. Text Contrast exhibited the highest number of errors, with

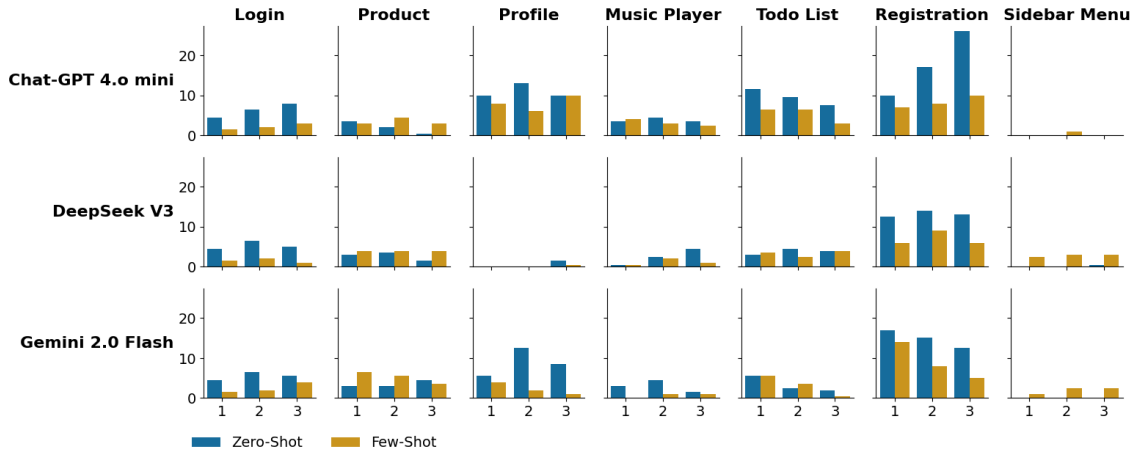

Figure 7. Variability and inconsistency of accessibility errors across LLMs for each screen type and prompt attempts.

a total of 328 occurrences, followed by Hidden Text and Touch Target Size, both with 263 occurrences each. Editable Item Label also presented a high frequency of errors, totaling 206 errors. In contrast, the categories Context (3), Item Type Label (4), Clickable Items (6), and Item Label (14) were considerably less frequent.

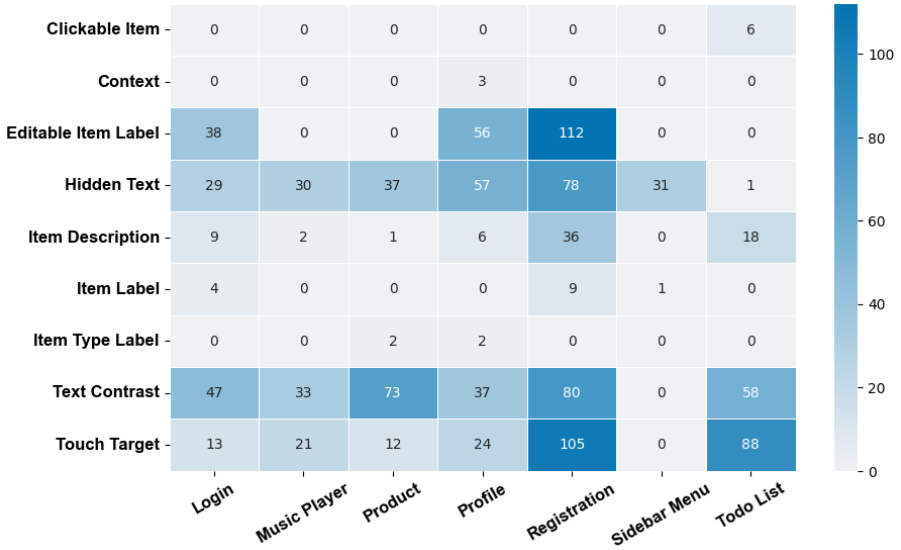

Figure 8. Types of Accessibility Issues Detected by Type of Screen.

Figure 9 further illustrates the findings on the total accessibility errors per category based on the prompt strategy used. With the notable exceptions of Item Label and Text Contrast, all other accessibility categories exhibited a reduction in the total number of errors when the few-shot prompting strategy was used compared to the zero-shot approach.

