FEDERAL UNIVERSITY OF CEARÁ

CENTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

PROGRAM OF MASTER AND DOCTORATE IN COMPUTER SCIENCE

DANIEL MESQUITA FEIJÓ RABELO

EVALUATING ACCESSIBILITY IN NATIVE ANDROID INTERFACES GENERATED BY LARGE LANGUAGE MODELS

FORTALEZA

2025

DANIEL MESQUITA FEIJÓ RABELO

EVALUATING ACCESSIBILITY IN NATIVE ANDROID INTERFACES GENERATED BY LARGE LANGUAGE MODELS

Dissertation submitted to the Program of Master and Doctorate in Computer Science of the Center of Science of the Federal University of Ceará, as a partial requirement for obtaining the title of Master in Computer Science. Concentration Area: Software Engineering

Advisor: Prof. Dr. Windson Viana de Carvalho

FORTALEZA

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

R114e Rabelo, Daniel Mesquita Feijó.
Evaluating Accessibility in Native Android Interfaces Generated by Large Language Models / Daniel Mesquita Feijó Rabelo. – 2025.
82 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2025.
Orientação: Prof. Dr. Windson Viana de Carvalho.

1. Large language models. 2. Mobile apps. 3. Accessibility. 4. Mobile accessibility. I. Título.

CDD 005

DANIEL MESQUITA FEIJÓ RABELO

EVALUATING ACCESSIBILITY IN NATIVE ANDROID INTERFACES GENERATED BY LARGE LANGUAGE MODELS

Dissertation submitted to the Program of Master and Doctorate in Computer Science of the Center of Science of the Federal University of Ceará, as a partial requirement for obtaining the title of Master in Computer Science. Concentration Area: Software Engineering

Approved on: May 20, 2025

To my parents for always believing in me and investing in me. Mom, your care and devotion have always given me the strength to continue. Dad, your presence and love have given me the security and knowledge that I am not alone on this journey.

ACKNOWLEDGEMENTS

To my parents, my guides, who in the moments of my absence dedicated to higher education, were always by my side, giving me the support and love I needed for my studies.

I wholeheartedly thank my godmother for all the affection, encouragement, and support she has given me throughout this journey. I also extend my gratitude to her family, her husband and children, for their warmth and understanding during this entire process. Your presence was essential and made this path lighter and more meaningful.

I would like to extend my thanks to my entire family, including uncles, aunts, and cousins, for their support throughout my academic journey. I also dedicate this achievement to my grandparents for the values and teachings they passed on to me.

To my supervisor, Prof. Dr. Windson Viana de Carvalho, for always being there for me during my Master's degree and for guiding me through my thesis in the best possible way. Also, for encouraging me to join the MDCC.

To my friends who I met during my undergraduate studies at the UFC, Ribamar Souza, Matheus Gomes, and Júlia Holanda, who have been by my side ever since, always supporting me and offering help when I needed it most. To my long-time friends, Francisco Lucas and Carlos André, for being there for me even in the rush of everyday life and for the lively conversations. Your friendship was essential.

I would like to thank all the teachers for giving me not only rational knowledge but also the manifestation of the character and affection of education in the process of professional training, for how much they dedicated themselves to me, not only for teaching me, but for making me learn.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

"For people without disabilities, technology makes things easier. For people with disabilities, technology makes things possible."

(Mary Pat Radabaugh)

ABSTRACT

Recent advances in artificial intelligence, particularly in large language models (LLMs), have opened up new possibilities for automating software development tasks, including code generation for mobile applications. This study explores the capabilities of LLMs, such as ChatGPT, in improving the accessibility of native Android applications. It examines whether LLM-generated code conforms to established accessibility standards, taking into account different screen layouts, prompt formulations, and interface generation strategies. Four studies were conducted to evaluate the accessibility of seven types of mobile interfaces. The first study analyzed the accessibility of mobile application screens created using a variety of layout strategies. In contrast, the second study focused specifically on Jetpack Compose and compared the output of several LLMs. The third study examined whether creating screens with English prompts affected accessibility. Finally, the fourth study used an LLM code assistant. A total of 702 accessibility issues were identified in all studies. Jetpack Compose consistently outperformed other layout approaches, and English prompts resulted in fewer issues. Interestingly, prompts that explicitly requested accessibility often resulted in more errors, suggesting that LLMs face challenges in correctly interpreting and implementing accessibility requirements. These findings highlight the importance of refining prompt strategies and LLM outputs to reduce the risk of accessibility errors in AI-generated mobile app code.

Keywords: large language models; mobile apps; accessibility; mobile accessibility.

RESUMO

Os avanços recentes na inteligência artificial, particularmente em modelos de linguagem de grande escala (LLMs), abriram novas possibilidades para automatizar tarefas de desenvolvimento de software, incluindo a geração de código para aplicativos móveis. Este estudo explora as capacidades dos LLMs, como o ChatGPT, em melhorar a acessibilidade de aplicativos Android nativos. É examinado se o código gerado por LLMs está em conformidade com os padrões estabelecidos de acessibilidade, levando em consideração diferentes layouts de tela, formulações de prompts e estratégias de geração de interfaces. Quatro estudos foram realizados para avaliar a acessibilidade de sete tipos de interfaces móveis. O primeiro estudo analisou a acessibilidade das telas de aplicativos móveis criadas usando uma variedade de estratégias de layout. Em contraste, o segundo estudo focou especificamente no Jetpack Compose e comparou os resultados gerados por vários LLMs. O terceiro estudo examinou se a criação de telas com prompts em inglês afetava a acessibilidade. Por fim, o quarto estudo utilizou um LLM assistente de código. Um total de 702 problemas de acessibilidade foram identificados ao longo de todos os estudos. O Jetpack Compose superou consistentemente outras abordagens de layout, e os prompts em inglês resultaram em menos problemas. Curiosamente, prompts que solicitavam explicitamente a acessibilidade frequentemente resultaram em mais erros, sugerindo que os LLMs enfrentam desafios ao interpretar e implementar corretamente os requisitos de acessibilidade. Esses achados destacam a importância de refinar as estratégias de prompt e os resultados gerados pelos LLMs para reduzir o risco de erros de acessibilidade no código de aplicativos móveis gerado por IA.

Palavras-chave: large language models; aplicações móveis; acessibilidade; acessibilidade móvel.

LIST OF FIGURES

Figure 1 – Examples of elements with sufficient and insufficient contrast. 18
Figure 2 – Examples of good touch target sizes for elements. 19
Figure 3 – Example of how describe elements. 19
Figure 4 – Example of good hierarchy of elements. 20
Figure 5 – Example of Low Contrast and Touch Area Size Issues in the ANDRADE *et al.* study. 21
Figure 6 – Example of Low Contrast and Missing Labels Issues in the MARTINS *et al.* study. 22
Figure 7 – Example of Low Contrast Issues in the MUNIZ *et al.* study. 23
Figure 8 – Common Workflow for Studies 32
Figure 9 – Two Login Screens Evaluated by the Accessibility Scanner. “Nome de usuário” and “Senha” mean username and password, respectively. 35
Figure 10 – Example of a Chat-JCR prompt requesting the “Product Details”. 37
Figure 11 – Study 1 Prompt Execution Structure: Login Screen example 38
Figure 12 – NReq prompt requesting the Product Details screen in ChatGPT-3.5. The translated prompt is shown in Figure 10. 39
Figure 13 – AccessReq prompt requesting an accessible Product Details screen in ChatGPT-3.5. The translated prompt means: “Rewrite the screen code to make it accessible.”. 39
Figure 14 – Accessibility Errors by Type of Screen and Prompts in Study 1. 41
Figure 15 – Type of Accessibility errors for each type of screen in Study 1. 42
Figure 16 – Accessibility errors across layouts for each type of screens in each type of layout, in Study 1. 42
Figure 17 – Boxplot showing the number of accessibility errors across all trials, for each layout investigated, in Study 1. 43
Figure 18 – Accessibility errors in each LLM, for each type of prompt, in Study 2. . . . 47
Figure 19 – Accessibility errors in each screen, for each type of prompt, in Study 2. . . . 47
Figure 20 – Type of Accessibility errors for each type of screen, in Study 2. 49
Figure 21 – Accessibility errors across LLMs for each type of screens, in Study 2. 49
Figure 22 – Boxplot showing the number of accessibility errors across all 42 trials, for each of the LLMs investigated, in Study 2 50

Figure 23 – Accessibility Errors by Type of Screen and Prompts in Study 3. 53
Figure 24 – Type of Accessibility errors for each type of screen, in Study 3. 53
Figure 25 – Accessibility errors across prompts for each type of screens, in Study 3. . . 54
Figure 26 – Accessibility Errors by Type of Screen and Prompts in Study 4. 57
Figure 27 – Type of Accessibility errors for each type of screen in each LLM, in Study 4. 57
Figure 28 – Accessibility errors across prompts for each type of screens, in Study 4. . . 58
Figure 29 – Accessibility Errors by Type of Screen for each Study. 60
Figure 30 – Mean Accessibility Errors by Type of Screen for each Study. 60
Figure 31 – Boxplot showing the number of accessibility errors across all trials, for each study. 61
Figure 32 – Accessibility Errors by Type of Request for each Study. 62
Figure 33 – Mean Accessibility Errors by Type of Request for each Study. 62
Figure 34 – Accessibility Errors by ChatGPT-4o. 63
Figure 35 – Type of Accessibility errors for each Study. 63

LIST OF TABLES

Table 1 – Summary of Related Work 28
Table 2 – Main differences between Studies. 31
Table 3 – List of requested types of screen and their descriptions 33
Table 4 – Types of errors found by Accessibility Scanner and their descriptions. 34
Table 5 – Errors by type of prompt from Study 1. 40
Table 6 – Accessibility errors for each type of screen, categorized by layout approach, in Study 1. 41
Table 7 – Errors by type of prompt from Study 2. 46
Table 8 – Accessibility Errors by Type of Prompt for Each Accessibility Error Category in Study 2. 48
Table 9 – Errors for each type of prompt, for each type of screen, in Study 2. 48
Table 10 – Errors by type of prompt from Study 3. 52
Table 11 – Errors by type of prompt from Study 4. 56
Table 12 – Errors by type of prompt from all studies. 59

LIST OF ABBREVIATIONS AND ACRONYMS

ADA	Americans with Disabilities Act
AI	Artificial Intelligence
LLMs	Large Language Models
PVI	People with Visual Impairments
PwDs	Persons with Disabilities
UI	User Interface
WCAG	Web Content Accessibility Guidelines

CONTENTS

1 INTRODUCTION 14
1.1 Main Goal 15
1.2 Research Questions 15
1.3 Master Thesis Organization 16
2 BACKGROUND 17
2.1 Mobile Accessibility 17
2.1.1 Mobile Accessibility Principles 17
2.1.2 Mobile Accessibility Studies 21
2.2 Large Language Models (LLMs) 23
2.2.1 LLMs in Software Development 24
2.2.2 Prompt Engineering 25
2.3 Related Work 27
2.4 Chapter Synthesis 28
3 METHODOLOGY 29
3.1 Research Goals 29
3.2 Screen selection 31
3.3 Materials and Methods 32
3.3.1 App Generation 32
3.3.2 Accessibility Test 33
3.4 Chapter Synthesis 35
4 STUDY 1 - LAYOUT TYPE 36
4.1 LLM selection 36
4.2 Prompt Design 36
4.3 Prompt Execution 38
4.4 Results 40
4.5 Chapter Synthesis 43
5 STUDY 2 AND 3 - LLM AND PROMPT LANGUANGE VARIATION . . 45
5.1 Study 2 45
5.1.1 LLMs selection 45
5.1.2 Prompt Design and Execution 46
5.1.3 Results 46

5.2 **Study 3** 51
5.2.1 *LLM selection* 51
5.2.2 *Prompt Design and Execution* 51
5.2.3 *Results* 52
5.3 **Chapter Synthesis** 54
6 **STUDY 4 - LLM FOR CODE GENERATION** 55
6.1 **LLM selection** 55
6.2 **Prompt Design and Execution** 55
6.3 **Results** 56
6.4 **Chapter Synthesis** 58
7 **DISCUSSION** 59
7.1 **Summary of the Studies** 59
7.2 **RQ1 - What is the accessibility level of the screens generated by LLMs for native Android?** 64
7.3 **RQ2 - What is the accessibility level of the screens generated by LLMs for native Android, when accessibility is required in the prompt?** 65
7.4 **RQ3 - Are the elements of the codes generated by the LLMs in the requested language?** 66
7.5 **RQ4 - Is there a variation in accessibility levels when repeating the same prompt?** 66
7.6 **Implications and takeaways** 67
7.7 **Threats to Validity** 68
7.7.1 *Construct Validity* 68
7.7.2 *External Validity* 68
7.7.3 *Internal Validity* 69
7.8 **Chapter Synthesis** 69
8 **CONCLUSION** 71
8.1 **Final Considerations** 71
8.2 **Main Contributions** 72
8.3 **Publications and datasets** 73
8.4 **Future work** 74
REFERENCES 75

1 INTRODUCTION

The rapid evolution of Artificial Intelligence (AI) has driven significant advances in the development of Large Language Models (LLMs), such as the widely known GPT (OPENAI, 2024). LLMs are neural networks trained to understand, generate, and manipulate natural language in a wide range of tasks, including translation, summarization, question answering, and content creation. These models are characterized by their substantial size and their ability to detect intricate linguistic patterns from large amounts of text data.

LLMs have shown an extraordinary ability to produce human-like text, enabling them to assist in numerous tasks, from content generation and customer service to software development (HOU *et al.*, 2024; ZHANG *et al.*, 2020; CHEN *et al.*, 2021). One particularly promising application of LLMs is code generation, which has the potential to transform how software applications (e.g., mobile, web) are developed (SIDDIQ *et al.*, 2022; SVYATKOVSKIY *et al.*, 2020; LIU *et al.*, 2024; LI *et al.*, 2024).

Recent studies indicate that the software development landscape is undergoing a notable transformation, driven by the increasing adoption of LLM-powered code assistant tools (e.g., GitHub Copilot). These tools and the LLMs themselves are now widely used by the majority of developers (MOWAR *et al.*, 2025). They have proven valuable in automating specific coding tasks, allowing developers to focus on more complex and creative work while substantially reducing development time and potentially minimizing human effort (GAO *et al.*, 2025; MOWAR *et al.*, 2025; PENG *et al.*, 2023; GOTTLANDER; KHADEMI, 2023).

However, despite these promising capabilities, the use of LLMs in generating code raises important questions about the quality of the code (e.g., correctness, readability, and maintainability) (DILLMANN *et al.*, 2024; FAN *et al.*, 2023; SIDDIQ *et al.*, 2022; SPIESS *et al.*, 2024), as well as the accessibility of the resulting software (ALJEDAANI *et al.*, 2024; SUH *et al.*, 2025).

In this context, accessibility refers to the ability of software to be used by all individuals, including those with disabilities such as visual, auditory, or motor impairments. The importance of accessibility in software development is underscored by ethical considerations and legal obligations, including those outlined in Americans with Disabilities Act (ADA) (U.S. Congress, 1990) and Web Content Accessibility Guidelines (WCAG) (W3C, 2023).

Given the growing reliance on LLMs, it is critical to examine whether mobile applications developed using LLM-generated code comply with current accessibility standards.

Previous studies have shown that mobile accessibility is often overlooked during the development process, creating barriers for users with disabilities (ZAINA *et al.*, 2022; SERRA *et al.*, 2015; PEREIRA *et al.*, 2024; OLIVEIRA *et al.*, 2023; ALSHAYBAN *et al.*, 2020; LEITE *et al.*, 2021). Moreover, there is a lack of detailed and specific feedback from Persons with Disabilities (PwDs) (OLIVEIRA *et al.*, 2023). Moreover, LLMs may not inherently prioritize accessibility unless explicitly instructed to do so, potentially exacerbating existing issues in inclusive software.

In addition, today's LLMs have been observed to have language bias and often exhibit suboptimal performance in code generation tasks when instructed in languages other than English (WANG *et al.*, 2024a; KOYANAGI *et al.*, 2024). In addition, these models often face challenges in capturing equivalent semantic meanings when natural language prompts with the same intent are expressed in different languages (PENG *et al.*, 2024; HELLAS *et al.*, 2023; JORDAN *et al.*, 2024).

1.1 Main Goal

To the best of our knowledge, no previous study has evaluated the accessibility of LLM-generated mobile screens, a crucial aspect of mobile app development. Although recent literature has begun to address accessibility in LLM-generated code for web applications (AL-JEDAANI *et al.*, 2024), the mobile domain remains unexplored. Given the increasing use of LLMs to support mobile application development, it is essential to assess whether the content generated by these models adheres to accessibility standards for users with disabilities.

This research aims to fill this gap by systematically analyzing the accessibility of native Android interfaces generated by LLMs. Throughout this study, the term *interface* refers specifically to the User Interface (UI), the graphical and interactive components with which users engage. It is important to clarify that the scope of this study is limited to evaluating the responses generated by LLMs, and does not extend to other aspects of mobile application development such as the underlying framework or hardware compatibility.

1.2 Research Questions

To achieve the objective of this research, four research questions were formulated. These questions aim to explore key aspects of the accessibility of LLM-generated mobile screens, focusing on both the general accessibility level and the specific effects of various parameters, such as prompt formulation and consistency of results. The questions are designed to assess how well LLM-generated code meets accessibility standards and to explore potential improvements for more inclusive development practices.

- **RQ1** - What is the accessibility level of the screens generated by LLMs for native Android?
- **RQ2** - What is the accessibility level of the screens generated by LLMs for native Android, when accessibility is required in the prompt?
- **RQ3** - Are the elements of the codes generated by the LLMs in the requested language?
- **RQ4** - Is there a variation in accessibility levels when repeating the same prompt?

1.3 Master Thesis Organization

To address these research questions, four studies were conducted to evaluate the accessibility of seven different types of mobile interface. The first study assessed how various layout strategies affect the accessibility of mobile app screens. The second study focused on Jetpack Compose, comparing the screen output of multiple LLMs. The third study examined how English-language prompts influence the accessibility of generated interfaces. Finally, the fourth study explored the effectiveness of an LLM-powered code assistant in generating accessible mobile screens.

The remainder of the thesis is structured as follows: Chapter 2 provides detailed information on research focusing on the accessibility of mobile applications, LLMs, and some related work. The subsequent chapters, Chapter 3, detail the methodological framework and research questions. Subsequently, Chapters 4, 5, and 6 detail the steps and methods used to evaluate the accessibility of LLM-generated screens. Finally, Chapters 7 and 8 present a discussion of the research questions involved in the investigation and the final considerations regarding the implications of these findings.

2 BACKGROUND

This chapter presents the background relevant to this research. It is divided into four sections. First, Section 2.1 discusses the topic of mobile accessibility, focusing on design principles, common challenges, and current studies. Then, Section 2.2 explores the role of LLMs in software development, with emphasis on their capabilities and limitations. Section 2.3 examines existing literature related to the intersection of LLMs and accessibility, especially in the context of code generation. Finally, Section 2.4 presents a brief conclusion of this chapter.

2.1 Mobile Accessibility

According to the World Health Organization (WHO)[1], an estimated 1.3 billion people, about 16% of the global population, currently experience significant disability. As highlighted in its latest report on vision (WHO *et al.*, 2019), it is estimated that more than 2.2 billion people have some form of visual impairment.

Mobile accessibility for People with Visual Impairments (PVI) is based on assistive technologies built into smartphones, such as Android's TalkBack[2], which provides auditory feedback for elements of the user interface. With TalkBack enabled, interface elements are outlined with a focus box, and the device narrates their content. Instead of typical touch interactions, users rely on gestures to navigate. Modern tools also provide additional features such as font customization, color filters, and screen magnification.

To ensure these accessibility tools function properly, applications need to follow specific standards and provide alternative content, such as proper markup, with descriptions that match the context and accessibility needs (CHEN *et al.*, 2022).

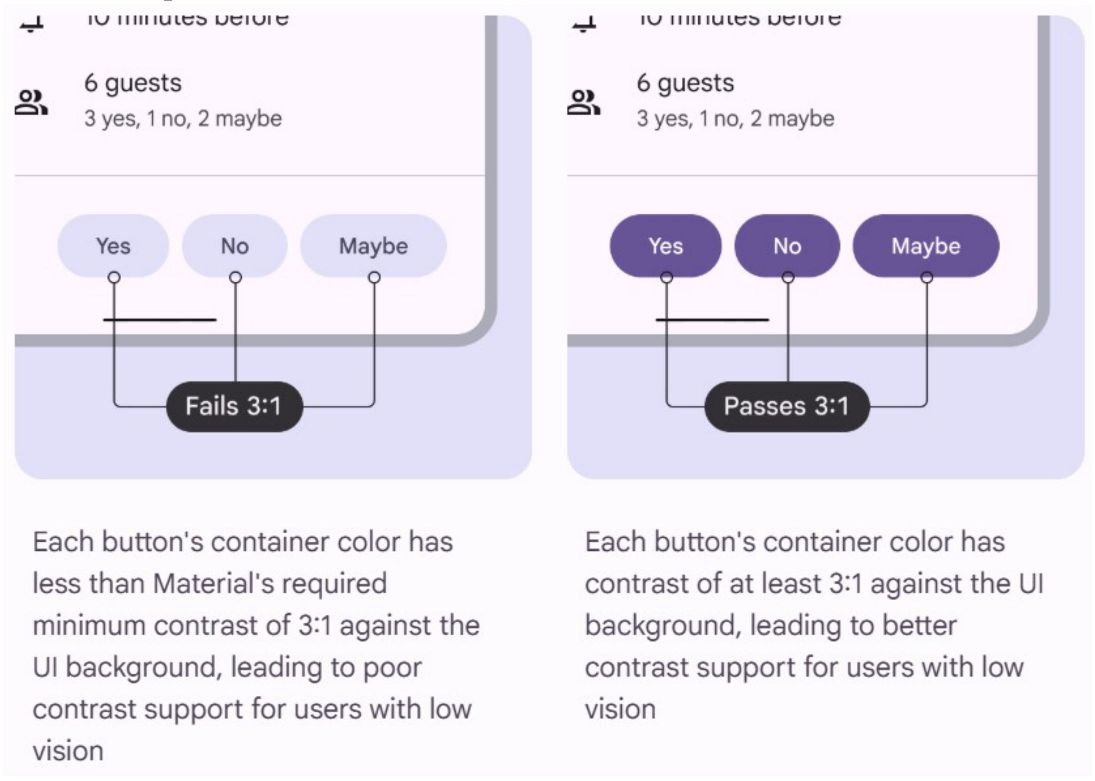
2.1.1 Mobile Accessibility Principles

To assist developers and designers of mobile apps, accessibility guidelines have been created in both industry (e.g., Android Accessibility Guidelines[3]) and academia (BALLANTYNE *et al.*, 2018). These are primarily inspired by the WCAG, covering a range of accessibility aspects, including UI design, control visibility, alternative text for images and other UI elements, as well as navigation and interaction, aiming to ensure accessible apps for everyone.