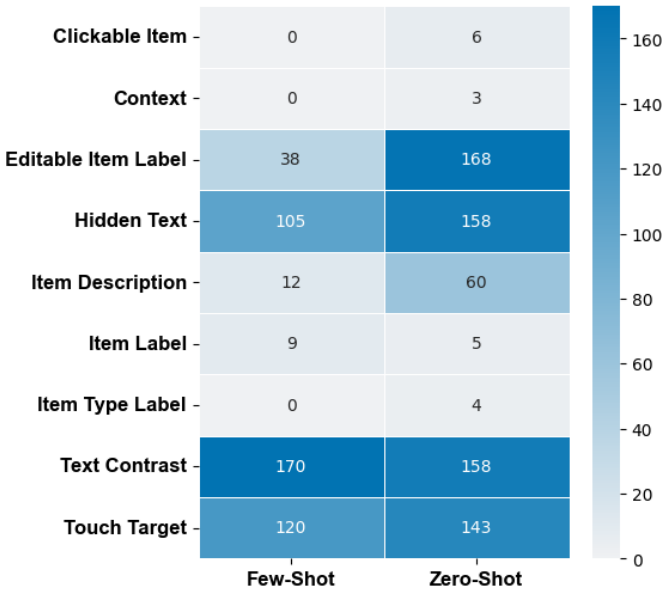

Figure 9. Types of Accessibility Issues Detected by Prompting Strategy.

5. Discussion

5.1. RQ1 - Which prompt strategy proves to be the most efficient with respect to the level of accessibility of the generated code?

Main result: The few-shot prompt strategy generally resulted in fewer accessibility errors compared to the zero-shot strategy in the evaluated LLMs.

Our results indicate that the few-shot prompt strategy was statistically more efficient in generating React Native code with fewer accessibility issues compared to zero-shot for ChatGPT 4.o mini and Gemini 2.0 Flash. For DeepSeek V3, the few-shot resulted in a lower number of issues, although the difference was not significant. The overall analysis showed an average of 3.6 errors per screen for few-shot versus 5.6 errors for zero-shot. This suggests that providing LLMs with concrete examples of accessible and inaccessible implementations for common issues is more effective than simply instructing the model to follow accessibility principles abstractly, as done in zero-shot.

These findings partially contrast with another study [Suh et al. 2025], in which the few-shot approach either underperformed compared to zero-shot or did not produce significant improvements. This discrepancy arises despite the fact that Suh's investigation explored different models and technologies, highlighting the context-dependent nature of prompt engineering effectiveness across varying LLM and development frameworks. However, our results align with the broader literature on prompt engineering [Brown et al. 2020, Fagadau et al. 2024], which demonstrates the influence of prompt strategy on code generation results and the general benefit of few-shot learning.

From an HCI perspective, the superior performance of the few-shot method is not merely a technical advance. It can also be seen as an issue of the usability of LLMs as a development tool to achieve specific quality attributes such as accessibility. Showing explicit examples, as in few-shot, appears to be a more effective interaction

for developers to convey complex requirements to the AI. This suggests that on tasks of subtle comprehension, such as implementing accessibility, zero-shot instruction (direct instruction) can be less optimal than learning via example. This finding can influence the design of future LLM-based developer tools, possibly in the direction of interfaces that make it easy to give good examples in different modalities.

5.2. RQ2 - To what extent does the language used in the prompts (e.g., Portuguese vs. English) affect the accessibility level of code generated by large language models?

Main result: The language of the prompt (Brazilian Portuguese vs. English) did not significantly impact the accessibility of the generated React Native code as measured by the Accessibility Scanner.

Analysis of the results shows that the language used in the prompt (Brazilian Portuguese vs. English) did not have a significant impact on the accessibility level of code generated by the evaluated LLMs. Direct comparison between the Portuguese and English prompts revealed very close total errors (598 and 561, respectively), and the paired t-test did not show significant differences ($p = 0.41126$). This suggests that the language models used demonstrated the ability to interpret accessibility instructions and generate corresponding code in both languages, at least with regard to the errors detectable by the Accessibility Scanner.

This finding finds a parallel in the study by [Andrade et al. 2024], which investigated the impact of changing the execution language (Portuguese, English, and Spanish) on accessibility of existing popular Android applications, using the same detection tool (Accessibility Scanner). Similarly to our results with the prompt language, Andrade also did not find significant differences in the number of errors detected by the tool when varying the execution language. Although the contexts are distinct – prompt language for generated code versus execution language for existing apps – the convergence of results obtained with the same evaluation tool is notable and raises questions about the scanner's sensitivity to language-related nuances.

This seeming insensitivity of the Accessibility Scanner to prompt language, in analyzing the generated code, has significant HCI implications. While it may indicate strength in the LLMs' understanding across the scanned languages for the scanner-detectable problems, it also invites skepticism regarding communicability from an end-user's standpoint. Real accessibility can be subtly influenced by linguistic subtleties (e.g., clarity of labels, cultural appropriateness of described icons) that may not be grasped by an automated tool like the Accessibility Scanner. This highlights that automated accessibility checkers are useful but cannot fully substitute for human assessment, including user testing with native speakers of those languages.

5.3. RQ3 - Which categories of accessibility errors are the most frequent on the screens generated by LLMs?

Main result: The most frequent accessibility error categories were Text Contrast, Hidden Text, Touch Target, and Editable Item Label, with Text Contrast being the most prevalent overall.

Summing the errors from the few-shot and zero-shot strategies, the most frequent category was Text Contrast, with 328 occurrences. Following this, with identical frequency, were Hidden Text and Touch Target, both with 263 occurrences. Another notably frequent category was the Editable Item Label, with 206 errors. In contrast, other categories such as Item Label, Clickable Item, Item Type Label, and Context were relatively low, totaling less than 20 errors each.

The high frequency of Text Contrast and Touch Target errors echoes the findings of studies that analyzed existing human-developed [Andrade et al. 2024, Alshayban et al. 2020, Vendome et al. 2019]. This convergence suggests that these two categories represent persistent and fundamental challenges in the development of accessible mobile interfaces, regardless of the code's origin (human or AI). The idea that LLMs are replicating common human errors suggests that today's models, trained on massive public codebases, are learning and potentially propagating common poor practices around accessibility [Aljedaani et al. 2024]. This has profoundly valuable implications for the trust and confidence of developers in AI-generated code. If developers believe that the AI output is more inherently "correct" or "complete", they will exhibit automation bias and overlook these inherited flaws. This highlights the importance of developers staying critically involved with AI tools, not treating them as flawless oracles but as powerful assistants whose output should be inspected. This connects to broader HCI questions around the ethical responsibilities in putting AI systems into the wild potentially reproducing existing social biases or, in this case, accessibility barriers.

Another relevant point concerns the effectiveness of the few-shot strategy specifically for these categories. In Figure 9 it is observed that, for Text Contrast, the few-shot strategy not only failed to reduce errors compared to zero-shot but actually presented a slightly higher number (170 vs 158). This occurred despite contrast examples being included in the few-shot prompts, suggesting a potential intrinsic limitation of LLMs in calculating or correctly applying the contrast ratios required by the guidelines, perhaps due to the visual and computational nature of the requirement that may be difficult to translate with code examples alone.

In the case of Touch Target, although the few-shot strategy resulted in a reduction in the number of errors (120 vs. 143), the improvement may not have been as significant as expected and the number of errors remained high. A possible explanation lies in how the examples were provided in the few-shot prompt. In our examples, we used a combination of attributes such as width, height, and padding to achieve the minimum recommended dimension of 48dp (e.g., 40dp of height + 4dp of vertical padding). It is plausible that the LLMs had difficulty interpreting this combination, perhaps focusing only on the explicit height and width values (40dp) and ignoring the padding, resulting in the failure to consistently achieve the necessary 48dp, which limited the reduction of errors in this category.

This difficulty in effectively instructing LLMs on specific, nuanced accessibility rules like contrast ratios or combined touch target dimensions, even with examples, mirrors human-AI communication difficulties for complex design constraints. It suggests that the way developers now "explain" these kinds of rules through prompts (even few-shot) may not be well represented in how LLMs "learn" or prioritize them. This points to a need for research into more effective interaction mechanisms or

intermediate representations that can better bridge the gap between human design intent for accessibility and AI code generation.

5.4. Implications and Takeaways

Our investigation yields significant implications and practical recommendations for designers and developers who use these tools, as well as directions for future research. The study demonstrates that the way we instruct LLMs directly impacts the accessibility of the resulting code, with the few-shot strategy significantly outperforming the zero-shot approach in most cases. Our findings also suggest that **using Portuguese prompts did not significantly hinder accessibility compared to English**. This indicates the viability of non-English prompts for developers, but critical validation remains essential.