[1] https://www.who.int/health-topics/disability
[2] https://support.google.com/accessibility/android/answer/6283677?hl=pt-br
[3] https://developer.android.com/guide/topics/ui/accessibility

In addition to the Android Accessibility Guidelines, the Material Design 3[4] website also mentions the main accessibility issues and how to work around them. One of the most common is low contrast, which is the difference in brightness between the color of the text and the color of the background behind the text. Contrast ratios represent how different one color is from another color; the greater the difference between the two numbers, the greater the difference in relative luminance between the colors. In general, elements should have a contrast ratio of 3:1, and elements that have low contrast are a hindrance to PVIs using the applications. Figure 1 shows two examples of sufficient and insufficient contrast.

Figure 1 – Examples of elements with sufficient and insufficient contrast.



Each button's container color has less than Material's required minimum contrast of 3:1 against the UI background, leading to poor contrast support for users with low vision

Each button's container color has contrast of at least 3:1 against the UI background, leading to better contrast support for users with low vision

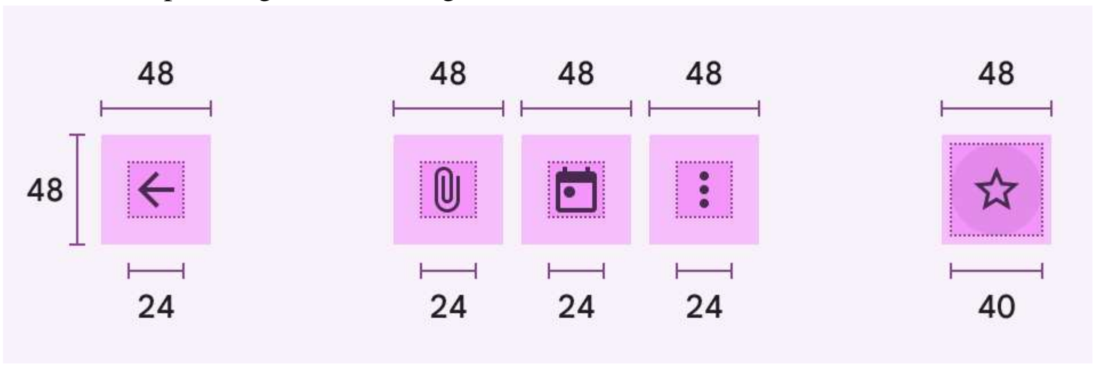
Source: https://m3.material.io/foundations/designing/color-contrast

Touch target size is an issue that affects users who cannot see the screen or who have difficulty tapping small touches on items in your application. For example, an icon may appear to be 24 x 24dp, but the padding surrounding it includes the full 48 x 48dp touch target. In general, it is recommended that the touch targets be at least 48 x 48dp. Figure 2 shows an example of how to make an icon with good touch target sizes.

Another very common problem is the absence or inadequacy of descriptive labels on screen elements. Accessibility labels assist users who cannot rely on a product's visual interface

[4] https://m3.material.io/foundations/overview/principles

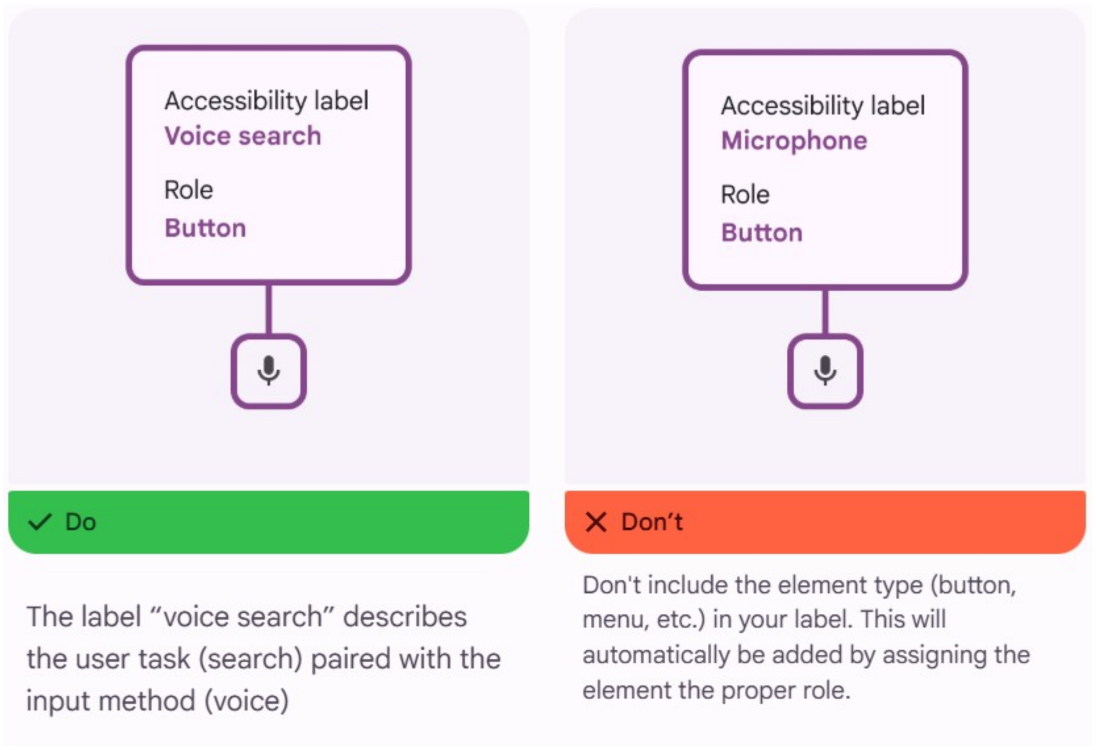
Figure 2 – Examples of good touch target sizes for elements.



Source: https://m3.material.io/foundations/designing/structure

and should concisely describe an element's content, purpose, and behavior. Figure 3 shows how the elements on the screen can be correctly described.

Figure 3 – Example of how describe elements.



The label "voice search" describes the user task (search) paired with the input method (voice)

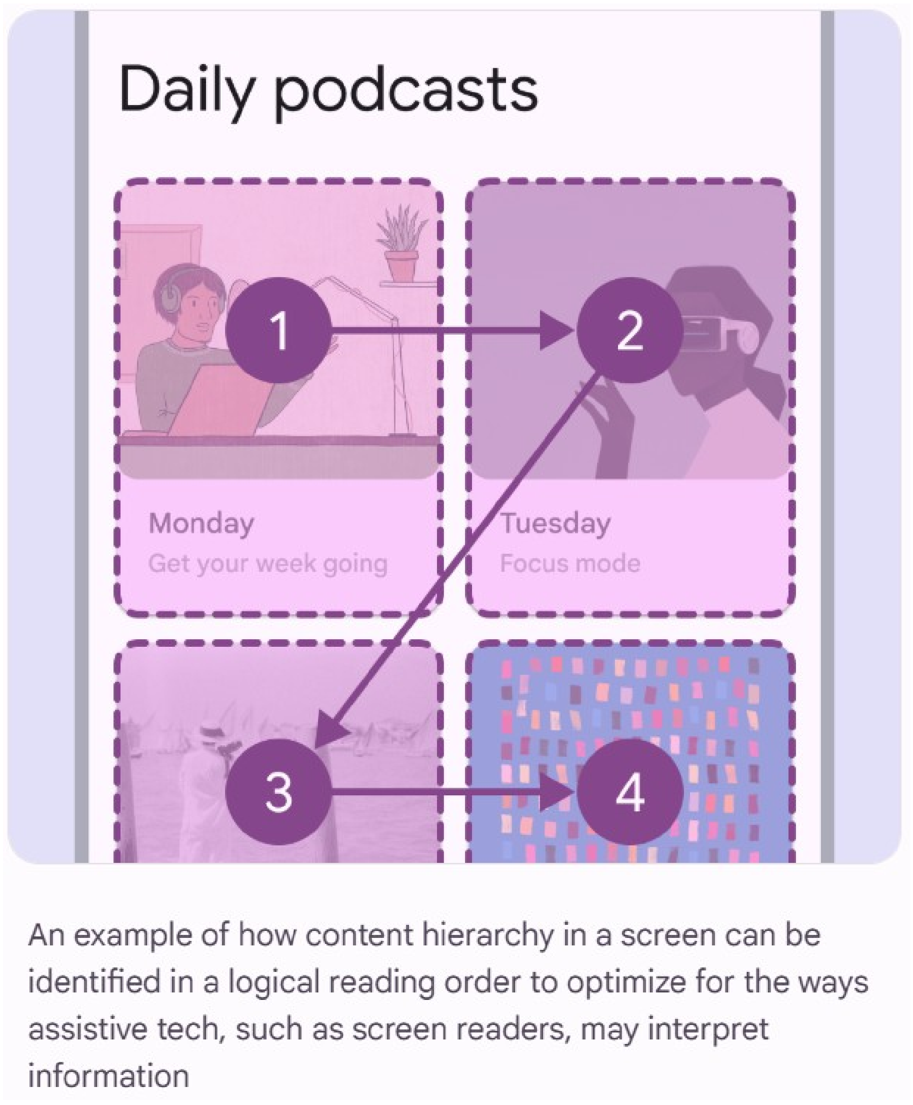
Don't include the element type (button, menu, etc.) in your label. This will automatically be added by assigning the element the proper role.

Source: https://m3.material.io/foundations/designing/elements

The hierarchy of information is fundamental for efficient navigation in an application. When navigation is clear, users can easily understand where they are and what is important. In order for the screen reader to read the content in the desired order, it is important that the elements on the screen also have a hierarchy. Figure 4 shows what a good hierarchy of screen elements looks like.

There are other accessibility issues, such as the lack of alternative text for images,

Figure 4 – Example of good hierarchy of elements.



An example of how content hierarchy in a screen can be identified in a logical reading order to optimize for the ways assistive tech, such as screen readers, may interpret information

Source: https://m3.material.io/foundations/designing/structure

and decorative icons and images that do not improve the experience for a PVI should be marked as decorative to hide them in the code. Therefore, text should be wrapped when it is critical to ensure comprehension or when there is space in the component. If long strings of enlarged text will not fit on a screen, consider adding a vertical scrollbar to provide access to more content.

In the current market, there are already free tools available to evaluate the accessibility of mobile applications. These include Accessibility Scanner[5] for Android and Xcode Accessibility Inspector[6] for iOS. In addition to these, the authors have developed proprietary tools to access accessibility, and all of such tools have been successful in identifying errors (ALSHAYBAN *et al.*, 2020; CHEN *et al.*, 2022; ELER *et al.*, 2018; SWEARNGIN *et al.*, 2024).

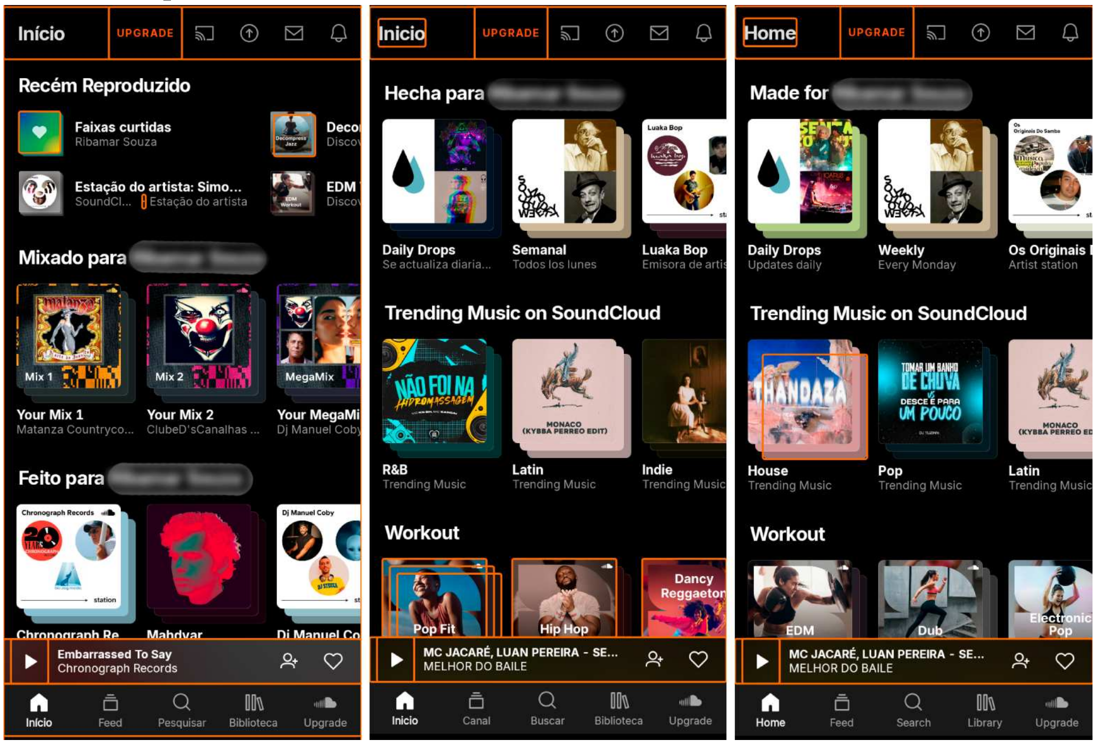
5 https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor
6 https://developer.apple.com/accessibility/

2.1.2 Mobile Accessibility Studies

Despite these efforts, unfortunately, the accessibility reality in mobile applications remains unsatisfactory (VENDOME *et al.*, 2019; ALSHAYBAN *et al.*, 2020; CHEN *et al.*, 2022; LEITE *et al.*, 2021). Accessibility errors are prevalent across various categories of apps, from Internet of Things (IoT) applications (e.g., Amazon Alexa) (MARTINS *et al.*, 2022; TAZI *et al.*, 2023), banking apps (LOPES *et al.*, 2022; KAMESWARAN; MURALIDHAR, 2019), government apps (SERRA *et al.*, 2015), and even in Figma's templates designed for mobile apps (MUNIZ *et al.*, 2024).

Obviously, this also happens with popular apps, and a study by (ANDRADE *et al.*, 2024) evaluated eight popular Android apps, through different tasks and in three different languages: Portuguese, English, and Spanish. The result showed a total of 2,355 accessibility errors, with the Spanish tests showing the most errors (828). Figure 5 shows an example of low-contrast and touch area size issues found in the SoundCloud app's home screen.

Figure 5 – Example of Low Contrast and Touch Area Size Issues in the ANDRADE *et al.* study.

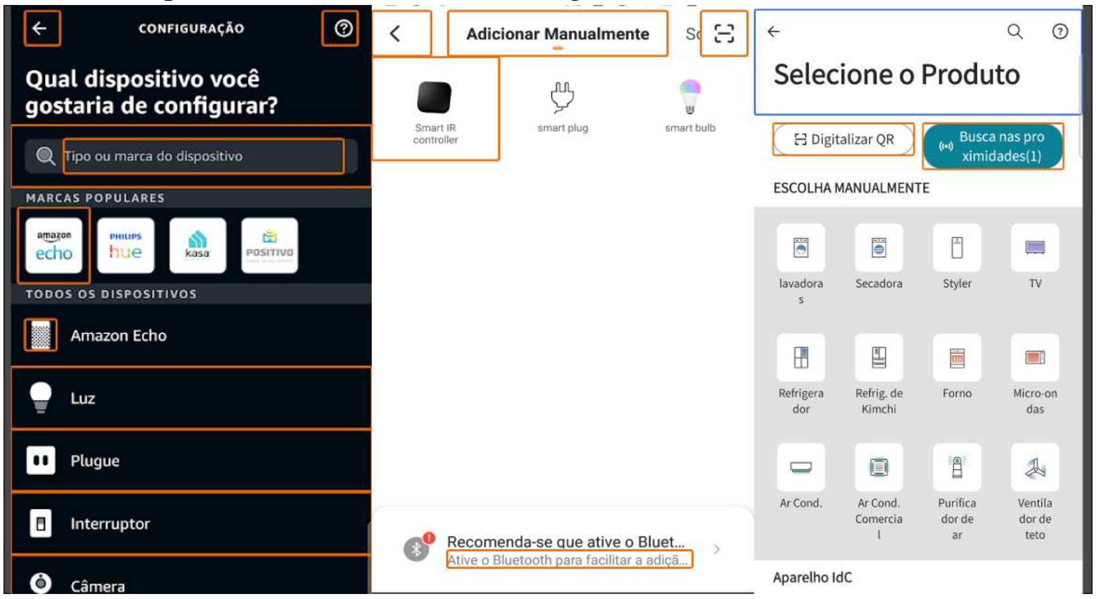


Source: (ANDRADE *et al.*, 2024)

These issues even occur in open source apps, VENDOME *et al.* investigated a set of 13,000 Android apps with code on GitHub. They showed that 46.25% of them had at least 50% visual elements without descriptions. 5,107 had all content descriptions empty.

Other authors manually evaluated the accessibility of 248 IoT apps from the Play Store (TAZI *et al.*, 2023). They found that only two apps had no accessibility errors and more than 80% of the apps had errors related to missing names and descriptions of items and/or images. Other studies with IoT apps (MARTINS *et al.*, 2022; OLIVEIRA *et al.*, 2016) show similar results. Figure 6 shows low contrast and missing label issues in the Amazon Alexa, LG ThinQ, and RSmart apps.

Figure 6 – Example of Low Contrast and Missing Labels Issues in the MARTINS *et al.* study.



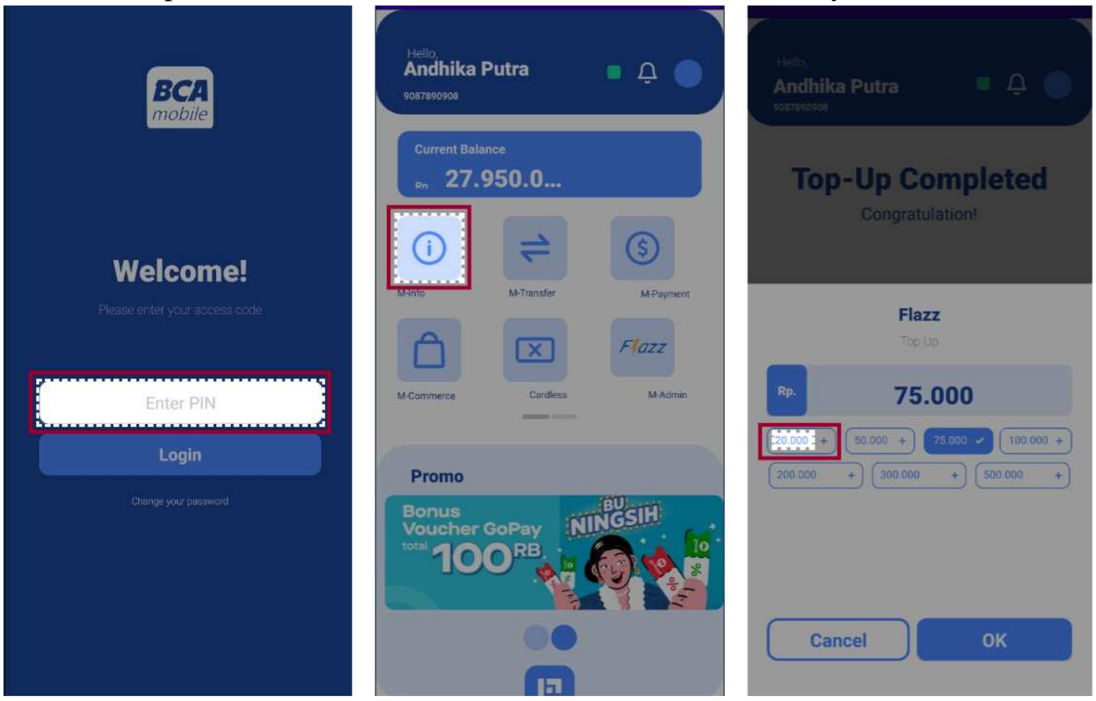
Source: (MARTINS *et al.*, 2022)

In a different approach, MUNIZ *et al.* built applications from ten widely used Figma templates and evaluated their accessibility. They discovered 738 accessibility issues, with an average of 7.61 errors per screen, and all templates had problems. Figure 7 shows an example of low-contrast issues identified in one of the templates.

Studies (ALSHAYBAN *et al.*, 2020; LEITE *et al.*, 2021; SANTOS; BRANCO, 2024), following surveys with mobile developers, attribute these accessibility issues to the limited knowledge of guidelines and the infrequent consideration of accessibility in app projects. Developers also pointed out the inappropriate use of development tools that often overlook accessibility. In addition, the application templates provided by Android Studio itself have accessibility issues, some of which are severe (ALSHAYBAN *et al.*, 2020).

Since the codebase of existing mobile applications available on the Internet tends to have accessibility issues, one of the hypotheses of our research is that these errors are reproduced by generative models trained with this codebase.

Figure 7 – Example of Low Contrast Issues in the MUNIZ *et al.* study.



Source: (MUNIZ *et al.*, 2024)

2.2 Large Language Models (LLMs)

Large Language Models (LLMs) are advanced AI systems trained on vast text datasets to understand and generate human language. These models form the foundation for a wide range of tools that enable advanced language processing tasks, such as text and image generation (BROWN *et al.*, 2020). When users interact with LLM, their input is tokenized, analyzed, and processed by the model, which predicts the most likely response based on the patterns learned (VASWANI *et al.*, 2023). This process allows LLMs to generate coherent and contextually relevant output on a broad spectrum of topics, including mobile code and screen descriptions.

Currently, the most well-known LLMs include GPT-3.5 (OPENAI, 2020) and GPT-4o (OPENAI, 2023), both developed by OpenAI. GPT-3.5, with 175 billion parameters, is recognized for its ability to generate and comprehend text. Meanwhile, GPT-4o promises to provide more accurate and coherent responses, with improved understanding of complex contexts. Following the success of ChatGPT, other companies have started releasing and promoting their own LLM. For example, Google has developed Gemini[7], which aims to enhance human-machine interaction by leveraging Google's infrastructure. Another example is Llama[8], from Meta, an

7 https://gemini.google.com
8 https://llama.meta.com

open source LLM.

Furthermore, the Brazilian company Maritaca AI has developed the Sabiá model (MARITACAAI, 2024), which is specifically trained to understand and generate text in Portuguese with high precision (PIRES *et al.*, 2023). This reflects significant progress in adapting LLMs to specific languages, contributing to improved accessibility and effectiveness of AI in regional linguistic contexts.

2.2.1 LLMs in Software Development

LLMs have an immense potential to help in software development (LI *et al.*, 2024). These models can automate various repetitive tasks, such as code generation, documentation, and testing processes (CHEN *et al.*, 2021; WANG *et al.*, 2024b; TIAN *et al.*, 2023; LI *et al.*, 2022). With their ability to understand natural language, LLMs enable developers to interact more intuitively with tools, increasing productivity, and facilitating the exploration of new ideas and creative approaches in software development.

Peng *et al.* (2023) conducted a trial in which programmers were assigned with implementing an HTTP server in JavaScript. The treated group had access to GitHub Copilot and a brief instructional video, while the control group had to rely on Internet search and Stack Overflow. The results indicated that the treated group completed task 55.8% faster, with a confidence interval of 21-89%. Performance gains were particularly pronounced among less experienced developers, older programmers, and those who programmed more hours per day.

In a similar vein, a study by Mowar *et al.* (2025) evaluated a custom GitHub Copilot extension that suggested accessibility-compliant code. The controlled study with 20 novice developers showed the extension's effectiveness in reinforcing accessibility practices, guiding developers through the process.