For researchers and tool developers, this may indicate limitations of the Accessibility Scanner in capturing language nuances, reinforcing the need for manual review, user testing, and research into cross-lingual evaluation. **LLMs frequently generate Text Contrast and Touch Target errors, reflecting challenges in human development**, so developers should carefully test these standard pitfalls. Comparing our findings with a similar study on native Android development by [Rabelo et al. 2025] offers additional insights. Although our study on React Native found an average of 4.6 errors per screen, the other reported a comparable average of 2.78 errors per screen. This suggests that generating accessible code might be more challenging for LLMs in React Native than in native development, or that specificities of the LLMs and prompts used here led to more errors. Expanding this comparison, our zero-shot approach, similar to their prompt that requires accessibility, resulted in a higher average error count (5.6 *vs.* 3.2). The performance of our ChatGPT 4.o mini also appeared inferior to the evaluated ChatGPTs [Rabelo et al. 2025], with caveats regarding different versions and frameworks.

In addition, it is important to note that the observed differences between our React Native study and the work on native Android could be partly attributed to the inherent characteristics of each framework. Standard React Native components possess distinct styling and accessibility properties compared to native ones, potentially introducing or mitigating certain errors.

Finally, our results reasonably indicate the need for Computer Science courses to prepare future developers as LLMs become more prevalent in software development, but going beyond the technical skills. This is in line with recent proposals within the HCI community to reform curricula to take into account AI and emerging technologies [Pereira et al. 2024]. The finding that contemporary LLMs replicate common accessibility issues highlights the need to extend education beyond mere technical competencies.

According to the GranDIHC-BR report, “Education in HCI must go beyond technical expertise, developing critical thinking, problem-solving skills, and a better understanding of ethical and social issues”. The results we discussed in this paper empirically support this conclusion. Developers in the future will not only have to be proficient in utilizing artificial intelligence tools but, more importantly, have the ability to critically evaluate their outputs, comprehend their limitations, and provide requisite interventions so that the resultant software is inclusive and ethically appropriate. This

requires cultivating a critical mentality to understand the implications of your design decisions when engaging in AI-generated code.

To equip students to succeed in this changing environment, include critical thinking and ethical design practices [Pereira et al. 2024] in their learning process about AI development and use. The goal is to create professionals who engage actively with AI recommendations rather than accepting them passively, as educated co-creators who can lead technological innovations to more fair, equitable, and responsible results, especially in regard to accessibility.

5.5. Threats to Validity and Limitations

In terms of construct validity, the study's findings depend on error counts from the Accessibility Scanner, a tool with known limitations. It may not capture all accessibility problems, such as those related to navigation logic, label context, or barriers identified only by assistive technologies such as TalkBack. Consequently, our results indicate conformity to this scanner, but it is not necessarily the complete real-world user experience. Furthermore, our evaluation of prompt efficiency focused on error quantity, omitting error severity or remediation effort, which influences actual accessibility impact.

Regarding internal validity, to investigate the inherent variability in LLM outputs for identical inputs, we executed each prompt three times. These executions used distinct LLM instances with memory features disabled (like those in ChatGPT) to ensure that each run was independent and relied solely on the current prompt. This process revealed some variation in error counts even with the same prompt in different independent executions, indicating a degree of inconsistency in how LLMs generate code and accessibility errors.

External validity includes the fact that our LLM selections were ChatGPT 4.o mini, DeepSeek V3, and Gemini 2.0 Flash. Therefore, the results may not generalize to other LLMs, particularly those specialized in coding. The findings are also specific to React Native code generation; LLM behavior and accessibility error patterns could substantially differ for native (Android/iOS) or other cross-platform frameworks. Finally, since we only tested the English and Brazilian Portuguese prompts, the error patterns might vary in other languages.

6. Final Considerations

Evaluating the accessibility of the React Native code generated by three LLMs and comparing prompt strategies and language, we found that, while they can accelerate development, they also introduce accessibility issues, which are indicative of limitations in human-AI collaboration for accessible design. Few-shot was overall superior to zero-shot in reducing them, which is reflective of the fact that the interaction performed by developers makes a difference, though performance varied by LLM and example clarity. The automated tools did not detect significant language differences in accessibility, reinforcing the HCI guideline that automated tools must be augmented with human-centered testing.

These results, along with previous literature, bring to light continued issues and emerging AI-based possibilities for mobile accessibility. Numerous current applications have defects, and LLMs excel only in producing partially accessible code. This calls

developers to critically evaluate the AI output and additional HCI research that will refine LLM-based techniques and develop effective solutions for mobile accessibility.