However, research has also identified some drawbacks to using LLMs in software development (ZHONG; WANG, 2024; WANG *et al.*, 2024c; TANG *et al.*, 2024; GAO *et al.*, 2025). For example, Zhong and Wang (2024) demonstrated that LLMs' answers to real-world Java coding questions frequently suffer from API misuse issues. Furthermore, Gao *et al.* (2025) discusses several challenges within the new software development paradigm that relies on LLM-based assistance, particularly in areas such as requirements, design, coding assistance, and code review.

In addition, Wang *et al.* (2024c) conducted a user study with 56 participants, who were randomly assigned to two groups to perform two development tasks. The first task involved solving two simple coding puzzles, while the second required fixing two real-world bugs from small-scale open-source projects. In Task 1, LLMs showed competence in handling well-defined and straightforward problems. However, in Task 2, they exhibited notable deficiencies in understanding complex tasks and multi-turn interactions.

Furthermore, current LLMs often face difficulties in generating source code based on instructions in languages other than English. They struggle with capturing equivalent semantics across different languages, especially when the prompts have the same intent (WANG *et al.*, 2024a; KOYANAGI *et al.*, 2024; PENG *et al.*, 2024; HELLAS *et al.*, 2023; JORDAN *et al.*, 2024).

Although there are many efforts to develop LLMs tuned for specific languages such as Arabic (AL-AZANI *et al.*, 2024) (AraBERT and MARBERT), Chinese (HU *et al.*, 2024) (Ernie and Ziya), French (BRETON *et al.*, 2024) (CamemBERT), and Portuguese (PIRES *et al.*, 2023) (Sabiá-3). Few studies have evaluated the effectiveness of prompt source code generation in languages other than English, especially when focusing on the development of mobile applications.

Despite these challenges, the potential for LLMs to transform software development remains strong. As advancements continue to address limitations in multilingual code generation and task complexity, LLMs hold the promise of making the development process more efficient, creative, and accessible, bridging language gaps in the global software ecosystem.

2.2.2 Prompt Engineering

Prompt engineering is the strategic design of natural language instructions to guide LLMs to produce accurate, relevant, and safe output (SAHOO *et al.*, 2025; TONMOY *et al.*, 2024; CHEN *et al.*, 2025). Well-designed prompts allow pre-trained models to be applied to a wide range of tasks without the need for fine-tuning, making them highly adaptable and efficient (SAHOO *et al.*, 2025).

In addition, prompt engineering plays a central role in improving model safety and fairness. Optimized prompting techniques can help mitigate bias and improve the generalizability of LLMs in different domains (XU *et al.*, 2024). Furthermore, prompt formulation significantly influences task success and that understanding model behavior through prompting is critical for responsible AI development (REYNOLDS; MCDONELL, 2021).

Several techniques have been developed to enhance the effectiveness of prompts when interacting with LLMs:

- **Zero-Shot Prompt:** In this approach, the model is asked to perform a task without giving any prior examples. Its performance relies entirely on knowledge acquired during pre-training. For instance, requesting "Translate to French: 'Good morning'" without examples is a zero-shot prompt (BROWN *et al.*, 2020).
- **Few-Shot Prompt:** This technique involves including a few input-output examples within the prompt to help the model learn the task format. It is particularly useful in domain-specific tasks such as translation or text classification (BROWN *et al.*, 2020).
- **Chain-of-Thought (CoT):** A prompting technique that encourages the model to generate a step-by-step reasoning path to solve complex problems, improving performance on logical and multi-step tasks (WEI *et al.*, 2023).
- **In-Context Prompt:** Refers to the model's ability to temporarily learn from the examples provided within the prompt itself, without requiring any retraining. This concept underpins zero-shot and few-shot prompting (BROWN *et al.*, 2020).
- **Prompt Injection:** A method in which manipulated or adversarial inputs are inserted into a prompt to alter the output of the model. While it can be used for personalization, it also raises concerns about the security and reliability of the model (PEREZ; RIBEIRO, 2022).

Prompt engineering has been widely applied in various sectors. In content creation, personalized prompts make it possible to control the style, tone, and structure of texts, facilitating the production of materials that are aligned with specific needs (OPPENLAENDER *et al.*, 2024). In addition, prompt engineering is increasingly important for AI security. Robust prompt design is critical for maintaining the integrity of AI systems in real-world deployments (PEREZ; RIBEIRO, 2022).

Despite its potential, prompt engineering faces challenges, such as the need for a deep understanding of the model and the constant evolution of techniques. However, as research continues to advance and specialized tools are developed, prompt engineering is expected to become an increasingly accessible and essential skill for professionals working with generative AI.

2.3 Related Work

To the best of our knowledge, no research has evaluated the accessibility of LLM-generated mobile app interfaces. However, it should be noted that a significant number of studies are published daily on the use of LLMs for code generation and developer support, as seen in work such as (ZHONG; WANG, 2024; TANG *et al.*, 2024; WANG *et al.*, 2024c), including in the context of Android development (VASILINIUC; GROZA, 2023). In general, these studies indicate that LLMs have the potential to assist in software development, but the code generated by these models cannot be fully trusted.

In the specific context of accessibility, some studies (ALJEDAANI *et al.*, 2024; OTHMAN *et al.*, 2023; DELNEVO *et al.*, 2024) have investigated the topic in relation to the use of LLMs, although limited to the accessibility of Web applications. For example, Aljedaani *et al.* (2024) evaluated the accessibility of the code of ChatGPT-generated Web applications with the assistance of 88 Web developers. Most of the websites (84%) exhibited significant accessibility issues, with more than 700 issues identified using two accessibility checkers.

Suh *et al.* (2025) present an empirical study that compares the accessibility of web code generated by GPT-4o and Qwen2.5-Coder-32B-Instruct AWQ against human-written code. The results of this study indicate that LLMs frequently generate code that is more accessible, particularly in basic features such as color contrast and alternative text. However, LLMs face challenges on complex issues, including ARIA attributes. The study also examined the impact of prompting techniques (e.g. Zero-Shot, Few-Shot, and Self-Criticism) on accessibility, finding that prompts consistently yielded code with reduced accessibility issues.

Another study, by Othman *et al.* (2023), used ChatGPT to identify and correct accessibility issues on two selected websites, one in English and the other in Arabic. Of the 39 identified errors, the LLM was able to correct 37, achieving a success rate of 94%. Furthermore, the results showed that the LLM can automatically resolve a substantial number of issues, reducing manual effort.

Finally, Delnevo *et al.* (2024) also used ChatGPT to identify accessibility issues in HTML snippets. The elements evaluated included HTML forms, tables, and images. Similarly to the previous study (OTHMAN *et al.*, 2023), the results demonstrated that LLM can automatically fix a significant number of issues but requires human oversight.

The last two studies (OTHMAN *et al.*, 2023; DELNEVO *et al.*, 2024) focus on the ability of LLMs to assess and solve accessibility issues of Web apps. In contrast, another study

(ALJEDAANI *et al.*, 2024; SUH *et al.*, 2025) specifically evaluates the accessibility of the code generated by LLM. Our research shares a similar objective, but with a focus on exploring the accessibility of code generated by LLMs to create mobile interfaces on Android devices.

2.4 Chapter Synthesis

This chapter has provided an overview of mobile accessibility, the role of LLMs in software development, and related research on accessibility in AI-generated code. While LLMs demonstrate potential in automating and improving coding tasks, especially for web platforms, their capabilities in mobile accessibility remain largely unexplored.

Table 1 summarizes key studies that have assessed the accessibility of LLM-generated Web applications. These findings underscore both the promise and the current limitations of generative models in producing accessible interfaces.

Table 1 – Summary of Related Work

	Research	**Result**
Aljedaani et al.	Evaluated the accessibility of Web app code generated by ChatGPT.	84% of the websites exhibited accessibility issues, with more than 700 issues identified.
Delnovo et al.	Utilized ChatGPT to identify accessibility issues in HTML snippets.	LLM can automatically fix a significant number of issues, but requires human oversight.
Othman et al.	Utilized ChatGPT to identify and correct accessibility issues in two selected websites.	LLM was able to correct 37 of 39 issues, achieving a success rate of 94%.
Suh et al.	Compared the accessibility of web code generated by GPT-4o and Qwen2.5-Coder-32B-Instruct AWQ against human-written code.	LLMs frequently generate code that is more accessible in basic features, however encounter challenges in complex issues.

Source: Prepared by the author.

Building upon this background, our research extends these efforts by focusing on mobile applications. Specifically, we investigate the accessibility of LLM-generated Android interfaces.

3 METHODOLOGY

This chapter outlines the methodological approach used to investigate whether the LLM-generated code produces accessible mobile user interfaces. We begin by defining the research objectives and questions in Section 3.1. Section 3.2 details the process of selecting representative screen types. Section 3.3 describes the procedures for generating applications, performing accessibility evaluations and analyzing the results. We conclude the chapter with a summary of the strengths and scope of the methodology in Section 3.4.

3.1 Research Goals

The objective of this research was to analyze whether the code proposed by LLM for different types of mobile application screens ensures accessibility. The core of the study involves simulating a developer requesting a screen example from an LLM and checking the suggested code for accessibility issues. The idea behind this research is similar to other studies (ALJEDAANI *et al.*, 2024; WANG *et al.*, 2024c; TANG *et al.*, 2024) that assess the quality of LLM code outputs.

To guide the research, we defined four research questions:

- **RQ1** - What is the accessibility level of the screens generated by LLMs for native Android?
  Rationale: This question evaluates the accessibility level of the screens generated by LLMs for native Android. The answer to this question may indicate whether artificial intelligence tools generate interfaces that meet accessibility requirements and whether improvements are needed to ensure more inclusive access. The results could imply changes in the design of AI systems to prioritize accessibility in their outputs.
- **RQ2** - What is the accessibility level of the screens generated by LLMs for native Android, when accessibility is required in the prompt?
  Rationale: This question investigates whether including an explicit request for accessibility in the prompt leads to improvements in the accessibility level of the generated screens. The results may provide insight into the effectiveness of the instructions given to the model and its ability to generate accessible screens under specific requirements, contributing to the enhancement of user-AI interaction in accessibility contexts.
- **RQ3** - Are the elements of the codes generated by the LLMs in the requested language?
  Rationale: This question investigates whether the code elements generated by LLMs are

in accordance with the requested language or if there are interpretation errors that may compromise the quality or functionality of the screen. Understanding this dynamic is important to validate the ability of LLMs to correctly translate user requirements into appropriate code and can reveal limitations of the tools when it comes to generating code in specific languages.

- **RQ4** - Is there a variation in accessibility levels when repeating the same prompt?
  Rationale: Consistency in output generation is a desirable feature in AI systems. This question examines whether there are variations in the accessibility levels of the generated screens when repeating the same prompt. If the outputs are inconsistent, this may indicate flaws in the AI models or algorithms used, compromising the reliability and quality of the automated development process. The answer to this question may provide insight into the stability of LLMs, suggesting improvements or highlighting the need for greater control over generated outputs to maintain consistent accessibility.

To address these questions, we structured our methodology into four studies, described in Chapters 4, 5 and 6. Table 2 highlights the key differences between them. Both studies follow the same procedure, and Figure 8 illustrates their common workflow. The first step involved selecting the LLMs to be used. The next step was choosing the screens to be requested (e.g., Login, User Profile, Music Player).

It is important to note that there are multiple ways to build the same graphical interface in native Android. The UI framework offers two paradigms: one based on XML [1], and another based on Jetpack Compose [2], where the interface is declarative and described directly in the Activity's code. Even after choosing a paradigm, various layout types can be used (e.g., Linear Layout, Constraint Layout, Column, Grid). These variations were considered during the planning of the prompts.

Subsequently, the prompts were designed, executed, and their outputs analyzed. The code was then extracted from the prompts and briefly modified to ensure a successful compilation. An Android project was created containing the generated screens, resulting in a native mobile application that included all the screens. Accessibility testing and analysis were performed on the app screens using Google's Accessibility Scanner[3], which generated compliance reports. Finally, we analyzed those reports to answer our research questions.

[1] https://developer.android.com/develop/ui/views/layout/declaring-layout
[2] https://developer.android.com/compose
[3] https://play.google.com/store/apps/details

Table 2 – Main differences between Studies.

	Main goal	**User Interface Approach**	**LLMs**
Study 1	Analyzed the accessibility of mobile application screen types generated by LLMs using various layouts, such as the Constraint Layout, as well as frameworks for creating mobile interfaces (i.e., XML-based approach and Jetpack Compose). The goal was to evaluate how the output of each layout and framework behaved in terms of accessibility.	Constraint Layout, Jetpack Compose and Layout Without Restrictions	ChatGPT-3.5
Study 2	Investigated the level of accessibility of the layout that presented the fewest errors (i.e., Compose). The idea was to evaluate whether this layout was, in fact, more accessible.	Jetpack Compose	ChatGPT-3.5, ChatGPT-4o and Sabiá-3
Study 3	Evaluated the accessibility of mobile application screens generated by an LLM using a prompt in English. The objective of this study was to examine the extent to which the LLM was capable of adapting to language changes and generating screens that adhered to accessibility standards.	Jetpack Compose	ChatGPT-4o
Study 4	Examined the accessibility of mobile application screens generated by an LLM specifically trained for code generation (e.g., GitHub Copilot). The focus of this study was to evaluate the quality of accessibility in the generated screens, assessing whether a code generation-oriented LLM could produce mobile layouts.	Jetpack Compose	GitHub Copilot (GPT-4o)

Source: Prepared by the author.

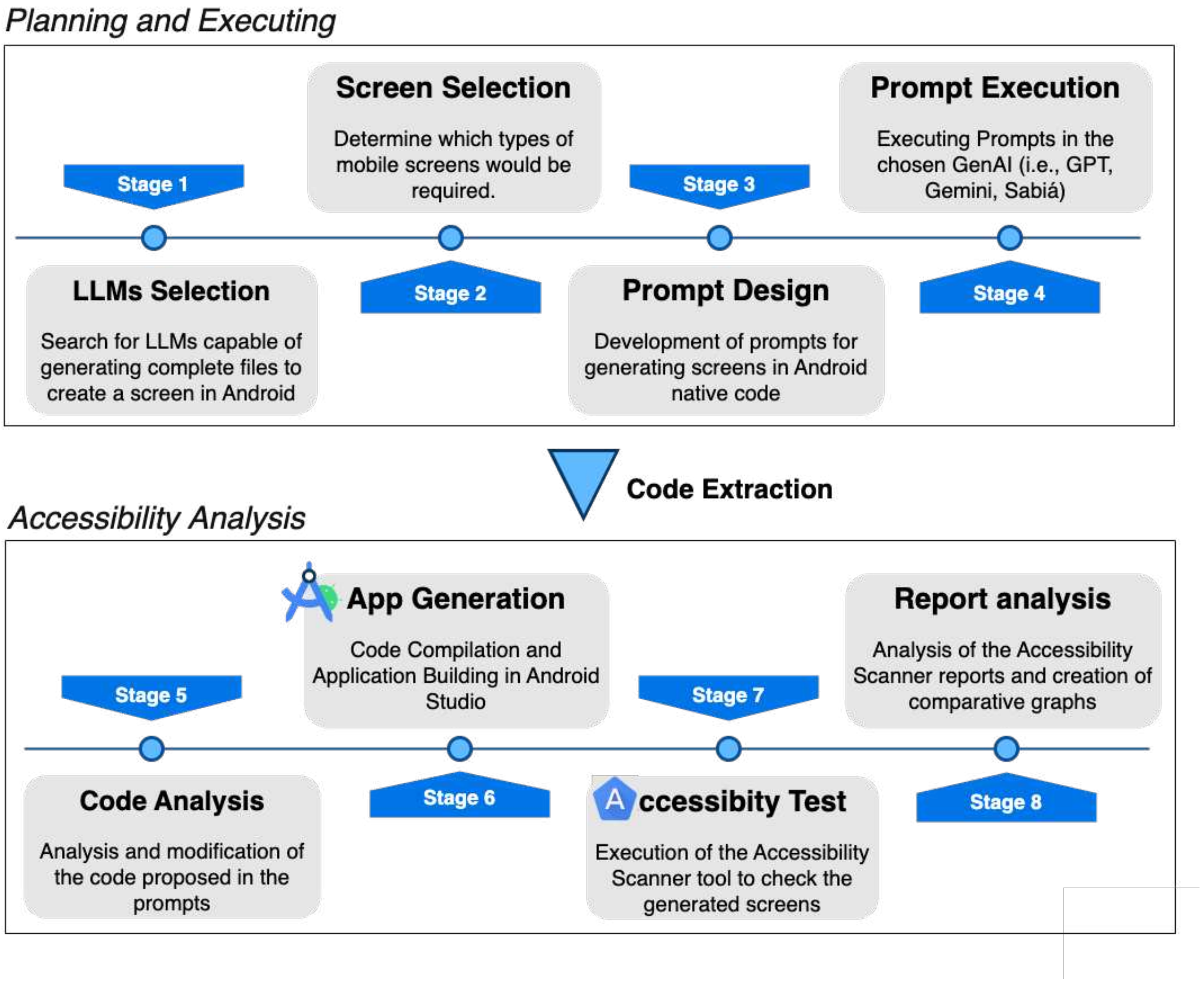
3.2 Screen selection

For the selection of the screens, we performed an exploratory visual examination of the 50 most popular apps on the Google Play Store. We analyzed the types of screen these apps most commonly featured and, based on this analysis, selected those we considered most relevant. This selection was further informed by informal conversations with developers and designers, who provided insights into which types of screens are generally viewed as important or frequently requested during the development process.

The relevance criteria also included our assumption regarding the kinds of screens that might interest both novice and experienced developers when interacting with an LLM. This assumption was not grounded in a formal study, but rather in these informal discussions. We recognize this as a limitation of our selection process.

In the end, seven types of screen were selected to be requested from the LLMs. Table 3 presents these types along with a description of the typical elements present in their graphical

Figure 8 – Common Workflow for Studies


Planning and Executing
Stage 1
LLMs Selection
Search for LLMs capable of generating complete files to create a screen in Android
Stage 2
Screen Selection
Determine which types of mobile screens would be required.
Stage 3
Prompt Design
Development of prompts for generating screens in Android native code
Stage 4
Prompt Execution
Executing Prompts in the chosen GenAI (i.e., GPT, Gemini, Sabiá)
Code Extraction
Accessibility Analysis
Stage 5
Code Analysis
Analysis and modification of the code proposed in the prompts
Stage 6
App Generation
Code Compilation and Application Building in Android Studio
Stage 7
Accessibity Test
Execution of the Accessibility Scanner tool to check the generated screens
Stage 8
Report analysis
Analysis of the Accessibility Scanner reports and creation of comparative graphs

Source: Prepared by the author.

interfaces.

3.3 Materials and Methods

This section describes the procedures and resources used to develop and evaluate the accessibility of the generated screens.

3.3.1 App Generation

To evaluate the accessibility of the LLM-generated screens, we developed a native Android application project using Android Studio. The project served as a centralized platform for organizing and testing all screen types under analysis.

For each LLM and different prompt languages, seven distinct APKs were generated, one for each screen type, resulting in a total of 35 APKs across the entire research. This

Table 3 – List of requested types of screen and their descriptions

Screen Type	Description
Login	The screen has fields for username/email and password, along with buttons for “Log In” and “Forgot Password”.
Registration Form	The screen has fields for first name, last name, email address, password, date of birth, and gender. Additionally, it includes buttons for “Submit” and “Cancel”.
Product Details	The screen displays information about a product, including its name, description, price, and image. It also includes buttons to add to the cart or favorites.
View Profile	The screen has a profile picture, fields for personal information (e.g., first name, last name, email, phone), and buttons for editing and saving. Additionally, it allows the user to update their information.
Music Player	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.
E-Commerce Sidebar Menu	The menu contains navigation items such as “Home”, “Promotions”, “My Orders”, “My Cart”, “My Account”, and “Log Out”.
To-Do List	The screen has a list with at least three items, each containing a checkbox, a task, and a date. The list is organized in descending order by date.

Source: Prepared by the author.

approach ensured that each LLMs outputs could be evaluated independently and fairly, without contamination from outputs generated by other models or prompt configurations.

All APKs were built using the minimum SDK version 29 (Android 10, Q), ensuring compatibility with the Accessibility Scanner and a wide range of contemporary devices. Minor modifications were made to the generated code to guarantee successful compilation (e.g., correcting missing imports or syntax issues). However, no semantic or structural changes were introduced, which preserved the authenticity of the LLMs’ output for accurate accessibility evaluation.

3.3.2 Accessibility Test

We used the Accessibility Scanner tool to evaluate applications. This Google tool assesses the interface of the app during manual navigation and identifies accessibility issues. Designed for developers and designers, it offers suggestions for improvement and generates detailed reports, identifying issues with contrast, font sizes, and button labels. The definitions of identified accessibility errors are available on the tool website[4].

[4] https://support.google.com/accessibility/android/faq/6376582?hl=pt-BR

Table 4 describes the issues detected by the Accessibility Scanner in this research. In addition, we include Context, which, while not classified as an error by the Scanner, is an important consideration related to the language and contextual appropriateness of UI elements.

Table 4 – Types of errors found by Accessibility Scanner and their descriptions.

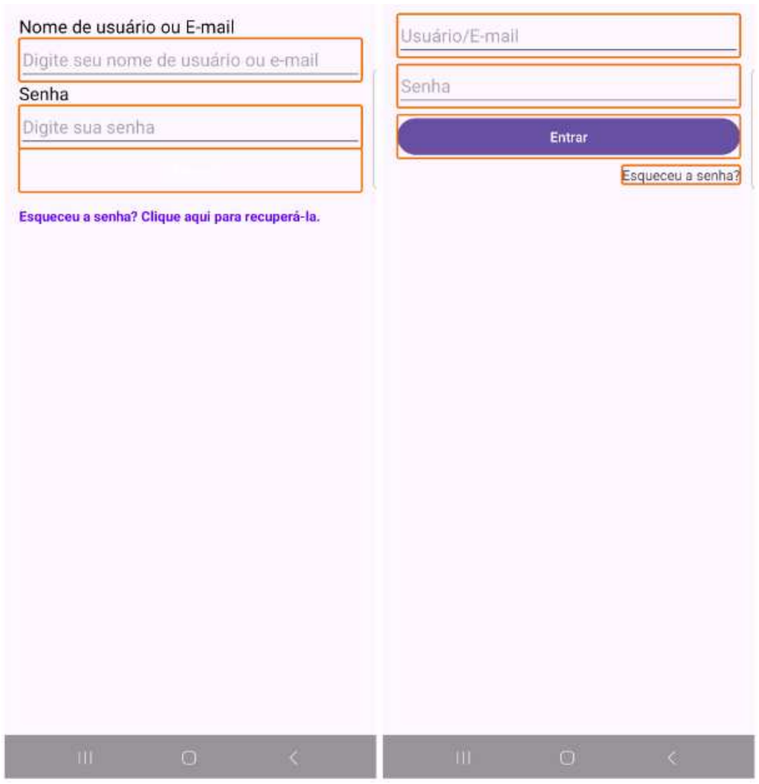
Error	Description
Context	Related to the context or language of the element, such as missing language indication for the interface or improper use of terms in a given context.
Touch Target	Indicates that a touch target (such as a button or link) is too small or too close to other elements.
Contrast	The contrast between text and background is insufficient, making reading difficult.
Item Description	Interactive elements (such as buttons or links) lack proper descriptions for screen readers.
Text Size	The text size is too small, making reading difficult.
Hidden Text	Hidden or invisible text that may be problematic for screen readers, either due to a lack of context or improper interpretation.
Item Label	Interactive elements (such as buttons and input fields) do not have clear labels, making it difficult to understand their purpose.
Item Type Label	The type of an interactive item is not properly identified, making navigation difficult for users relying on assistive technologies.
Editable Item Label	Editable fields (such as form inputs) lack clear labels or proper descriptions, making it harder to fill in the correct information.
Clickable Item	A clickable item does not have a clear indication of its interactivity, making it harder for users to identify it.

Source: Prepared by the author.

For example, Figure 9 highlights accessibility errors detected with the help of the Accessibility Scanner. The tool visually identified text contrast errors and issues with element labels on the login screens. Other studies (MARTINS *et al.*, 2022; LOPES *et al.*, 2022; ANDRADE *et al.*, 2024) have used the Accessibility Scanner as a testing tool to assess accessibility.

The accessibility testing process was conducted by a single evaluator using a Galaxy A54 5G smartphone. All tests were performed using the light theme of the smartphone. The choice of the light theme aims to ensure that the results are more relevant and applicable to the general public, as many users may not be familiar with dark themes or high-contrast modes. Furthermore, it is important to note that during the tests, image contrast errors were not considered. This decision was made because the use of certain images may lead to errors that do not necessarily reflect the accessibility of the generated interface.

Figure 9 – Two Login Screens Evaluated by the Accessibility Scanner. “Nome de usuário” and “Senha” mean username and password, respectively.


Nome de usuário ou E-mail
Digite seu nome de usuário ou e-mail
Senha
Digite sua senha
Esqueceu a senha? Clique aqui para recuperá-la.
Usuário/E-mail
Senha
Entrar
Esqueceu a senha?

Source: Prepared by the author.

3.4 Chapter Synthesis

This chapter describes the methodology used to assess the accessibility of LLM-generated mobile UI code. By defining clear research questions and systematically testing a variety of screen types, layouts, and prompt formulations, the study provides a robust framework to evaluate the behavior of LLMs in accessibility-sensitive contexts. The methodology takes advantage of real-world examples and employs a well-established testing tool, ensuring reliability and practical relevance. The next chapters present studies and discussions based on the four studies designed using this methodological foundation.

4 STUDY 1 - LAYOUT TYPE

This chapter presents Study 1 of this research, which investigates how different layout requirements influence the accessibility of the Android interface code generated by LLM. Section 4.1 justifies the LLM chosen in this study. In Section 4.2, we describe the design of the prompts, including the layout variations and the two types of prompt objectives: with and without accessibility requirements. Section 4.3 details the execution strategy, including how the prompts were run and how the outputs were selected. The results of the accessibility analysis are presented and discussed in Section 4.4. Finally, Section 4.5 summarizes the key findings of this study.

4.1 LLM selection

ChatGPT-3.5, developed by OpenAI, was selected as the LLM for this study due to its robust ability to understand and generate natural language text, along with its strong performance in producing functional and syntactically valid code. These features are particularly useful for tasks that involve prompt-based code generation for user interfaces. In addition, it supports a wide variety of programming languages and frameworks, including Android development, making it a flexible and practical tool for developers.

At the time this research was initiated, ChatGPT-3.5 was the most popular and widely used LLM available. Its accessibility, being free to use and intuitive interface made it a common choice among both experienced developers and novices. By selecting this model, our aim was to simulate a realistic usage scenario that reflects how a typical user might employ an LLM for mobile UI generation in real-world settings.

4.2 Prompt Design

The first step in the design process was to determine the essential elements that needed to be present on each screen, such as the structure and elements in the interface (e.g., which interface components are necessary on a Product screen). The next step was to clearly structure the prompts so that the LLMs could easily generate the required code.

Initially, we tested the prompts as continuous text, where everything that the screen needed was mentioned. Then, we created a structured prompt in bullet points, where the necessary items, such as the objective, restrictions, and screen elements, were clearly separated.

After analyzing the first results, we chose the second prompt model, as its output was more complete and suitable for compilation.

Finally, we determined that it would be crucial to understand how the LLM adapts to the native Android interface construction models and the various layout types available. Therefore, for each of the 7 screen types, we created 3 prompts, each requiring a different layout, as listed below:

- Chat-WR: layout without restrictions;
- Chat-CLR: Constraint Layout was explicitly required;
- Chat-JCR: Jetpack Compose was explicitly required.

Once the structure for the screen prompts was defined, two types of request were made: an initial one not requiring accessibility on the screens (NReq) and another that explicitly required accessibility (AccessReq). Figure 10 shows an example of a translated NReq prompt in a Chat-JCR that requests a Product Details screen. For the other prompts, an additional restriction was added that specifies the required layout type.

Figure 10 – Example of a Chat-JCR prompt requesting the “Product Details”.

Context: I am an Android mobile developer.
General Need: Help me develop a product detail screen for a native Android mobile application.
Screen Design Elements: This screen should display detailed information about a specific product, including its name, description, price, and image. Additionally, include interactive elements such as buttons to add to the cart or favorites. It should follow Material Design guidelines. Generate the texts for each element.
Constraints: Specify all the files needed to compile the code. Use Jetpack Compose.

Source: Prepared by the author.

The AccessReq prompt is simpler and is issued immediately after the NReq response. We informed that the screen generated by NReq had accessibility issues (even before testing it) and request its correction. The translated prompt is: “Rewrite the screen code to make it accessible.” It is worth noting that all prompts were written in the native language of the evaluators, **Brazilian Portuguese (pt-BR)**. Although we provided some structure in the prompt, NReq can be seen as the In-Context Prompt, which provides little context, task descriptions, and initial text to generate usable code or responses from natural language input, often following a "from scratch" approach without any contextual example (LI *et al.*, 2024), while the AccessReq prompt can be seen as a zero-shot prompt. We considered that novice developers would likely use this sort of approach.

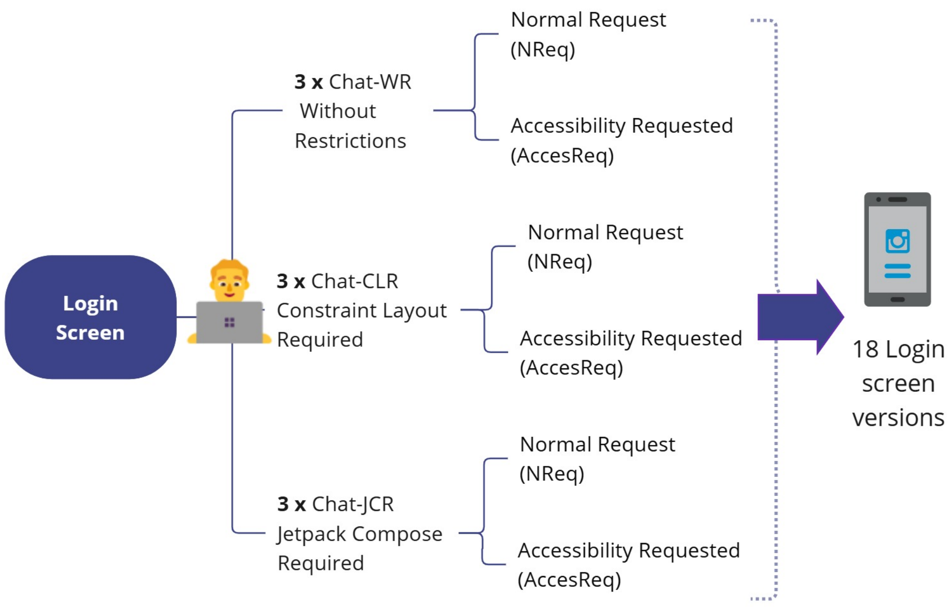
4.3 Prompt Execution

Once the prompts were defined, a structured approach was established to guide their execution, taking into account the potential variability in the responses generated by the models. To address this variability and ensure more consistent and reliable results, each prompt was executed multiple times. Specifically, for each type of screen, the prompt was repeated three times for each layout option, resulting in a total of nine prompt executions per screen.

Since both versions of each prompt were tested, one without explicit accessibility requirements (NReq) and one with accessibility explicitly requested (AccessReq), the process resulted in 18 generated versions for each screen. All generated outputs were carefully reviewed to identify incomplete responses or missing interface elements that could compromise the analysis.

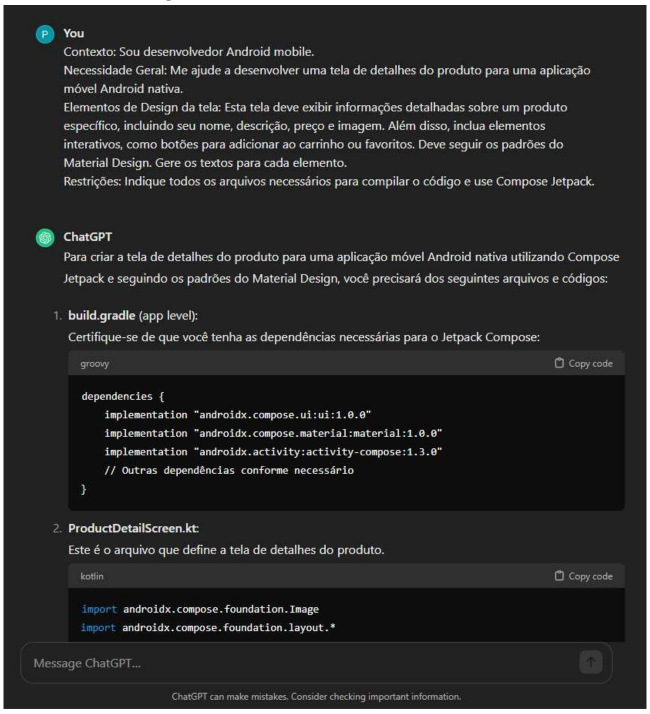
Figure 11 illustrates how this execution strategy was applied to the Login screen. As an example, Figure 12 shows a prompt with the NReq variation executed in ChatGPT 3.5. After NReq execution, the AccessReq variation was submitted in the same conversational context, as shown in Figure 13.

Figure 11 – Study 1 Prompt Execution Structure: Login Screen example


Login Screen
3 x Chat-WR Without Restrictions
3 x Chat-CLR Constraint Layout Required
3 x Chat-JCR Jetpack Compose Required
Normal Request (NReq)
Accessibility Requested (AccesReq)
Normal Request (NReq)
Accessibility Requested (AccesReq)
Normal Request (NReq)
Accessibility Requested (AccesReq)
18 Login screen versions

Source: Prepared by the author.

Figure 12 – NReq prompt requesting the Product Details screen in ChatGPT-3.5. The translated prompt is shown in Figure 10.



You
Contexto: Sou desenvolvedor Android mobile.
Necessidade Geral: Me ajude a desenvolver uma tela de detalhes do produto para uma aplicação móvel Android nativa.
Elementos de Design da tela: Esta tela deve exibir informações detalhadas sobre um produto específico, incluindo seu nome, descrição, preço e imagem. Além disso, inclua elementos interativos, como botões para adicionar ao carrinho ou favoritos. Deve seguir os padrões do Material Design. Gere os textos para cada elemento.
Restrições: Indique todos os arquivos necessários para compilar o código e use Compose Jetpack.

ChatGPT
Para criar a tela de detalhes do produto para uma aplicação móvel Android nativa utilizando Compose Jetpack e seguindo os padrões do Material Design, você precisará dos seguintes arquivos e códigos:

1. **build.gradle** (app level):
   Certifique-se de que você tenha as dependências necessárias para o Jetpack Compose:

   ```groovy
   dependencies {
       implementation "androidx.compose.ui:ui:1.0.0"
       implementation "androidx.compose.material:material:1.0.0"
       implementation "androidx.activity:activity-compose:1.3.0"
       // Outras dependências conforme necessário
   }
   ```

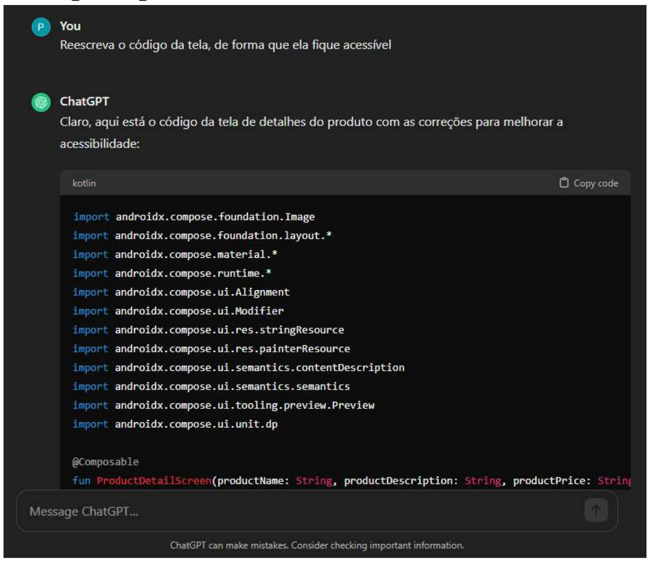
2. **ProductDetailScreen.kt:**
   Este é o arquivo que define a tela de detalhes do produto.

   ```kotlin
   import androidx.compose.foundation.Image
   import androidx.compose.foundation.layout.*
   ```

Copy code
Message ChatGPT...
ChatGPT can make mistakes. Consider checking important information.

Source: Prepared by the author.

Figure 13 – AccessReq prompt requesting an accessible Product Details screen in ChatGPT-3.5. The translated prompt means: “Rewrite the screen code to make it accessible.”.



You
Reescreva o código da tela, de forma que ela fique acessível

ChatGPT
Claro, aqui está o código da tela de detalhes do produto com as correções para melhorar a acessibilidade:

```kotlin
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp

@Composable
fun ProductDetailScreen(productName: String, productDescription: String, productPrice: Strin
```

Copy code
Message ChatGPT...
ChatGPT can make mistakes. Consider checking important information.

Source: Prepared by the author.

4.4 Results

In this first study with ChatGPT 3.5, prompts were generated for 126 screens (i.e., 7 x 18 versions). However, only 120 screens were evaluated; since for the "E-commerce Sidebar Menu" screen in Chat-WR, ChatGPT had already provided a code using the Constraint Layout.

All of these screens were manually evaluated with the Accessibility Scanner. A summary of accessibility error results by prompt type is presented in Table 5. The table also indicates the number of errors identified, the average number of errors, and the total standard deviation for each prompt.

Among the 120 screens evaluated, a total of 333 accessibility errors were identified, with an average of 2.78 errors per screen. In particular, the prompts requiring accessibility features ("AccessReq") accounted for more errors (192) than those without accessibility requirements ("NReq") (141).

Table 5 – Errors by type of prompt from Study 1.

	NReq	**AccessReq**	**Total**
# ERRORS	141	192	333
MEAN ERRORS	2.35	3.2	2.78
STANDARD DEVIATION	2.17	3.54	2.96

Source: Prepared by the author.

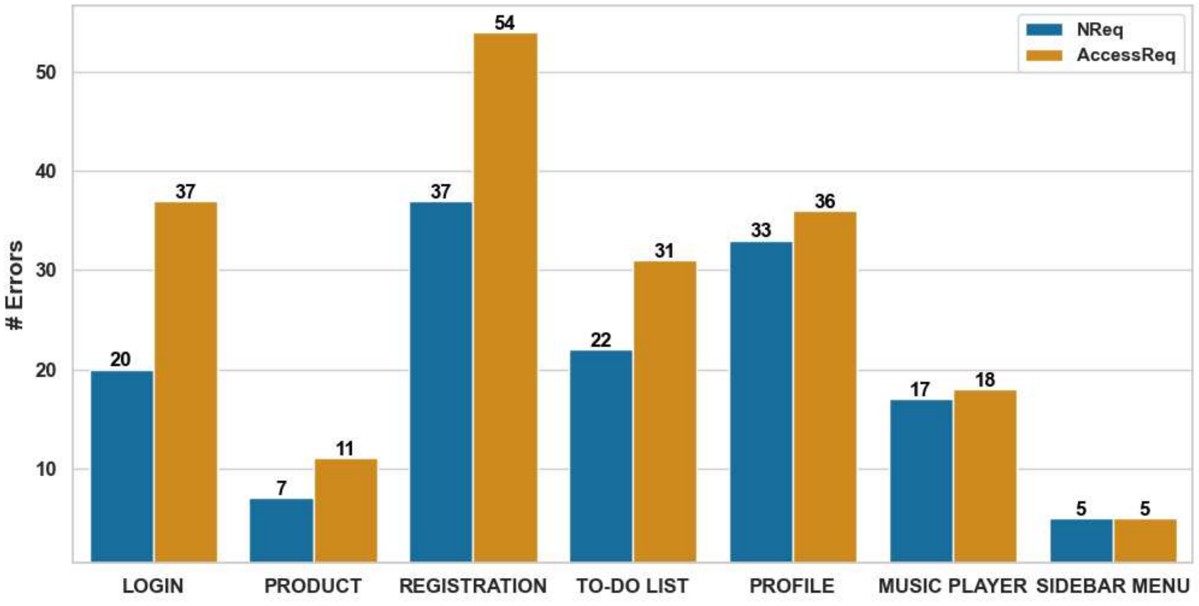
For prompts specifically requiring Jetpack Compose, AccessReq generally produced fewer errors compared to NReq. On two screens, AccessReq did not present accessibility errors. However, in four out of seven screens, AccessReq was not effective in reducing accessibility errors.

To analyze the difference between the two approaches, we compared the 60 prompts generated with NReq to their corresponding "AccessReq" versions. Only 8 of the 60 prompts showed a reduction in the number of errors. Applying a paired Student's t-test (STUDENT, 1908) to these comparisons, we obtained the following results: $t = 2.406, p = 0.01926$. The result ($p < 0.05$) indicates that the differences are statistically significant.

Figure 14 shows the distribution of accessibility errors in the prompts for each type of screen. Following the trend of the previous result, all the screens generated by the "AccessReq" prompts have more errors than the "NReq" prompts, except for the "E-Commerce Sidebar Menu" screen, which has the same number of errors.

Table 6 presents a heatmap detailing the number of accessibility errors found on the

Figure 14 – Accessibility Errors by Type of Screen and Prompts in Study 1.



Source: Prepared by the author.

screens, classified according to the required layout approach. As explained, we do not have data on the “E-Commerce Sidebar Menu” screen in Chat-CLR, as in Chat-WR, the screen was already generated using the Constraint Layout. The “Registration” screen has the highest total number of errors (91) and also presents the most errors in Chat-CLR and in general (56). Chat-CLR is the layout approach with the highest total number of errors (138), followed by Chat-WR (132). Only for the “Product” (16) and “To-Do-List” (19) screens did Chat-JCR exhibit more errors; for the other screens, this layout approach resulted in the fewest accessibility errors.

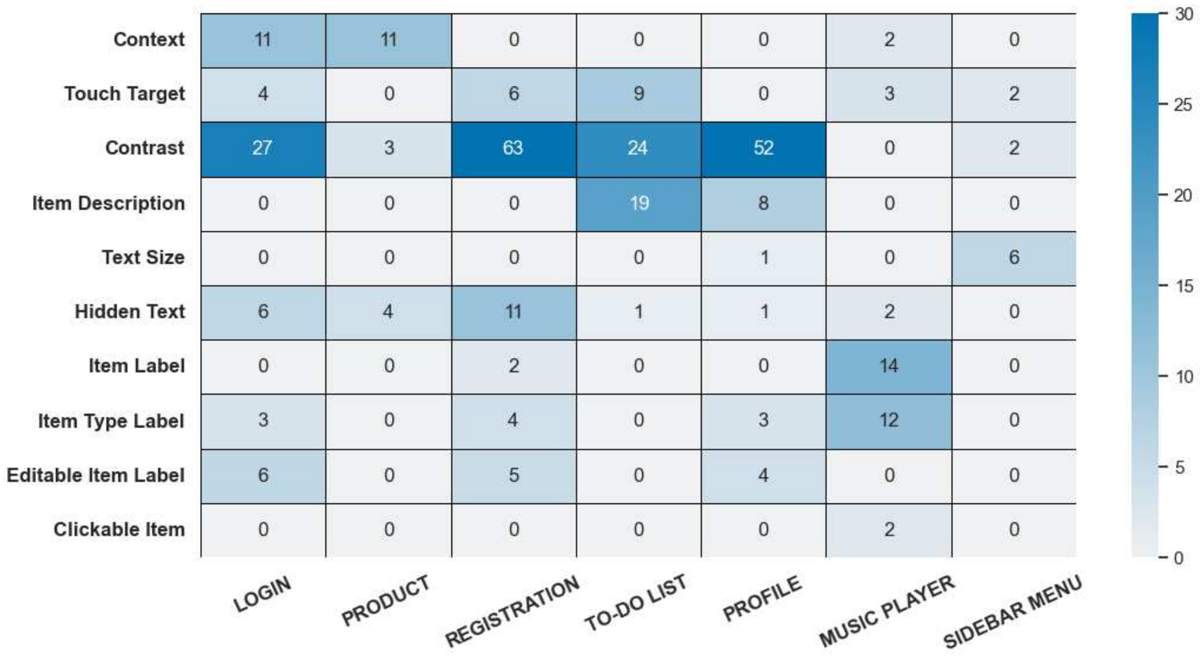
Table 6 – Accessibility errors for each type of screen, categorized by layout approach, in Study 1.

		LOGIN	PRODUCT	REGIS-TRATION	TO-DO LIST	PROFILE	MUSIC PLAYER	SIDEBAR MENU	TOTAL
Chat-WR	**NReq**	8	0	16	6	12	6	3	**51**
	AccessReq	18	0	17	10	23	10	3	**81**
Chat-CLR	**NReq**	6	0	20	9	13	8	—	**56**
	AccessReq	14	2	36	9	13	8	—	**82**
Chat-JCR	**NReq**	6	7	1	7	8	3	2	**34**
	AccessReq	5	9	1	12	0	0	2	**29**

Source: Prepared by the author.

Figure 15 shows a heatmap detailing accessibility errors by screen type, categorized by different aspects of accessibility (e.g, Text Size, Contrast, Item Description). The most common error identified was Contrast (171), which was the highest in four screens, while Clickable Item (2) was the least frequent error.

Figure 15 – Type of Accessibility errors for each type of screen in Study 1.