Follow-up studies should reach a wider range of LLMs, particularly advanced models fine-tuned for code generation. Researchers should analyze complex screen designs, explore additional mobile frameworks such as Flutter, and include target operating systems like iOS. Above all, automated testing must be augmented with qualitative testing, such as usability testing with people with disabilities, to reveal barriers that cannot be found by the tools. Understanding why LLMs make common errors may guide model development. More research is also required to optimize prompt engineering for accessibility and investigate LLM capability for autonomous bug identification and repair of existing code.

Ethical Considerations

In this research, we do not deal with human evaluations or sensitive data that violate the LGPD or the SBC code of ethics.

Acknowledgment and Additional Information

This research was partially funded by CNPQ (under grant number 314425/2021-7).

All prompts, generated code, and evaluation reports from the Accessibility Scanner are available on GitHub [`https://github.com/CarlosDavidAraujo/Dataset-LLM-Accessibility-React-Native`]. We used the LLMs ChatGPT and NotebookML to assist in text revision for this paper.

References

Ahmed, A., Fresco, M., Forsberg, F., e Grotli, H. (2025). From code to compliance: Assessing chatgpt's utility in designing an accessible webpage – a case study. In *arXiv: https://arxiv.org/abs/2501.03572*.

Aljedaani, W., Habib, A., Aljohani, A., Eler, M., e Feng, Y. (2024). Does chatgpt generate accessible code? investigating accessibility challenges in llm-generated source code. In *Proceedings of the 21st International Web for All Conference*, W4A '24, page 165–176, New York, NY, USA. Association for Computing Machinery.

Alshayban, A., Ahmed, I., e Malek, S. (2020). Accessibility issues in android apps: State of affairs, sentiments, and ways forward. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1323–1334.

Andrade, M., Rabelo, D., Martins, R., e Viana, W. (2024). Investigating the accessibility of popular mobile android apps: a prevalence, category, and language study. In *Proceedings of the 30th Brazilian Symposium on Multimedia and the Web*, pages 400–404, Porto Alegre, RS, Brasil. SBC.

Android Developers (2025). User interface accessibility. Available at: `https://developer.android.com/guide/topics/ui/accessibility?hl=pt-br`. Accessed on April 15, 2025.

AppBrain (2025). Android app frameworks statistics. Available at: `https://www.appbrain.com/stats/libraries/tag/app-framework/android-app-frameworks?list=top500`. Accessed on April 15, 2025.

Barmer, H., Dzombak, R., Gaston, M., Palat, V., Redner, F., Smith, C., e Smith, T. (2021). Human-centered ai. Report, Carnegie Mellon University. Available at https://doi.org/10.1184/R1/16560183.v1. Accessed on May 01, 2025.

Biørn-Hansen, A., Grønli, T.-M., e Ghinea, G. (2018). A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. *ACM Comput. Surv.*, 51(5).

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., e Amodei, D. (2020). Language models are few-shot learners. In *arXiv: https://arxiv.org/abs/2005.14165*.

Campoverde-Molina, M., Luján-Mora, S., e García, L. V. (2020). Empirical studies on web accessibility of educational websites: A systematic literature review. *IEEE Access*, 8:91676–91700.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., e Zaremba, W. (2021). Evaluating large language models trained on code. In *arXiv: https://arxiv.org/abs/2107.03374*.

Darvishy, A. (2022). Verifying screen reader accessibility of apps developed using google flutter. In Zallio, M., editor, *Human Factors in Accessibility and Assistive Technology*, volume 37 of *AHFE Open Access*, USA. AHFE International.

DeepSeek (2025). Deepseek — advanced ai solutions and language models. Available at: https://www.deepseek.com/. Accessed on April 15, 2025.

Delnevo, G., Andruccioli, M., e Mirri, S. (2024). On the interaction with large language models for web accessibility: Implications and challenges. In *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*, pages 1–6.

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., et al. (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*, pages 1–19.

Expo (2025). Expo — build universal native apps with react. Available at: https://expo.dev/. Accessed on April 15, 2025.

Expo Project (2025). Expo go. https://play.google.com/store/apps/details?id=host.exp.exponent&hl=pt_BR. Available at: https://

expo.dev/. Accessed on April 15, 2025.

Fagadau, I. D., Mariani, L., Micucci, D., e Riganelli, O. (2024). Analyzing prompt influence on automated method generation: An empirical study with copilot. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, ICPC '24, page 24–34. ACM.

Google (2025a). Flutter — build apps for any screen. Available at: https://flutter.dev/. Accessed on April 15, 2025.