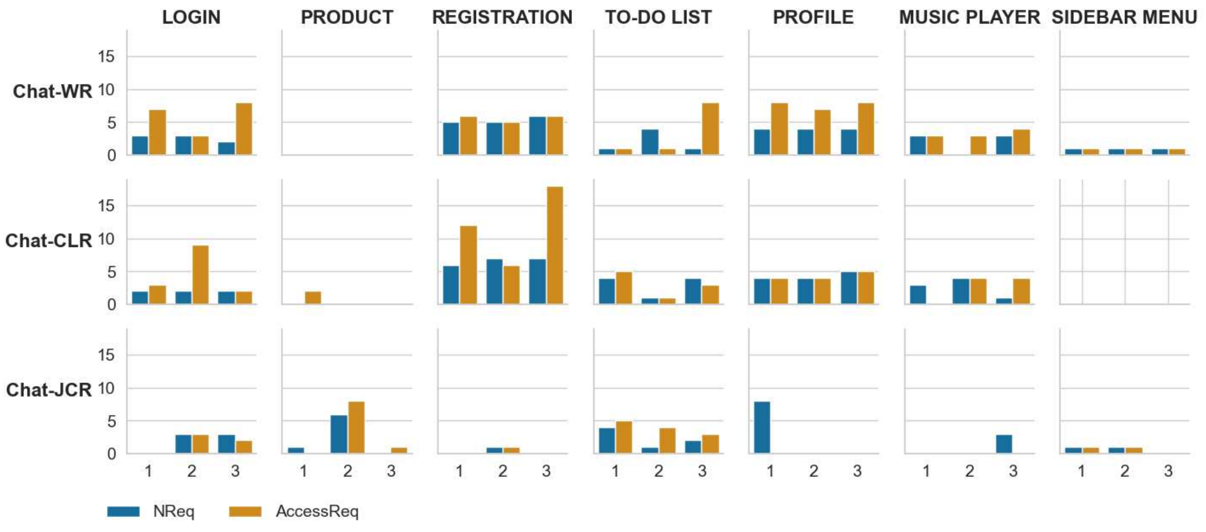


	LOGIN	PRODUCT	REGISTRATION	TO-DO LIST	PROFILE	MUSIC PLAYER	SIDEBAR MENU
Context	11	11	0	0	0	2	0
Touch Target	4	0	6	9	0	3	2
Contrast	27	3	63	24	52	0	2
Item Description	0	0	0	19	8	0	0
Text Size	0	0	0	0	1	0	6
Hidden Text	6	4	11	1	1	2	0
Item Label	0	0	2	0	0	14	0
Item Type Label	3	0	4	0	3	12	0
Editable Item Label	6	0	5	0	4	0	0
Clickable Item	0	0	0	0	0	2	0

Source: Prepared by the author.

Figure 16 shows a comparative analysis of the performance of ChatGPT-3.5 for each type of layout in the seven types of screen. For each screen, the results of the three individual attempts are displayed. Notably, Chat-CLR often registers higher AccessReq values, particularly on the “Registration” screen, presenting one prompt with 12 errors, another had 6 or more, and the third had 18 issues.

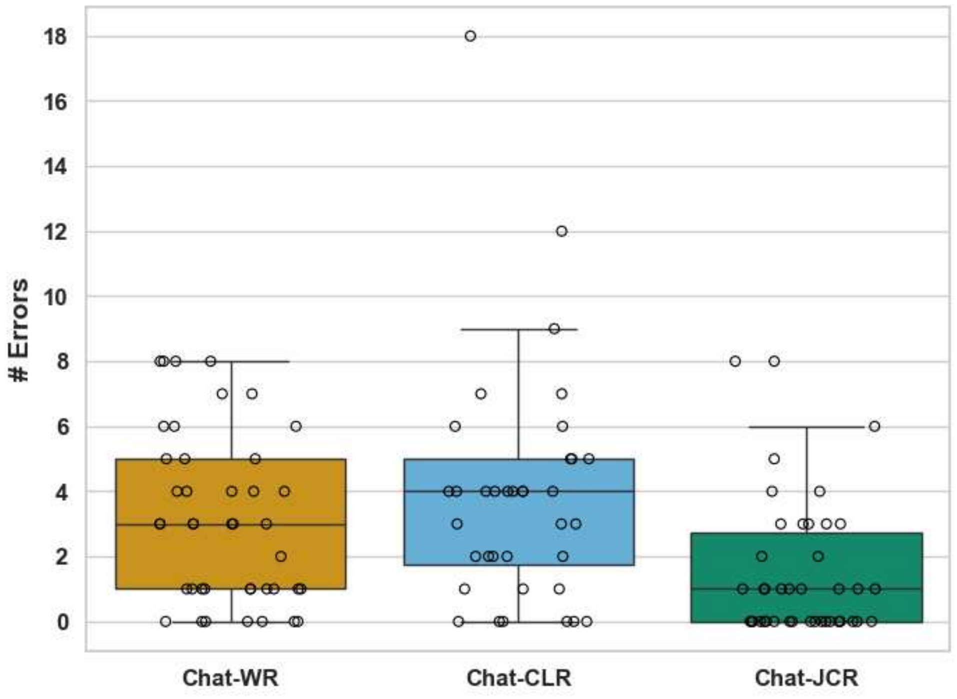
Figure 16 – Accessibility errors across layouts for each type of screens in each type of layout, in Study 1.



Source: Prepared by the author.

Figure 17 shows a boxplot that illustrates the number of accessibility errors found in all attempts for each layout investigated. It allows for a comparison of the performance of the different layouts, highlighting both the consistency and variability in accessibility errors between them. Chat-JCR showed less variation in the number of errors, while Chat-CLR had a higher average number of errors with greater variability.

Figure 17 – Boxplot showing the number of accessibility errors across all trials, for each layout investigated, in Study 1.



Source: Prepared by the author.

4.5 Chapter Synthesis

This first study revealed important insights into the impact of layout constraints on accessibility in LLM-generated Android interfaces. Contrary to expectations, prompts that explicitly requested accessibility features did not consistently lead to fewer accessibility errors. In fact, AccessReq prompts produced more errors in general, with only 8 of the 60 prompts, AccessReq showed a reduction in the number of errors.

Among layout strategies, Jetpack Compose (Chat-JCR) generally led to better accessibility outcomes, showing fewer errors and less variance in performance. Constraint Layout (Chat-CLR), while powerful, often resulted in the most accessibility issues.

The results obtained raise questions about the possible influence of language and the choice of a general-purpose LLM. These factors may have had a significant impact on the results. In the following chapters, these issues will be explored in more detail in order to clarify their impact on the analysis performed.

5 STUDY 2 AND 3 - LLM AND PROMPT LANGUANGE VARIATION

This chapter presents the methodology and findings of Studies 2 and 3, which explore the effects of linguistic variation in prompts on the accessibility performance of LLMs. While Study 2 (Section 5.1) investigates the differences between multiple LLMs using Brazilian Portuguese (PT-BR) prompts, Study 3 (Section 5.2) evaluates the impact of the prompt language by comparing English prompts with Portuguese ones, using a single LLM. Section 5.3 summarizes the key findings of these studies.

5.1 Study 2

Study 2 aims to evaluate and compare the accessibility performance of three different LLMs: ChatGPT-3.5, ChatGPT-4o, and Sabiá-3 when requested in Brazilian Portuguese. This study builds on the methodology used in the previous study. This section is structured into three parts: LLM selection (Subsection 5.1.1), prompt design and execution (Subsection 5.1.2), and the results analysis (Subsection 5.1.3).

5.1.1 LLMs selection

The selection of language models for Study 2 was guided by the intention to compare models with different architectures, levels of advancement, and regional focus. Initially, the study planned to include ChatGPT-4o (OpenAI's most recent public model as of mid-2024), Sabiá-3 (a large language model trained specifically for Brazilian Portuguese), and Gemini (Google's multilingual model). ChatGPT-4o was selected for being an evolution of GPT-3.5, offering improved reasoning capabilities and a wider context understanding, and it had recently become partially available for free. Sabiá-3 was included due to its focus on local development and potential advantages in processing prompts written in Portuguese.

However, during preliminary testing in July 2024, Gemini exhibited major limitations. The model frequently failed to complete the code blocks or would abruptly stop during generation, significantly impacting the reliability and consistency of the output. As a result, Gemini was excluded from the final experiment. The study thus proceeded with three models: ChatGPT-3.5 (used in Study 1 as a baseline), ChatGPT-4o (to assess improvements in newer architectures), and Sabiá-3 (to explore performance differences in a Portuguese native model).

5.1.2 Prompt Design and Execution

In Study 1, the results showed that the interfaces developed with Jetpack Compose had fewer accessibility errors. This finding suggests that Jetpack Compose may offer notable advantages in terms of accessibility for people with disabilities (PwD). Therefore, it was the only layout required in study 2 (Chat-JCR). Jetpack Compose is a more recent and modern framework for creating Android apps and has been receiving continuous updates and improvements.

The execution of the prompts followed the same procedure as in the previous study, with a normal request (NReq) and a request that explicitly considered accessibility (AccessReq) in each conversation. The prompts were executed exclusively using the Jetpack Compose layout, as exemplified in Figure 10. For each of the three LLMs (ChatGPT-3.5, ChatGPT-4o, and Sabiá-3), and for each of the seven screen types, the evaluator repeated the prompt three times. In total, 42 screen codes were generated by ChatGPT-3.5, 42 by ChatGPT-4o, and 42 by Sabiá-3, which is 126 versions of the 7 requested screens.

5.1.3 Results

A summary of the accessibility error results by prompt type is presented in Table 7. The table also indicates the number of errors identified, the average number of errors, and the total standard deviation for each prompt. Among the 126 screens evaluated, a total of 270 accessibility errors were identified, with an average of 2.14 errors per screen. Notably, the prompts requiring accessibility features (“AccessReq”) accounted for more errors (144) than those without accessibility requirements (“NReq”) (126).

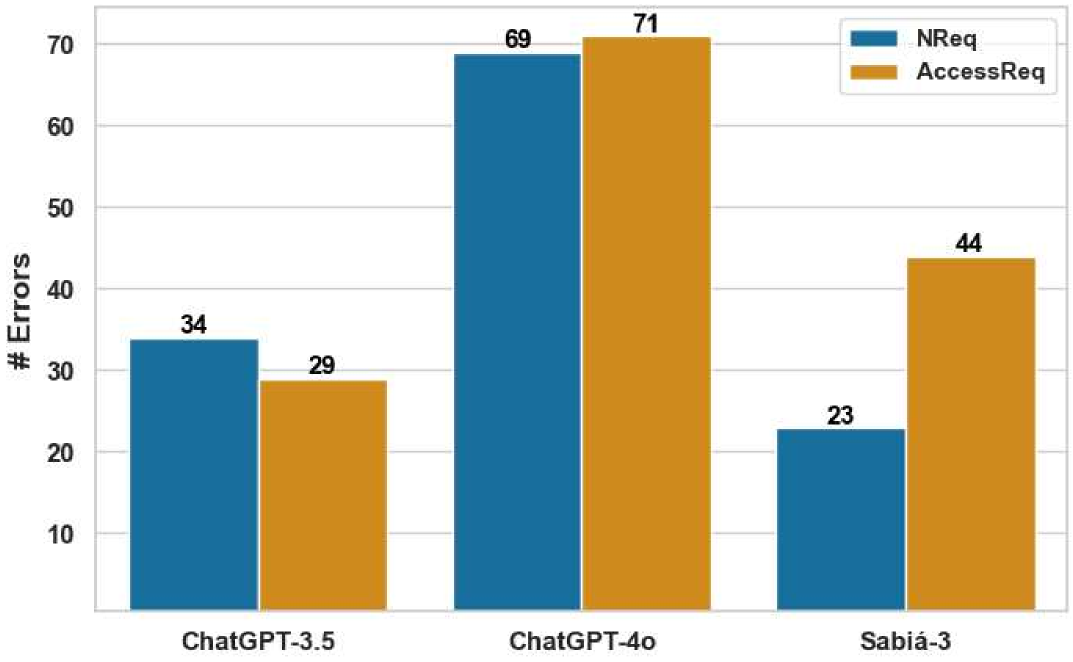
Table 7 – Errors by type of prompt from Study 2.

	NReq	**AccessReq**	**Total**
# ERRORS	126	144	270
MEAN ERRORS	2	2.29	2.14
STANDARD DEVIATION	2.53	2.45	2.48

Source: Prepared by the author.

Figure 18 shows the number of accessibility errors in the prompts for each LLM. ChatGPT-4o was the model with the highest number of errors (140), followed by Sabiá-3 (67) and ChatGPT-3.5 (63). The AccessReq prompt had the most errors in ChatGPT-4o (71) and Sabiá-3 (44), while in ChatGPT-3.5 (29), the NReq prompt had more errors (34).

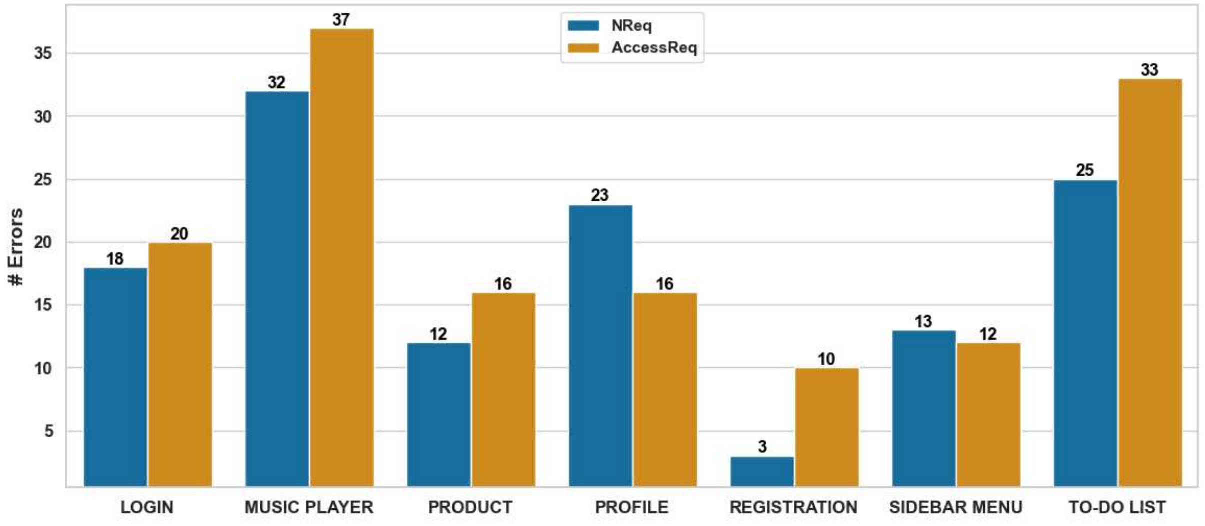
Figure 18 – Accessibility errors in each LLM, for each type of prompt, in Study 2.



Source: Prepared by the author.

Figure 19 shows the distribution of accessibility errors in the prompts for each type of screen. In 5 of the 7 types of screens, those generated by the “AccessReq” prompts have more errors than the “NReq” prompts, except for the “Sidebar Menu” and “Profile” screens, where the opposite is observed.

Figure 19 – Accessibility errors in each screen, for each type of prompt, in Study 2.



Source: Prepared by the author.

Table 8 shows the number of errors identified by the Accessibility Scanner for each type of error, categorized by model and prompt. ChatGPT-4o was the model with the highest number of Context errors (74), while ChatGPT-3.5 had the fewest (22). ChatGPT-4o also showed the most errors in both prompts, with 69 errors in NReq and 71 in AccessReq. Sabiá-3 had the highest number of errors in the AccessReq prompt, except for Touch Target errors, where it had the same count as the other models.

Table 8 – Accessibility Errors by Type of Prompt for Each Accessibility Error Category in Study 2.

		Context	**Touch Target**	**Contrast**	**Item Description**	**Hidden Text**	**Item Label**	**Item Type Label**
ChatGPT-3.5	**NReq**	12	6	9	7	0	0	0
	AccessReq	10	4	9	4	2	0	0
ChatGPT-4o	**NReq**	37	6	19	3	1	3	0
	AccessReq	37	3	18	1	2	1	9
Sabiá-3	**NReq**	12	4	2	4	1	0	0
	AccessReq	16	4	7	7	3	0	7

Source: Prepared by the author.

Table 9 illustrates the distribution of errors between different types of screen. The ChatGPT-4o model showed the highest number of errors on the screens “To-Do List” (27), “Profile” (30), and “Music Player” (46), while the Sabiá-3 model had more issues in the screens “Login” (14), “Registration” (7), and “Sidebar Menu” (11). ChatGPT-3.5 only showed more errors in the “Product” (16) screen. The highest number of errors was found on the “Music Player” screen with NReq (22) and AccessReq (24) in ChatGPT-4o.

Table 9 – Errors for each type of prompt, for each type of screen, in Study 2.

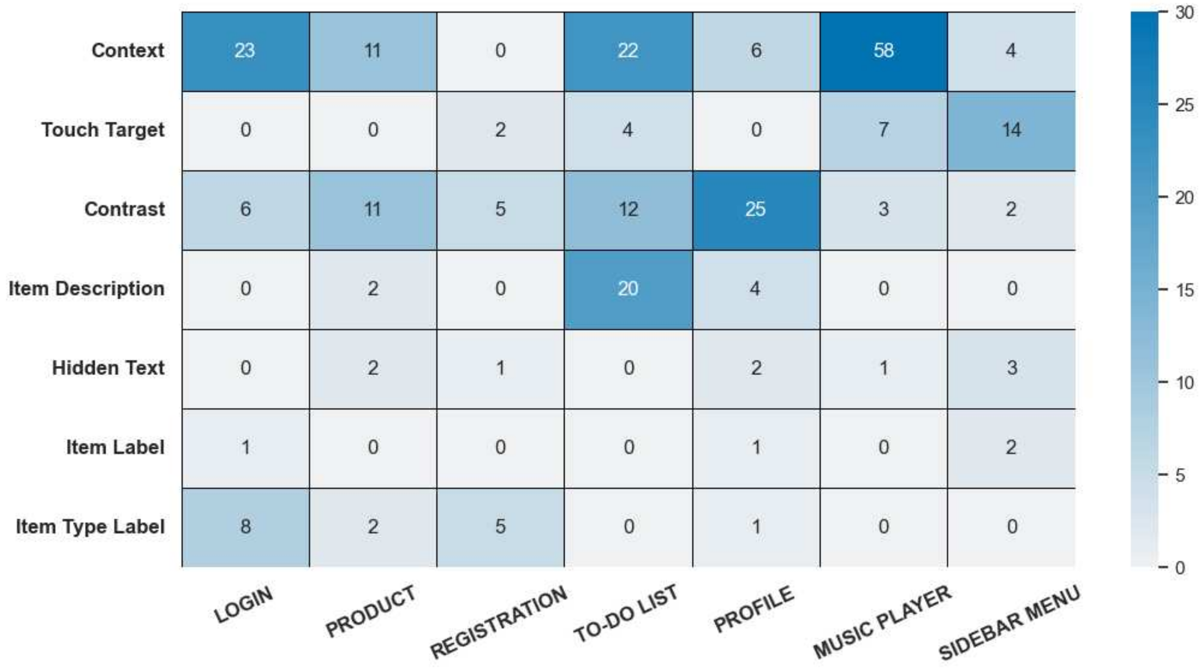
		LOGIN	**PRODUCT**	**REGIS-TRATION**	**TO-DO LIST**	**PROFILE**	**MUSIC PLAYER**	**SIDEBAR MENU**
ChatGPT-3.5	**NReq**	6	7	1	7	8	3	2
	AccessReq	5	9	1	12	0	0	2
ChatGPT-4o	**NReq**	7	4	0	15	15	22	6
	AccessReq	6	6	4	12	15	24	4
Sabiá-3	**NReq**	5	1	2	3	0	7	5
	AccessReq	9	1	5	9	1	13	6

Source: Prepared by the author.

Figure 20 shows a heatmap detailing accessibility errors by screen type, categorized by different aspects of accessibility. The most common error identified was Context (124), which was the highest in four screens, while Item Label (4) was the least frequent error.

Figure 21 presents a comparative analysis of the performance of ChatGPT-3.5, ChatGPT-4o, and Sabiá-3 in 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 “Product” in ChatGPT-3.5, one prompt resulted in 1 error, another had 6 or more,

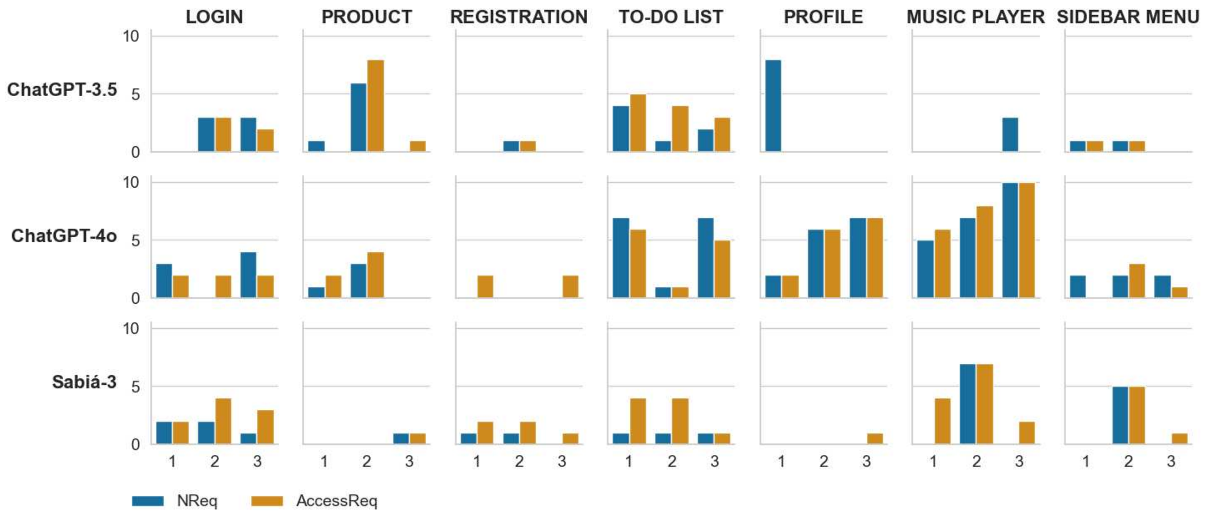
Figure 20 – Type of Accessibility errors for each type of screen, in Study 2.



Source: Prepared by the author.

and the third had none.

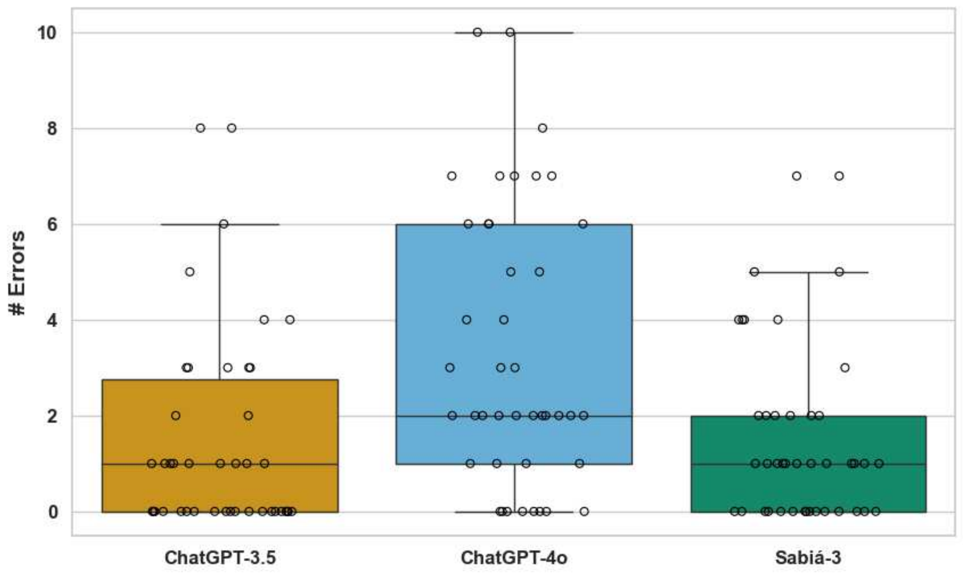
Figure 21 – Accessibility errors across LLMs for each type of screens, in Study 2.



Source: Prepared by the author.

Figure 22 presents a boxplot that illustrates the number of accessibility errors found in all attempts for each LLM investigated. It allows for a comparison of the performance of the different LLMs, highlighting both the consistency and variability in accessibility errors between them. Sabiá-3 showed less variation in the number of errors, while ChatGPT-4o had a higher

Figure 22 – Boxplot showing the number of accessibility errors across all 42 trials, for each of the LLMs investigated, in Study 2



Source: Prepared by the author.

average number of errors with greater variability.

We applied Student's t-test by pairing the evaluation results two by two. In only one combination, the difference in the number of errors was not significant $(p < 0.05)$; for the other two combinations, it was significant. For ChatGPT-3.5-Sabiá-3, the t-value was 0.206687 and the p-value was 0.83728. For ChatGPT-3.5-ChatGPT-4o, the t-value was 3.34968 and the p-value was 0.00174. And finally, for Sabiá-3-ChatGPT-4o, the t-value was 3.552866 and the p-value was 0.00098. In summary, the results indicate that there are significant differences between ChatGPT-3.5 and ChatGPT-4o, and between Sabiá-3 and ChatGPT-4o. However, there is no significant difference between ChatGPT-3.5 and Sabiá-3.

To analyze the difference between the two approaches (NReq and AccessReq), we compared the 63 prompts generated with "NReq" to their corresponding "AccessReq" versions. The results of this comparison are as follows: only 10 of the 63 prompts showed a reduction in the number of errors, an increase was shown by 24, and the number of errors remained the same for 29. Subsequent to the application of a paired Student's t-test to these comparisons, we obtained the following results: $t = -1.374, p = 0.17434$. The result $(p > 0.05)$ indicates that the differences are not statistically significant.

5.2 Study 3

In Studies 1 and 2, all the prompts were written in Brazilian Portuguese (PT-BR). In this study, we explore the impact of changing the prompt language to English. The goal is to investigate whether the use of English affects the accessibility performance of the generated code. Study 3 is based on the methodology employed in previous studies. This section is structured into three parts: LLM selection (Subsection 5.2.1), prompt design and execution (Subsection 5.2.2), and the results analysis (Subsection 5.2.3).

5.2.1 LLM selection

The initial plan for Study 3 was to evaluate the performance of both ChatGPT-4o and Sabiá-3 when generating Android screen code based on English language prompts. The inclusion of Sabiá-3 aimed to explore whether a model primarily trained for Portuguese could maintain consistent performance when asked in a different language. Meanwhile, ChatGPT-4o, being a state-of-the-art model released by OpenAI, represented a strong benchmark for evaluating prompt language effects on accessibility-aware code generation.

However, during the prompt execution phase, considerable issues arose with the outputs produced by Sabiá-3, mainly due to inconsistencies between builds and version-related discrepancies. As a result, the model was excluded from this study and the analysis focused solely on the output generated by ChatGPT-4o. This decision allowed for a more controlled investigation into the effect of changing the prompt language from Portuguese (used in previous studies) to English, isolating this variable without interference from architectural or training discrepancies between LLMs.

5.2.2 Prompt Design and Execution

Other studies (WANG *et al.*, 2024a; KOYANAGI *et al.*, 2024) demonstrated that LLMs frequently exhibit suboptimal performance in source code generation when instructed in languages other than English. Consequently, we chose the prompt language in this study as English.

The execution of the prompts followed the same procedure as in previous studies, with a normal request (NReq) and a request that explicitly considered accessibility (AccessReq) in each conversation. The prompts were executed exclusively using the Jetpack Compose layout, as exemplified in Figure 10 and in the English language. For each of the seven screen types, the evaluator repeated the prompt three times. In total, 42 screen codes were generated by ChatGPT-4o.

5.2.3 Results

A summary of the accessibility error results by prompt type is presented in Table 10. The table also indicates the number of errors identified, the average number of errors, and the total standard deviation for each prompt. Among the 42 screens evaluated, a total of 47 accessibility errors were identified, with an average of 1.12 errors per screen. Unlike other studies, prompts without accessibility requirements (“NReq”) caused more errors (28) than those requiring accessibility features (“AccessReq”) (19).

Table 10 – Errors by type of prompt from Study 3.

	NReq	**AccessReq**	**Total**
# ERRORS	28	19	47
MEAN ERRORS	1.33	0.9	1.12
STANDARD DEVIATION	2.2	1.34	1.81

Source: Prepared by the author.

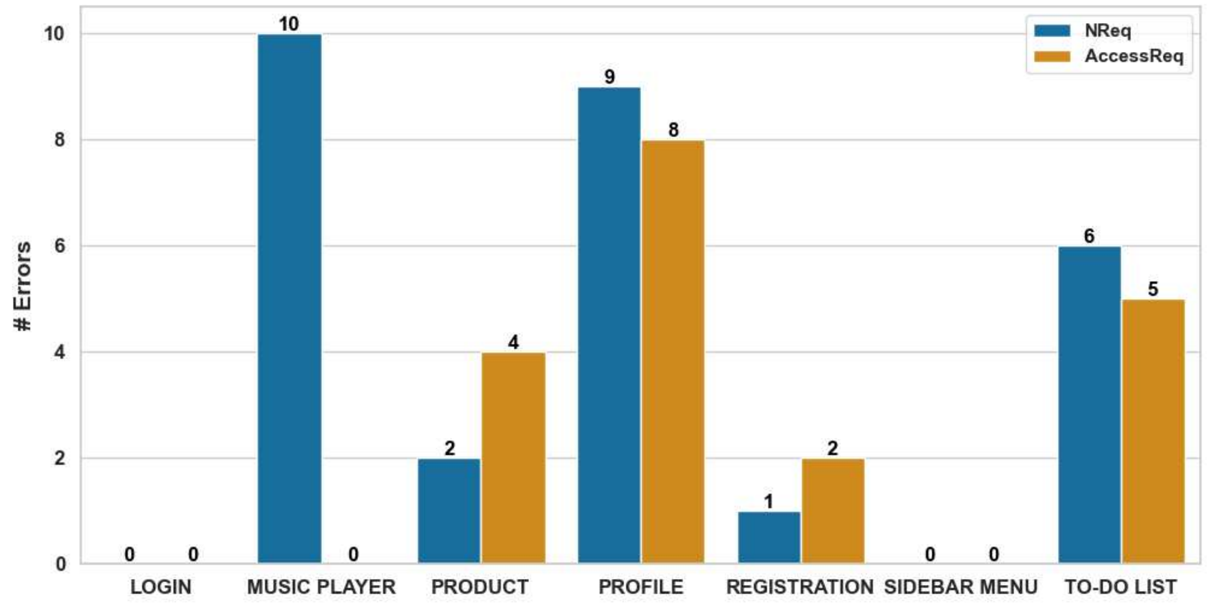
Figure 23 shows the distribution of accessibility errors in the prompts for each type of screen. Contrary to the findings of other studies, the AccessReq prompt exhibited a higher prevalence of errors only on the "Product" and "Registration" screens compared to the NReq prompt. In contrast, on all other screens, the opposite was observed. Notably, no errors were detected on the "Login" and "Sidebar Menu" screens.

Figure 24 shows a heatmap detailing accessibility errors by type of screens, categorized by different aspects of accessibility. The most common error identified was Contrast (36), especially in the "Profile" screen (17), while Hidden Text (2) was the least frequent error.

Figure 25 shows a comparative analysis of performance for the five types of screen. Since “Login” and “Sidebar Menu” did not show any errors, they were not added to the chart. Note that for each screen, the results of each of the three attempts are shown. There is a variation in accessibility between screen prompts. For instance, in “Music Player”, one prompt had 9 errors, another had 1, and the last one had none.

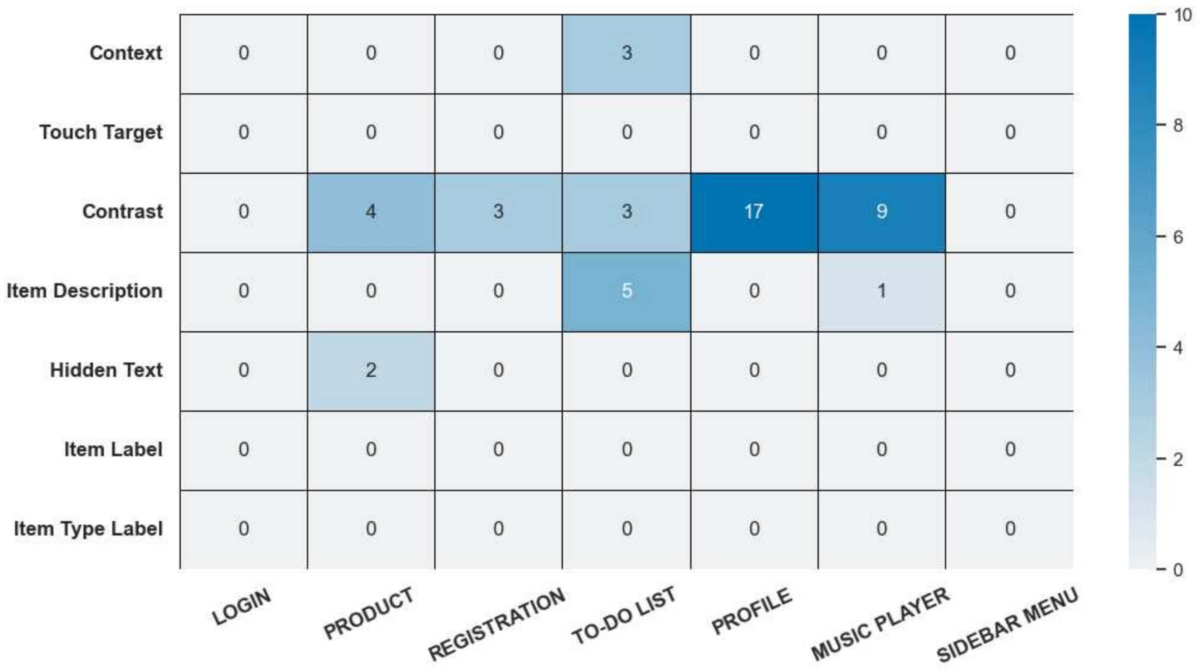
To analyze the difference between the two approaches, we compared the 21 prompts generated with NReq to their corresponding “AccessReq” versions. The results of this comparison

Figure 23 – Accessibility Errors by Type of Screen and Prompts in Study 3.



Source: Prepared by the author.

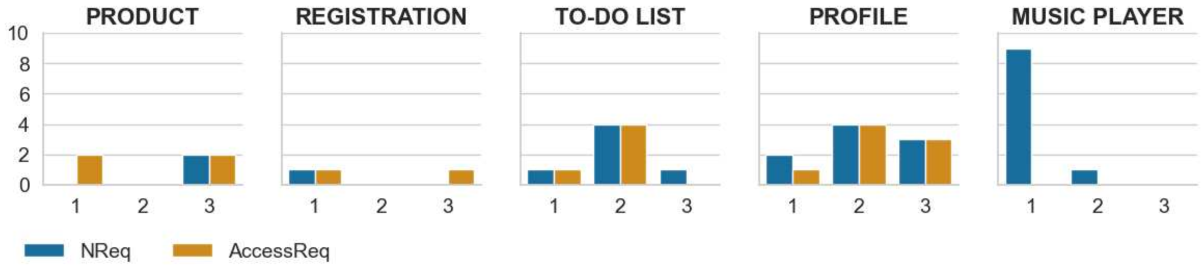
Figure 24 – Type of Accessibility errors for each type of screen, in Study 3.



	LOGIN	PRODUCT	REGISTRATION	TO-DO LIST	PROFILE	MUSIC PLAYER	SIDEBAR MENU
Context	0	0	0	3	0	0	0
Touch Target	0	0	0	0	0	0	0
Contrast	0	4	3	3	17	9	0
Item Description	0	0	0	5	0	1	0
Hidden Text	0	2	0	0	0	0	0
Item Label	0	0	0	0	0	0	0
Item Type Label	0	0	0	0	0	0	0

Source: Prepared by the author.

Figure 25 – Accessibility errors across prompts for each type of screens, in Study 3.



Source: Prepared by the author.

are as follows: 4 of the 21 prompts showed a reduction in the number of errors, an increase was shown by only 2, and the number of errors remained the same for 15. It is noteworthy that 6 of the 15 prompts are from the “Login” and “Sidebar Menu” screens, which do not show errors. Subsequent to the application of a paired Student’s *t*-test to these comparisons, we obtained the following results: $t = 0.952, p = 0.35253$. The result ($p > 0.05$) indicates that the differences are not statistically significant.

5.3 Chapter Synthesis

Study 2 revealed significant differences in accessibility performance among the LLMs tested. ChatGPT-4o generated the most accessibility errors, while ChatGPT-3.5 and Sabiá-3 performed similarly, with no statistically significant differences between them. Surprisingly, prompts that included accessibility instructions (AccessReq) did not produce fewer errors, in fact, they often performed worse.

Although Study 3 found that using English prompts with ChatGPT-4o resulted in fewer accessibility errors rate compared to Portuguese prompts used in Studies 1 and 2. Interestingly, accessibility instructions (AccessReq) prompts performed slightly better than standard prompts (NReq), although the difference was not statistically significant. These findings align with previous literature and suggest that English prompts can lead to better results in code generation tasks, particularly in terms of accessibility.

6 STUDY 4 - LLM FOR CODE GENERATION

This chapter presents Study 4, which builds on the methodology used in previous studies. The key difference in this study lies in the selection of LLM. We investigate the performance of a code-specific LLM, GitHub Copilot, in generating Android screen interfaces, with a particular focus on accessibility. Section 6.1 justifies the LLM chosen in this study. In Section 6.2, we describe the design of the prompts and detail the execution strategy. The results of the accessibility analysis are presented and discussed in Section 6.3. Finally, Section 6.4 summarizes the key findings of this study.

6.1 LLM selection

The LLM selected for this study was GitHub Copilot, which uses the GPT-4o model. GitHub Copilot[1] is designed specifically for code generation and is trained on a vast dataset that includes public and private code repositories. Unlike general-purpose models used in previous studies (SUH *et al.*, 2025; ALJEDAANI *et al.*, 2024; OTHMAN *et al.*, 2023), Copilot is optimized for software development tasks, making it an ideal candidate to assess whether a task-specific LLM can produce more accessible Android interfaces (MASTROPAOLO *et al.*, 2023; VAITHILINGAM *et al.*, 2022).

This study also provides a comparison between GitHub Copilot and OpenAI general-purpose models (ChatGPT) to explore the potential advantages of using a tool trained specifically for code generation.

6.2 Prompt Design and Execution

Based on previous studies, the objective of selecting GitHub Copilot was to determine the efficacy of the models that have been trained for the purpose of code generation. The central question guiding this research was to assess whether a model trained for this task could result in more accessible screens.

The execution of the prompts followed the same procedure as in previous studies, with a normal request (NReq) and a request that explicitly considered accessibility (AccessReq) in each conversation. The prompts were executed exclusively using the Jetpack Compose layout, as exemplified in Figure 10 and in the language Portuguese-Brazilian (pt-BR). For each of the

[1] https://github.com/features/copilot

seven screen types, the evaluator repeated the prompt three times. The total number of screen codes generated by GitHub Copilot (GPT-4o) was 42.

6.3 Results

A summary of the accessibility error results by prompt type is presented in Table 11. The table also indicates the number of errors identified, the average number of errors, and the total standard deviation for each prompt. Among the 42 screens evaluated, a total of 115 accessibility errors were identified, with an average of 2.74 errors per screen. Notably, prompts requiring accessibility features (“AccessReq”) accounted for more errors (61) than those without accessibility requirements (“NReq”) (54).

Table 11 – Errors by type of prompt from Study 4.

	NReq	AccessReq	Total
# ERRORS	54	61	115
MEAN ERRORS	2.57	2.9	2.74
STANDARD DEVIATION	2.84	2.9	2.84

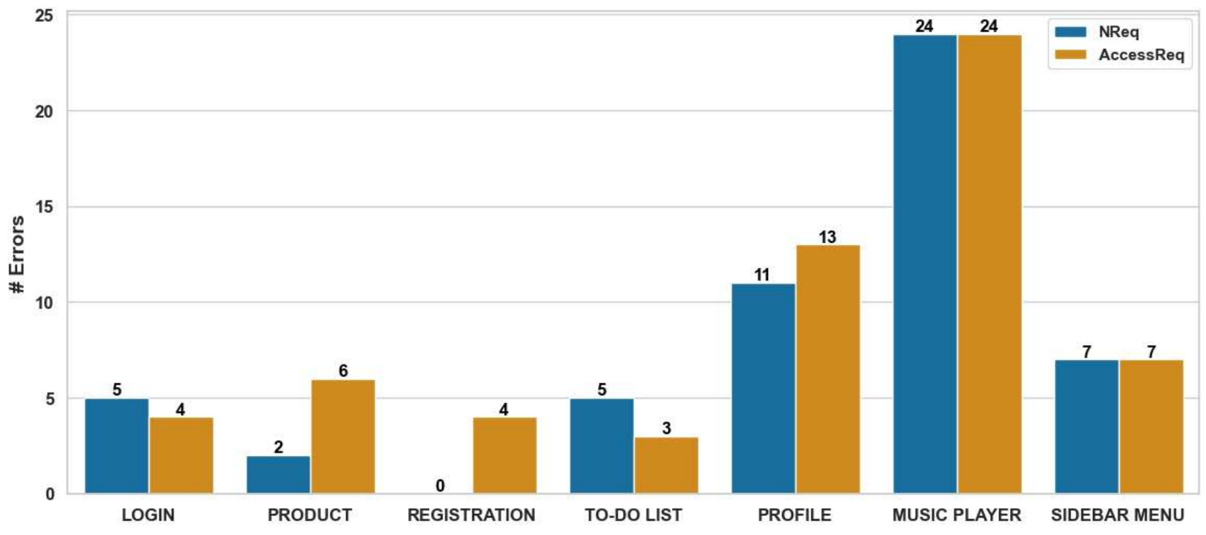
Source: Prepared by the author.

Figure 26 shows the distribution of accessibility errors in the prompts for each type of screen. In the screens “Login” and “To-Do List”, the NReq prompts exhibit a higher incidence of errors compared to the AccessReq prompts. In contrast, in “Product”, “Registration”, and “Profile”, AccessReq prompts demonstrate a greater prevalence of errors. The screens of “Music Player” and ´´Sidebar Menu” exhibited equivalent error counts, with “Music Player” registering the highest number (48).

Figure 27 shows a heatmap detailing accessibility errors by screen type, categorized by different aspects of accessibility. The most common error identified was Context (54), particularly in the "Music Player" screen (42). Conversely, Hidden Text and Item Description (2) were the least frequent errors.

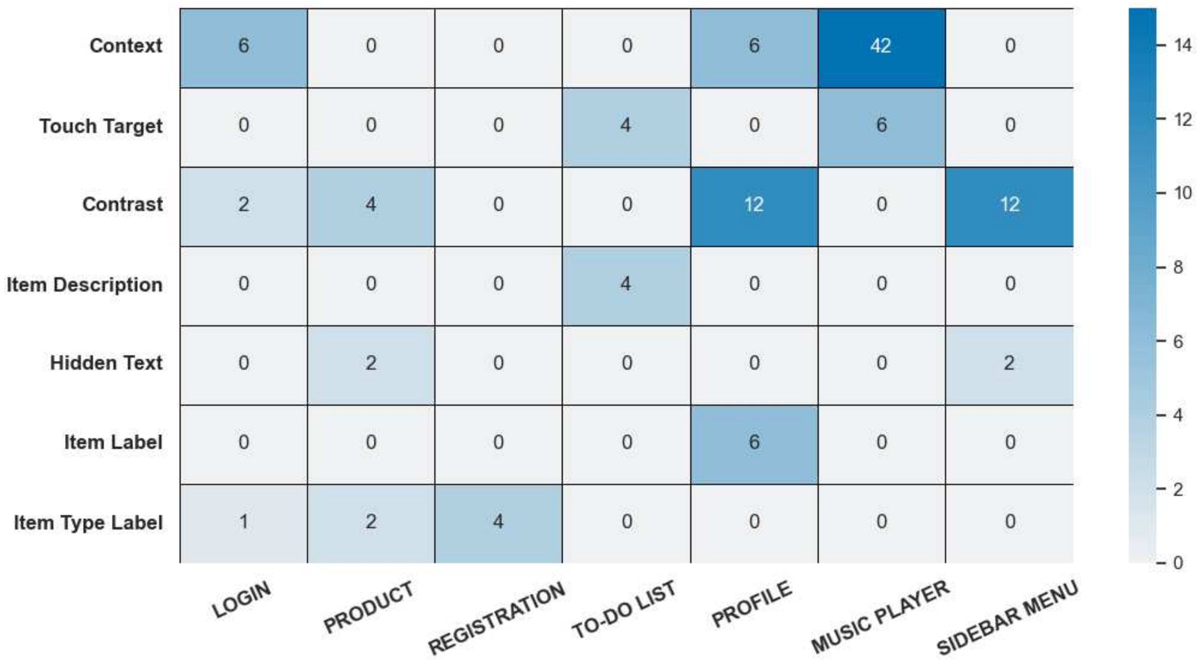
Figure 28 shows a comparative analysis of performance for the seven types of screen. Note that for each screen, the results of each of the three attempts are shown. There is a variation in accessibility between screen prompts. For instance, in "Product", one prompt had 1 error, another had 5, and the last one had none. It should be noted that only the “Music Player” and “Sidebar Menu” exhibited consistent patterns in terms of the number of errors for each prompt and request.

Figure 26 – Accessibility Errors by Type of Screen and Prompts in Study 4.



Source: Prepared by the author.

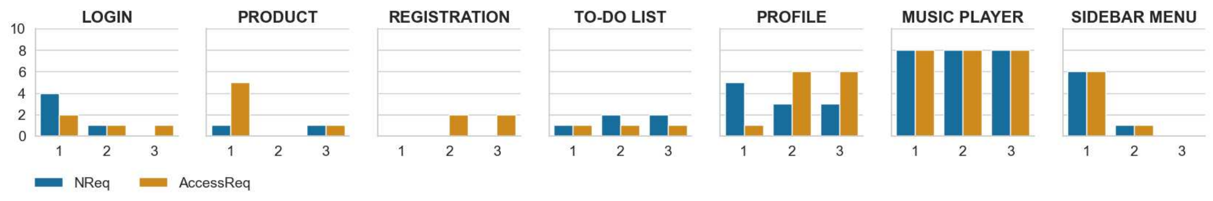
Figure 27 – Type of Accessibility errors for each type of screen in each LLM, in Study 4.