Google (2025b). Gemini – google ai assistant. Available at: https://gemini.google.com/. Accessed on April 15, 2025.

Google LLC (2025). Accessibility scanner. Available at: https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor. Accessed on April 15, 2025.

Henry, S. L., Abou-Zahra, S., e Brewer, J. (2014). The role of accessibility in a universal web. In *Proceedings of the 11th Web for All Conference*, W4A '14, New York, NY, USA. Association for Computing Machinery.

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. In *arXiv: https://arxiv.org/abs/2308.10620*.

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.

Mascetti, S., Ducci, M., Cantù, N., Pecis, P., e Ahmetovic, D. (2021). Developing accessible mobile applications with cross-platform development frameworks. In *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, pages 1–5.

Mateus, D. A., Silva, C. A., De Oliveira, A. F., Costa, H., e Freire, A. P. (2021). A systematic mapping of accessibility problems encountered on websites and mobile apps: A comparison between automated tests, manual inspections and user evaluations. *Journal on Interactive Systems*, 12(1):145–171.

Mateus, D. A., Souza, M. R. D. A., e Freire, A. P. (2023). Accessibility of mobile apps for visually impaired users: Problems encountered by user evaluation, inspections and automated tools. In *Proceedings of the XXII Brazilian Symposium on Human Factors in Computing Systems*, pages 1–11.

Meta Platforms, Inc. (2025). React native — learn once, write anywhere. Available at: https://reactnative.dev/. Accessed on April 15, 2025.

Muniz, J. H., Mesquita Feijó Rabelo, D., e Viana, W. (2024). Assessing accessibility levels in mobile applications developed from figma templates. In *Proceedings of the 17th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '24, page 316–321, New York, NY, USA. Association for Computing Machinery.

Nunes, E. H. d. C. (2025). Design e avaliação de um guia para acessibilidade em dispositivos móveis com a abnt nbr 17060. Dissertação de mestrado, Universidade Federal do Ceará, Fortaleza, Brasil.

Nunes, E. H. D. C., Ribeiro, G. V., Monteiro, I. T., e Gonçalves, E. (2023). Digital accessibility at the brazilian symposium on human factors in computing systems (ihc): An updated systematic literature review. In *Proceedings of the XXII Brazilian Symposium on Human Factors in Computing Systems*, pages 1–15.

OpenAI (2025). Openai. Available at: `https://openai.com/`. Accessed on April 15, 2025.

Othman, A., Dhouib, A., e Nasser Al Jabor, A. (2023). Fostering websites accessibility: A case study on the use of the large language models chatgpt for automatic remediation. In *Proceedings of the 16th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '23, page 707–713, New York, NY, USA. Association for Computing Machinery.

Pereira, R., Darin, T., e Silveira, M. S. (2024). Grandihc-br: Grand research challenges in human-computer interaction in brazil for 2025-2035. In *Proceedings of the XXIII Brazilian Symposium on Human Factors in Computing Systems*, IHC '24, New York, NY, USA. Association for Computing Machinery.

Rabelo, D. M. F., de Souza Martins, R., Santos, I., da Silva, P. H. G., Gama, K., e Viana, W. (2025). Breaking barriers in mobile accessibility: A study of llm-generated native android interfaces. In *Proceedings of the 12th International Conference on Mobile Software Engineering and Systems (MOBILESoft '25)*, Ottawa, Canada. ACM.

Stack Overflow (2024). Stack overflow developer survey 2024: Most popular technologies. Available at: `https://survey.stackoverflow.co/2024/technology#most-popular-technologies-misc-tech-prof`. Accessed on April 15, 2025.

Suh, H., Tafreshipour, M., Malek, S., e Ahmed, I. (2025). Human or llm? a comparative study on accessible code generation capability. In *arXiv: `https://arxiv.org/abs/2503.15885`*.

Vendome, C., Solano, D., Liñán, S., e Linares-Vásquez, M. (2019). Can everyone use my app? an empirical study on accessibility in android apps. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 41–52.

World Wide Web Consortium (W3C) (2023). Web content accessibility guidelines (wcag) 2.2. Available at: `https://www.w3.org/TR/WCAG22/`. Accessed on April 15, 2025.

Zhang, G., Raina, A., Cagan, J., e McComb, C. (2021). A cautionary tale about the impact of ai on human design teams. *Design Studies*, 72:100990.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.-Y., e Wen, J.-R. (2025). A survey of large language models. In *arXiv: `https://arxiv.org/abs/2303.18223`*.