	LOGIN	PRODUCT	REGISTRATION	TO-DO LIST	PROFILE	MUSIC PLAYER	SIDEBAR MENU
Context	6	0	0	0	6	42	0
Touch Target	0	0	0	4	0	6	0
Contrast	2	4	0	0	12	0	12
Item Description	0	0	0	4	0	0	0
Hidden Text	0	2	0	0	0	0	2
Item Label	0	0	0	0	6	0	0
Item Type Label	1	2	4	0	0	0	0

Source: Prepared by the author.

Figure 28 – Accessibility errors across prompts for each type of screens, in Study 4.



Source: Prepared by the author.

To analyze the difference between the two approaches, we compared the 21 prompts generated with NReq to their corresponding "AccessReq" versions. The results of this comparison are as follows: a reduction in the number of errors was shown by only 4 of the 21 prompts, an increase was shown by 6, and the number of errors remained the same for 11. Subsequent to the application of a paired Student's t-test to these comparisons, we obtained the following results: $t = -0.863, p = 0.3984$. The result ($p > 0.05$) indicates that the differences are not statistically significant.

6.4 Chapter Synthesis

The results of Study 4 indicate that GitHub Copilot, despite being a domain-specific LLM designed for code generation, does not significantly outperform general-purpose models in terms of accessibility when generating Android UIs. In fact, the AccessReq prompts resulted in slightly more errors than the general NReq prompts. The most common issue was related to contextual accessibility problems, particularly in more complex screens such as "Music Player". The variation in the number of errors on repeated prompts also highlights inconsistency in the model output.

7 DISCUSSION

This chapter presents a comprehensive discussion of the findings derived from the four empirical studies carried out throughout this dissertation. The aim is to synthesize the key results, compare the behavior of the models across different scenarios, and critically analyze the implications of these findings in the context of generating accessible mobile interfaces using LLMs. The discussion is organized as follows: Section 7.1 provides a summary and comparison of the studies; Sections 7.2 to 7.5 address the research questions explored; Section 7.6 discusses the practical implications of the results; and Section 7.7 outlines the threats to the validity of the study.

7.1 Summary of the Studies

The objective of this section is to provide a comparative analysis of the four studies that were conducted. This analysis will summarize the main information and results from each study.

A summary of the accessibility error results by prompt type from all studies is presented in Table 12. The table also indicates the number of errors identified, the average number of errors, and the total standard deviation for each prompt. Among the 288 screens evaluated, a total of 702 accessibility errors were identified, with an average of 2.44 errors per screen. Notably, prompts requiring accessibility features ("AccessReq") accounted for more errors (387) than those without accessibility requirements ("NReq") (315).

Table 12 – Errors by type of prompt from all studies.

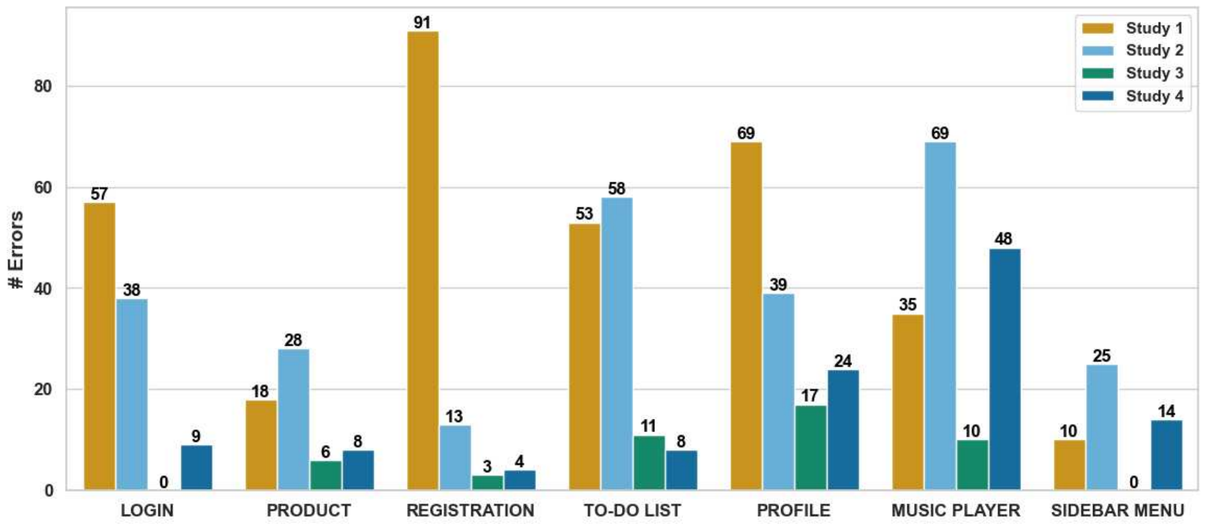
	NReq	**AccessReq**	**Total**
# ERRORS	315	387	702
MEAN ERRORS	2.19	2.69	2.44
STANDARD DEVIATION	2.44	3.0	2.74

Source: Prepared by the author.

To analyze the difference between the two approaches (NReq and AccessReq), we compared the 144 prompts generated with "NReq" to their corresponding "AccessReq" versions. We applied a paired Student's t-test to these comparisons and obtained the following results: $t = -2.739, p = 0.00694$. The result ($p < 0.05$) indicates that the differences are statistically significant.

Figure 29 shows the distribution of accessibility errors in the studies for each type of screen. In 3 of the 7 screens, Study 1 had the most errors, while for the rest of the screens, Study 2 had the most. The “Registration” screen in Study 1 had the highest number of errors (91) of all studies.

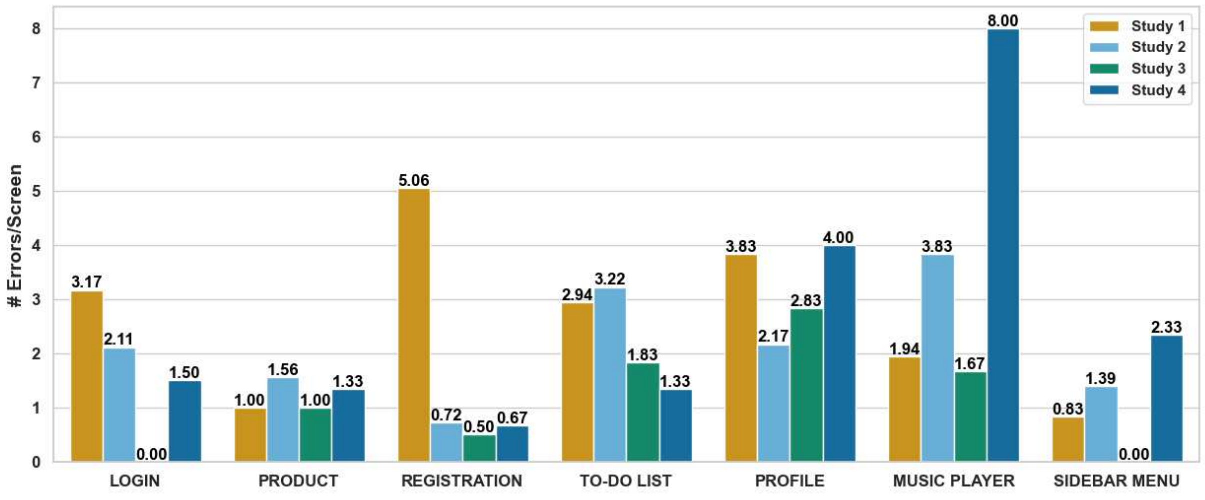
Figure 29 – Accessibility Errors by Type of Screen for each Study.



Source: Prepared by the author.

However, given the varying number of screens evaluated in the studies, Figure 30 presents the mean accessibility errors per screen in the studies, classified by screen type. The “Music Player” screen in Study 4 exhibited the highest mean (8), followed by the “Registration” screen in Study 1 (5.06).

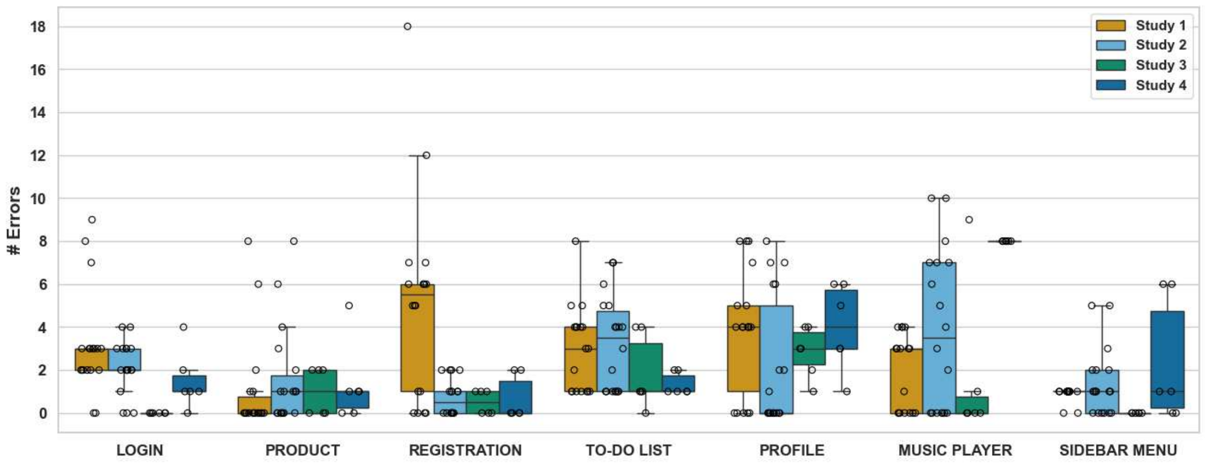
Figure 30 – Mean Accessibility Errors by Type of Screen for each Study.



Source: Prepared by the author.

Figure 31 shows a boxplot that illustrates the number of accessibility errors found in all attempts for each study. It allows for a comparison of the performance of the studies, highlighting both the consistency and variability in accessibility errors between them. “Login” screen showed less variation in the number of errors, while “Profile” and “Music Player” had a higher variation.

Figure 31 – Boxplot showing the number of accessibility errors across all trials, for each study.



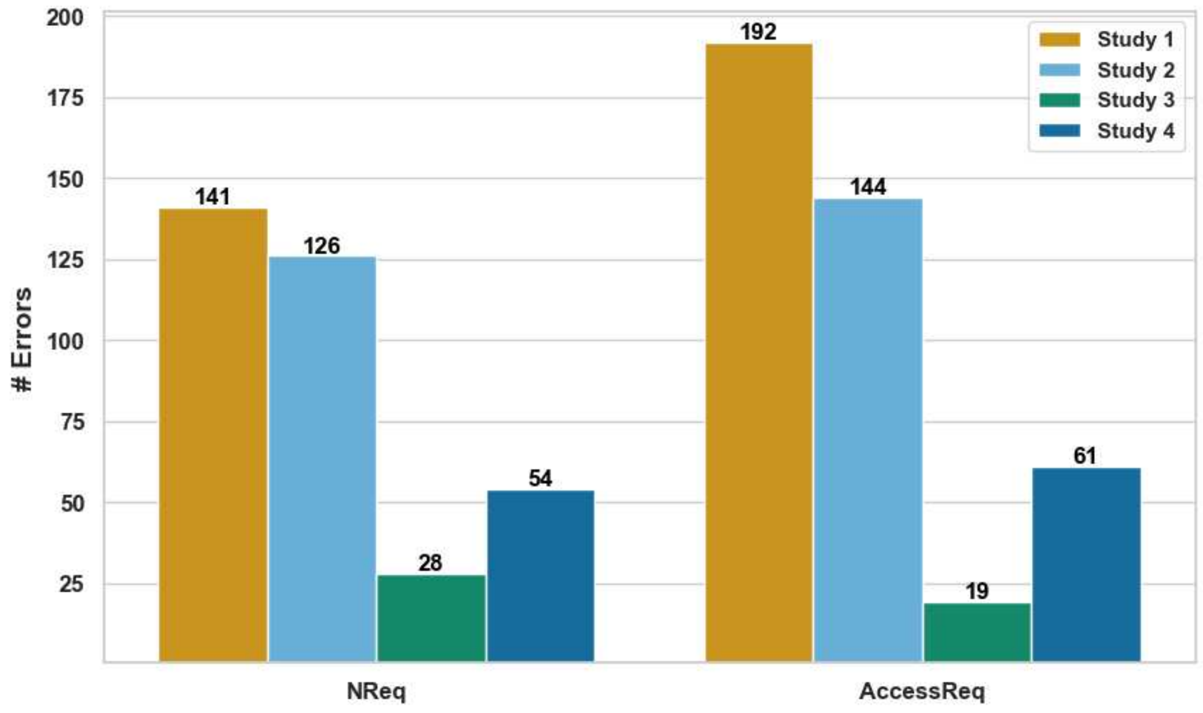
Source: Prepared by the author.

Figure 32 shows the distribution of accessibility errors in the studies for each type of request. In general, the AccessReq prompt (416) showed more errors than the NReq prompt (349). Following the same principle as in Figure 30, Figure 33 presents the mean accessibility errors per screen in the studies, categorized by request type.

Since the same LLM (ChatGPT-4o) was used in Studies 2 and 3, differing only in the prompt language, Figure 34 presents the distribution of accessibility errors in the studies for each type of screen. All types of screens in Study 2 were observed with a higher number of errors compared to Study 3. The "Music Player" screen (46) and the "Profile" screen (30) had more errors, both from Study 2.

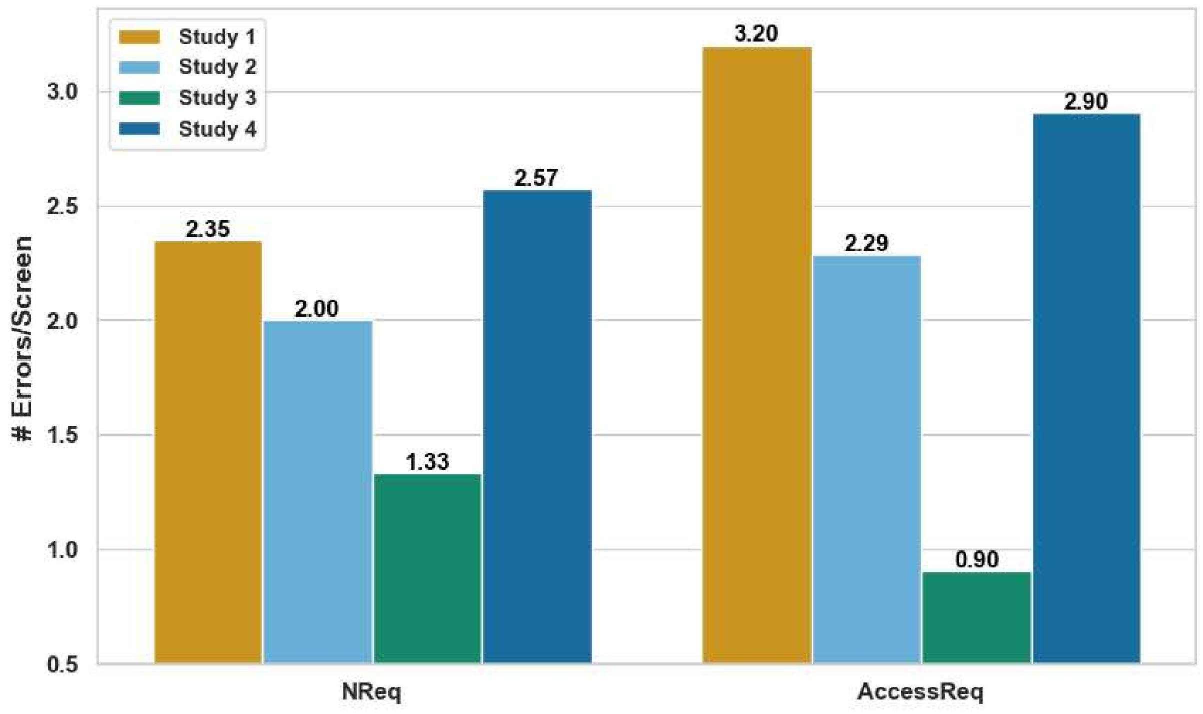
Figure 35 shows a heatmap detailing accessibility errors by studies, categorized by different aspects of accessibility. The most prevalent error identified was Contrast (301), followed by Context (205), while Text Size (7) and Clickable Item (2) were the least frequent errors.

Figure 32 – Accessibility Errors by Type of Request for each Study.



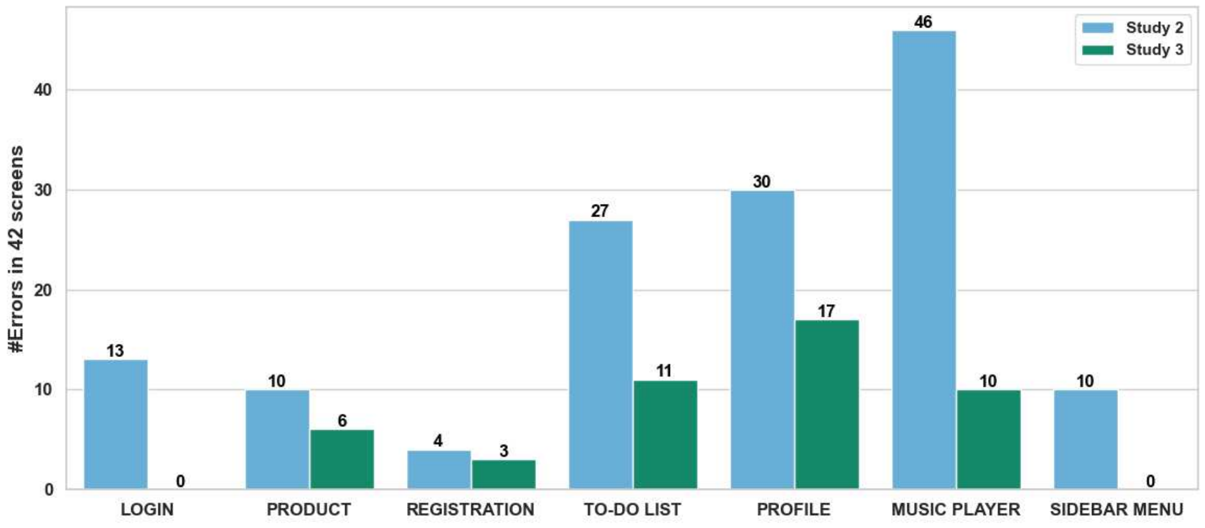
Source: Prepared by the author.

Figure 33 – Mean Accessibility Errors by Type of Request for each Study.



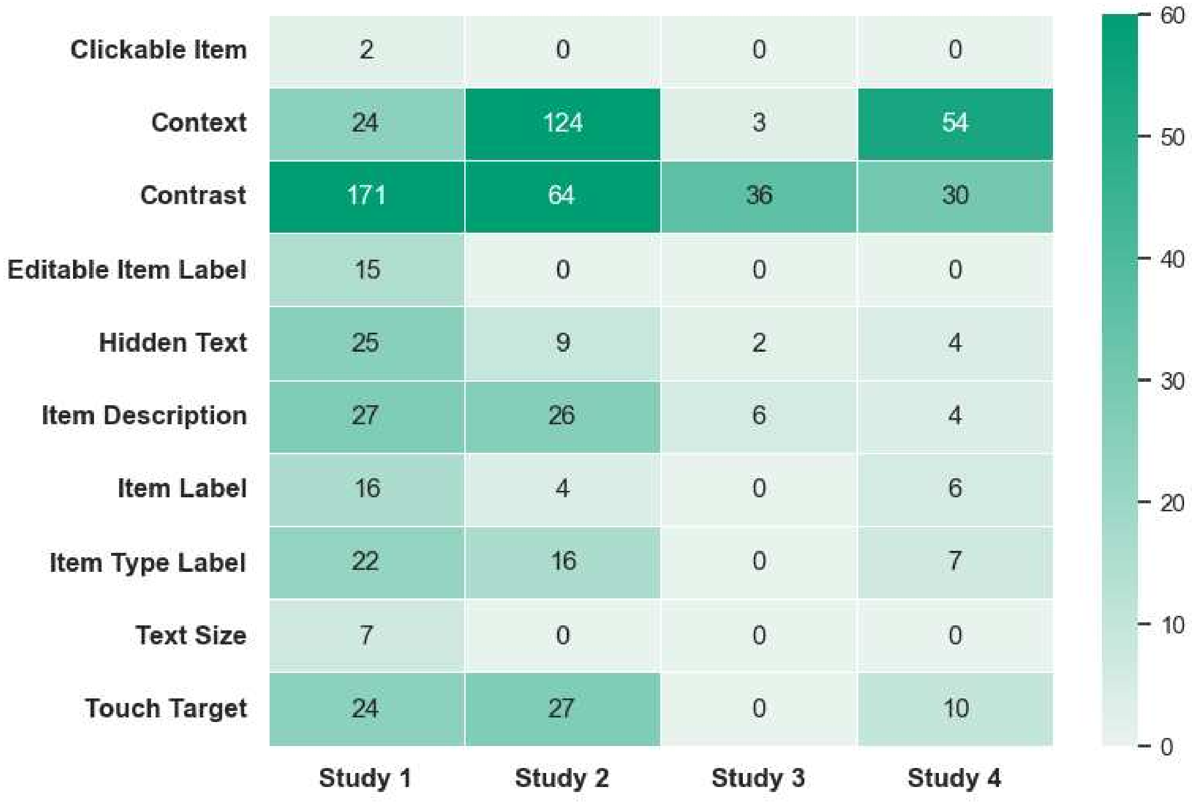
Source: Prepared by the author.

Figure 34 – Accessibility Errors by ChatGPT-4o.



Source: Prepared by the author.

Figure 35 – Type of Accessibility errors for each Study.



Source: Prepared by the author.

7.2 RQ1 - What is the accessibility level of the screens generated by LLMs for native Android?

The results of this study highlight significant issues regarding the accessibility of LLM-generated interfaces. In Study 1, the interfaces generated by ChatGPT-3.5 had an average of 2.78 accessibility errors per screen, with a total of 333 errors. In Study 2, 207 additional errors were identified across 84 screens generated by ChatGPT-4o and Sabiá-3, resulting in an average of 2.46 errors per screen. In Studies 3 and 4, more 47 and 115 errors were identified on 84 screens (42 for each study) generated by ChatGPT-4o and GitHub Copilot.

This error rate is lower than those reported in studies of real applications and application templates (MUNIZ *et al.*, 2024; VENDOME *et al.*, 2019) (e.g., 7.61 errors per screen in (MUNIZ *et al.*, 2024)). However, it is important to note that the screens required in this study contained fewer elements than those in real applications, suggesting that LLM-generated code still contains accessibility issues that cannot be overlooked.

In Study 1, the number of errors varied depending on the layout types used, underscoring how the choice of framework can directly influence accessibility quality. Notably, the Jetpack Compose framework stood out, as the prompts requesting this layout resulted in fewer accessibility errors compared to other layouts. This result may indicate that the more modern framework offers advantages in terms of accessibility. It may suggest that the codes using Jetpack Compose, which were used in the training of the LLMs, have accessibility features, such as text or image descriptions, more frequently than the other layout models. However, the number of errors remains significant, emphasizing the need for developers to carefully validate the accessibility of LLM-generated interfaces.

In Study 2, ChatGPT-4o and Sabiá-3 generated more errors than ChatGPT-3.5, indicating that the latest version of GPT still faces challenges in addressing accessibility requirements. In terms of error types, inadequate contrast was the most common issue, followed by problems with element descriptions. These findings align with other studies (VENDOME *et al.*, 2019; ALSHAYBAN *et al.*, 2020), which show that these errors are persistent issues in mobile applications.

In Study 3, with English prompts, ChatGPT-4o generated considerably fewer errors and mean errors compared to the other studies, and especially compared to Study 2. This finding is consistent with the conclusions of other studies (WANG *et al.*, 2024a; KOYANAGI *et al.*, 2024) that code generation tasks are often under-performed when instructions are given in languages other than English.

In Study 4, GitHub Copilot (GPT-4o) showed a similar error rate to Studies 1 and 2, both with pt-BR prompt, and showed more errors than Study 3. This suggests that even though the model was trained to generate code, there was not much improvement in terms of accessibility.

7.3 RQ2 - What is the accessibility level of the screens generated by LLMs for native Android, when accessibility is required in the prompt?

For most prompts, screens generated when accessibility was explicitly requested (AccessReq) surprisingly had more errors than those generated without such a requirement (NReq). This suggests that even when instructed to prioritize accessibility, LLMs may misinterpret the guidelines or over-implement them, resulting in additional issues.

On several occasions of AccessReq, the LLMs pointed out generic accessibility issues before generate code, many of which were not present in the NReq code. At no point were the LLMs able to correctly identify the actual errors in the first generated code.

One possible explanation for this result lies in the quality and coverage of accessibility examples in the models' training data. If accessible UI code is underrepresented or inconsistently implemented across the datasets used to train these models, the LLM may rely on flawed or incomplete patterns when attempting to fulfill accessibility-related requests. This highlights a potential gap in the training data regarding accessibility best practices and suggests a broader need for curating higher-quality, accessibility-aware code examples in future LLM training pipelines.

It is clear that models require a deeper understanding of accessibility guidelines and how they apply to the context of each interface element, avoiding both over- and under-implementation. A common example of over-implementation in our study was the case where elements were given unnecessary accessible descriptions that confused screen readers, such as placing a "contentDescription" in an editable text field. This result suggests that LLMs still need refinement, especially when it comes to specific instructions related to accessibility.

7.4 RQ3 - Are the elements of the codes generated by the LLMs in the requested language?

The use of the prompt language Brazilian Portuguese (pt-BR) revealed that the descriptive elements in the generated code were predominantly in the intended language. However, there were instances where the translation of terms and descriptions deviated from the requested language.

We found 180 "Context" errors in three studies with prompt pt-BR, of 655 in total, representing approximately 27% of the total errors identified. In Study 2, a significant portion of the errors on the "MUSIC PLAYER" screen, in ChatGPT-4o, were related to "Context", with a large number of English terms (e.g., Song 1, Artist Name, Previous, Next). A similar pattern was observed in Study 4, where the "Context" category exhibited the highest number of errors in both studies.

The use of the English prompt also presented a "Context" issue, albeit in a significantly smaller number. A total of three errors were documented on the "To-Do List" screen, attributed to the use of the Portuguese Brazilian (pt-BR) format for the due date. The potential causes of this error include IP issues, cookies, or caching in the browser in a language other than English.

This raises questions on the LLMs ability to handle linguistic nuances when instructions are given in specific languages. Linguistic accuracy is key to ensure that developers in different regions can use the generated code without the need for significant adjustments. This aligns with the literature (KOYANAGI *et al.*, 2024), in which the authors observed differences in the GitHub Copilot output when prompts to solve AtCoder contests were given in Chinese, Japanese, or English. Thus, the training of LLMs can be improved to make them effective in multilingual contexts, especially in less prevalent languages in training datasets.

7.5 RQ4 - Is there a variation in accessibility levels when repeating the same prompt?

In all studies, differences in the code suggested by the same prompt were observed in all three attempts. For example, as seen in Figures 16, 21, 25, and 28, there was variation in accessibility presented between the screens on the three identical prompts that were repeated for each of the seven screens in the three LLMs. In some cases, the difference in the number of errors is minimal, but in others, it ranged from zero to seven.

This also highlights that reproducibility and repeatability are still not easily achievable features for code generation in the LLMs tested. The variation in results suggests that the models may interpret the same prompt differently depending on internal factors, such as randomization or subtle differences in their training. This raises concerns about the reliability of LLMs in contexts where consistency and accuracy are essential, such as in software development environments, where even a small coding error can have significant consequences.

These fluctuations in results emphasize the need for improvements in training algorithms and LLMs architecture to reduce discrepancies and ensure greater stability and predictability in the generated code. Additionally, it suggests that users of these models should adopt a critical approach when using them for tasks that require precision, especially when expecting the generated code to be reproducible and error-free.

7.6 Implications and takeaways

Our results have significant implications for the development of accessible mobile applications. The first major takeaway is that while LLM can accelerate development and assist in the creation of interfaces, **LLMs are not yet capable of replacing specialized accessibility knowledge**. Over-reliance on these models risks producing interface code that is not fully compliant with current accessibility guidelines and that does not adequately address the needs of people with disabilities.

Confirming the findings of the literature, we found that **an LLM tailored to a specific language can generate code with better results with prompts not written in English**. This is important for developers who are not English speakers.

The research also showed that the **code generated for Jetpack Compose has a greater potential to ensure proper accessibility**, but still requires adjustments. Another important implication is **the need to improve the prompt clarity and precision**. This can help guide LLMs to produce better results, particularly in addressing complex requirements such as accessibility. Developers should experiment with different prompt styles (e.g., zero-shot, one-shot, or few-shot) (SASAKI *et al.*, 2024) to identify the most effective approach for their specific use cases.

The results showed a **similar rate of accessibility errors** on screens generated **by both ChatGPT and GitHub Copilot**. Despite differences in interaction style and the underlying model architecture, both tools produced outputs with comparable levels of accessibility issues. This suggests that current LLM-based coding assistants, regardless of interface, may share similar limitations when it comes to generating accessible mobile user interfaces.

Analysis of the results of our study raises questions about the effectiveness of trained code generation models in terms of accessibility. Despite the expectation that these models could drive the creation of more accessible interfaces, our results show that such models did not lead to significant improvements in the accessibility of generated mobile screens.

The findings of this research also indicate that **accessibility is not yet a priority for LLMs** when generating code for mobile interfaces. Consequently, developers who use tools such as ChatGPT or GitHub Copilot are advised to employ complementary strategies to ensure the accessibility of their applications. These strategies may include the use of tools such as the Accessibility Scanner for code validation.

7.7 Threats to Validity

This section delineates the threats to the validity of this research. These threats are classified according to the study of Lima *et al.* (2014) and are divided into three categories: Construct, External, and Internal Validity.

7.7.1 Construct Validity

The evaluation centered on accessibility compliance, a pivotal component of usability. However, this emphasis might potentially overlook other crucial factors, such as navigation efficiency and the clarity of information, thereby constraining the experiment's validity. A more comprehensive analysis that integrates accessibility with other user experience components would facilitate a more detailed examination of the subject and increase the credibility of the results. Additionally, the absence of evaluation components such as integrations with assistive technologies (e.g., external keyboards and mice) and Flash may have led to the undetection of additional accessibility errors.

7.7.2 External Validity

One limitation of this research is the exclusive use of zero-shot prompts. While zero-shot prompting is valuable for evaluating the model's ability to generalize with minimal input, it does not reflect the broader range of prompting techniques, such as one-shot or few-shot prompting, commonly used in real-world applications. This restriction may limit the generalizability of the findings, as models might exhibit different performances when provided with examples or more structured contexts. Future studies should consider exploring a variety of prompting strategies to better assess model behavior across diverse contexts.

Additionally, the evaluation was conducted on only one type of device. This restriction may introduce bias, as different devices may possess disparate hardware and software configurations that could influence the performance or interaction with the system under study.

Our screen selection process also presents a limitation. While the choices were informed by an analysis of common screens in popular mobile applications and enriched by informal conversations with developers and designers, they were not derived from a formal or systematic study. As a result, assumptions regarding which screens are most relevant or interesting may not be fully generalizable to the broader developer community.

Finally, the simplicity of the interfaces used in this study. The screens selected for evaluation represent relatively common and structurally simple mobile interfaces, which may not capture the full complexity typically found in real-world applications.

7.7.3 Internal Validity

This study may encounter challenges in terms of internal validity due to the stochastic nature of the LLM output, which can exhibit variability for the same prompt. To address this challenge, each prompt was executed multiple times, and the variability of the results was subjected to a thorough analysis. Additionally, the utilization of a singular evaluation instrument, the Accessibility Scanner, might introduce bias, as it is conceivable that certain accessibility concerns might not be detected. To ensure a more comprehensive evaluation, future research should incorporate multiple tools and manual validation.

Another factor that could influence results is consistency in prompt execution. Variations in phrasing, model configurations, or external factors such as computational load could confound the findings. To mitigate these potential risks, the study standardized prompt structures and testing conditions.

7.8 Chapter Synthesis

In summary, the points discussed in this chapter highlight both the potential and the current limitations of LLMs in generating accessible mobile interfaces. The analyzes revealed significant challenges related to understanding accessibility guidelines, variations in performance depending on the language of the prompt, inconsistencies in responses to identical prompts, and the impact of the type of layout used. Although models, such as ChatGPT-4o, show progress in code generation, they are not yet mature enough to reliably guarantee accessibility. The conclusions drawn here reinforce the need for more careful approaches, improvements in the models themselves, and greater attention to the ways in which people interact with these tools.

8 CONCLUSION

This chapter presents the conclusion of this research. Section 8.1 shows our final considerations. Section 8.2 highlights the main contributions of this work. Section 8.3 shows the publications and datasets related to this work. Finally, Section 8.4 presents proposals for further research.

8.1 Final Considerations

This research sought to evaluate the accessibility of user interfaces for LLM-generated mobile applications, with a focus on native Android development. In four structured studies, the investigation focused on the influence of different LLMs, layout paradigms, languages, and prompt formulations on the accessibility of generated screens.

The findings of this study indicated that accessibility continues to pose a substantial challenge in the realm of LLM-generated interfaces. The analysis identified a total of 702 accessibility issues on 288 evaluated screens, with an average of 2.44 errors per screen.

Despite the increasing use of LLMs in mobile app development, the average number of accessibility issues per screen was substantial in all models evaluated, including ChatGPT-3.5, ChatGPT-4o, Sabiá-3, and GitHub Copilot. Of particular interest is the observation that, in instances where accessibility was explicitly requested in the prompt, the anticipated enhancements were not evident. In some cases, the number of accessibility issues increased, suggesting that current models may still lack a comprehensive understanding of accessibility best practices.

Among the key findings, Jetpack Compose emerged as the layout framework that resulted in fewer accessibility errors, reinforcing its potential as a more inclusive development approach. Furthermore, screens generated from prompts in English exhibited a substantially lower incidence of errors compared to those generated from prompts in Brazilian Portuguese. This observation is consistent with previous research that has documented language biases present in LLMs.

The categories “Contrast” and “Context” were the most prevalent accessibility issues, underscoring the need for models to improve their ability to manage visual design elements and linguistic consistency. The limited impact of explicitly requesting for accessibility on the prompts underscores a gap in current LLM capabilities that developers and model providers must address.

8.2 Main Contributions

The results of this research highlight the need for advances in the automatic generation of accessible code. Despite the potential of language models to accelerate the development of mobile interfaces, the data obtained indicate that these tools still have significant limitations when it comes to creating accessible components. The error analysis revealed recurring problems with contrast, lack of appropriate descriptions of interactive elements, and contextual errors. In addition, accessibility prompts did not always lead to improvements, suggesting that LLMs still lack a refined understanding of this aspect.

Among the most relevant findings was the fact that the choice of framework used to build the interface has a direct impact on the accessibility of the screens generated. Jetpack Compose performed better than XML, suggesting that advances in development tools can help improve the accessibility of generated code. In addition, the comparison between different LLMs showed variations in the number of errors, with ChatGPT-4o (pt-BR prompt) being the model with the highest number of problems, while Sabiá-3 had a relatively more stable performance. These results point to the need for improvements in the training of these AIs, including more rigorous databases aligned with accessibility guidelines such as WCAG.

Also, the language of the prompt may also have a significant impact on LLM performance. Notably, the study using an English prompt reported both the lowest number of errors and the lowest error rate compared to studies using prompts in other languages and in different LLMs. This suggests, like other studies (WANG *et al.*, 2024a; KOYANAGI *et al.*, 2024), that LLMs may be more optimized or better trained for English input, potentially due to the predominant English language data in their training corpora.

Based on these findings, this research suggests that developers using LLMs for code generation should not blindly trust the answers they receive. A manual review process and the use of validation tools such as Google Accessibility Scanner are essential to ensure that the resulting applications are accessible to all users. In addition, refining prompt design to include one-shot and few-shot approaches that provide richer context and examples could allow for better error correction and guidance on how to create accessible application interfaces. In addition, formulating more detailed prompts, including specific requirements for accessible elements, can possibly minimize errors and make the generated code more suitable for different audiences.

8.3 Publications and datasets

During the master's program, I participated in additional research projects related to mobile accessibility, which resulted in published papers at academic conferences and the creation of publicly available datasets.

One of these studies was published in the Proceedings of the 30th Brazilian Symposium on Multimedia and the Web (WebMedia 2024)[1]. In this work, Andrade *et al.* (2024) evaluated the accessibility of eight popular Android applications performing various tasks in three languages: Portuguese, English, and Spanish. The study reported a total of 2,355 accessibility issues, the highest of which was found in the Spanish tests (828). To enable replication and foster future research, an open dataset was created compiling the accessibility issues identified across all applications. The dataset is available on GitHub[2].

Another study, presented at the 17th International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '24)[3], by Muniz *et al.* (2024), involved the development of mobile applications based on ten widely used Figma templates. Accessibility was assessed in these apps and 738 issues were identified, with an average of 7.61 errors per screen. All templates showed some level of accessibility non-compliance.

Additionally, part of the present research was accepted for publication at the 12th International Conference on Mobile Software Engineering and Systems (MOBILESoft 2025)[4]. This paper explores the use of LLMs in the context of mobile accessibility. We provide a complete replication infrastructure, including all prompts, application project files, and test reports generated using the Accessibility Scanner tool. The dataset is available on GitHub[5].

These publications and their associated datasets not only support the findings presented in this dissertation but also contribute to the broader research community. All datasets are available upon request and are intended to be released as open-access resources to support ongoing and future work on AI-driven digital accessibility.

[1] https://sol.sbc.org.br/index.php/webmedia/article/view/30340
[2] https://github.com/Test-Accessibility/Dataset-accessibility-assessments/tree/main/Study%2002
[3] https://doi.org/10.1145/3652037.3652075
[4] https://www.researchgate.net/publication/390235970_Breaking_Barriers_in_Mobile_Accessibility_A_Study_of_LLM-Generated_Native_Android_Interfaces
[5] https://github.com/Test-Accessibility/Dataset-accessibility-assessments/tree/main/Study_LLMs

8.4 Future work

Given the increasing role of AI in software development, accessibility must be treated as a core component rather than as an afterthought. One promising area of research is the creation of automated metrics that allow LLMs to evaluate and self-correct their output before delivering the final code. This kind of feedback loop could dramatically improve code quality and reduce developer intervention.

Furthermore, we propose the development and training of a language model specialized in digital accessibility. This would involve curating a training dataset composed of mobile accessibility guidelines, inclusive design patterns, and annotated examples of both compliant and non-compliant UI code. Such a model could assist developers in real time by flagging accessibility issues, suggesting improvements, and even automatically generating accessible alternatives.

Future research should focus on improving prompt design and improving error correction within real-world mobile app interfaces. This includes the development of more intuitive and context-sensitive prompts that can effectively guide users through tasks and interactions. The incorporation of one-shot and few-shot prompting techniques, which provide richer context and concrete examples, holds significant potential in enhancing the generation of accessible user interfaces and the correction of accessibility-related errors.

In addition, accessibility evaluation is essential to expand accessibility evaluation to other languages and frameworks widely used in application development, such as React Native and Flutter. Future research should consider how LLMs can be adapted to understand and generate accessible components in other languages, respecting platform-specific guidelines and ensuring that good practices are maintained in different contexts of use.

By ensuring that AI tools evolve in this direction, it will be possible to promote more inclusive technological development, benefiting millions of users with disabilities around the world.

REFERENCES

AL-AZANI, S.; ALTURAYEIF, N.; ABOUELRESH, H.; ALHUNIEF, A. A comprehensive framework and empirical analysis for evaluating large language models in arabic dialect identification. In: **2024 International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2024. p. 1–7.

ALJEDAANI, W.; HABIB, A.; ALJOHANI, A.; ELER, M.; FENG, Y. Does ChatGPT generate accessible code? investigating accessibility challenges in llm-generated source code. In: **Proceedings of the 21st International Web for All Conference**. New York, NY, USA: Association for Computing Machinery, 2024. (W4A '24), p. 165–176. ISBN 9798400710308. Available at: https://doi.org/10.1145/3677846.3677854. Accessed on: 9 June 2025.

ALSHAYBAN, A.; AHMED, I.; MALEK, S. Accessibility issues in android apps: state of affairs, sentiments, and ways forward. In: **Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2020. (ICSE '20), p. 1323–1334. ISBN 9781450371216. Available at: https://doi.org/10.1145/3377811.3380392. Accessed on: 9 June 2025.

ANDRADE, M. G. de; RABELO, D. M. F.; SOUZA, R. M. de; VIANA, W. 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**. Porto Alegre, RS, Brasil: SBC, 2024. p. 400–404. ISSN 0000-0000. Available at: https://sol.sbc.org.br/index.php/webmedia/article/view/30340. Accessed on: 9 June 2025.

BALLANTYNE, M.; JHA, A.; JACOBSEN, A.; HAWKER, J. S.; EL-GLALY, Y. N. Study of accessibility guidelines of mobile applications. In: **Proceedings of the 17th international conference on mobile and ubiquitous multimedia**. [S.l.: s.n.], 2018. p. 305–315.

BRETON, J.; BILLAMI, M. B.; CHEVALIER, M.; TROJAHN, C. Empowering camembert legal entity extraction with llm boostrapping. In: **International Conference on Knowledge Engineering and Knowledge Management**. [S.l.]: Springer, 2024. p. 86–101.

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.; AMODEI, D. **Language Models are Few-Shot Learners**. 2020. Available at: https://arxiv.org/abs/2005.14165. Accessed on: 9 June 2025.

CHEN, B.; ZHANG, Z.; LANGRENÉ, N.; ZHU, S. Unleashing the potential of prompt engineering for large language models. **Patterns**, Elsevier BV, p. 101260, may 2025. ISSN 2666-3899. Available at: http://dx.doi.org/10.1016/j.patter.2025.101260. Accessed on: 9 June 2025.

CHEN, M.; TWOREK, J.; JUN, H.; YUAN, Q.; PINTO, H. P. de O.; 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.; ZAREMBA, W. **Evaluating Large Language Models Trained on Code**. 2021. Available at: https://arxiv.org/abs/2107.03374. Accessed on: 9 June 2025.

CHEN, S.; CHEN, C.; FAN, L.; FAN, M.; ZHAN, X.; LIU, Y. Accessible or not? an empirical investigation of android app accessibility. **IEEE Transactions on Software Engineering**, v. 48, n. 10, p. 3954–3968, 2022.

DELNEVO, G.; ANDRUCCIOLI, M.; MIRRI, S. On the interaction with large language models for web accessibility: Implications and challenges. In: **2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)**. [S.l.: s.n.], 2024. p. 1–6.

DILLMANN, M.; SIEBERT, J.; TRENDOWICZ, A. **Evaluation of large language models for assessing code maintainability**. 2024. Available at: https://arxiv.org/abs/2401.12714. Accessed on: 9 June 2025.

ELER, M. M.; ROJAS, J. M.; GE, Y.; FRASER, G. Automated accessibility testing of mobile apps. In: **2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.: s.n.], 2018. p. 116–126.

FAN, Z.; GAO, X.; MIRCHEV, M.; ROYCHOUDHURY, A.; TAN, S. H. Automated repair of programs from large language models. In: **Proceedings of the 45th International Conference on Software Engineering**. IEEE Press, 2023. (ICSE '23), p. 1469–1481. ISBN 9781665457019. Available at: https://doi.org/10.1109/ICSE48619.2023.00128. Accessed on: 9 June 2025.

GAO, C.; HU, X.; GAO, S.; XIA, X.; JIN, Z. The current challenges of software engineering in the era of large language models. **ACM Trans. Softw. Eng. Methodol.**, Association for Computing Machinery, New York, NY, USA, Jan. 2025. ISSN 1049-331X. Available at: https://doi.org/10.1145/3712005. Accessed on: 9 June 2025.

GOTTLANDER, J.; KHADEMI, T. **The Effects of AI Assisted Programming in Software Engineering**: An observation of github copilot in the industry. Master's Thesis (Master's Thesis) — University of Gothenburg, 2023.

HELLAS, A.; LEINONEN, J.; SARSA, S.; KOUTCHEME, C.; KUJANPÄÄ, L.; SORVA, J. Exploring the responses of large language models to beginner programmers' help requests. In: **Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1**. [S.l.: s.n.], 2023. p. 93–105.

HOU, X.; ZHAO, Y.; LIU, Y.; YANG, Z.; WANG, K.; LI, L.; LUO, X.; LO, D.; GRUNDY, J.; WANG, H. Large language models for software engineering: A systematic literature review. **ACM Trans. Softw. Eng. Methodol.**, Association for Computing Machinery, New York, NY, USA, v. 33, n. 8, Dec. 2024. ISSN 1049-331X. Available at: https://doi.org/10.1145/3695988. Accessed on: 9 June 2025.

HU, D.; SUN, G.; LIU, L.; LIU, C.; WANG, D. Evaluation of ancient chinese natural language understanding in large language models based on achnlu. In: **Wisdom, Well-Being, Win-Win**. Cham: Springer Nature Switzerland, 2024. p. 3–18. ISBN 978-3-031-57860-1.

JORDAN, M.; LY, K.; RAJ, A. G. S. Need a programming exercise generated in your native language? chatgpt's got your back: Automatic generation of non-english programming exercises using openai gpt-3.5. In: **Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1**. New York, NY, USA: Association for Computing Machinery, 2024. (SIGCSE 2024), p. 618–624. ISBN 9798400704239. Available at: https://doi.org/10.1145/3626252.3630897. Accessed on: 9 June 2025.

KAMESWARAN, V.; MURALIDHAR, S. H. Cash, digital payments and accessibility: A case study from metropolitan india. **Proceedings of the ACM on human-computer interaction**, ACM New York, NY, USA, v. 3, n. CSCW, p. 1–23, 2019.

KOYANAGI, K.; WANG, D.; NOGUCHI, K.; KONDO, M.; SEREBRENIK, A.; KAMEI, Y.; UBAYASHI, N. Exploring the effect of multiple natural languages on code suggestion using github copilot. In: IEEE. **2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)**. [S.l.], 2024. p. 481–486.

LEITE, M. V. R.; SCATALON, L. P.; FREIRE, A. P.; ELER, M. M. Accessibility in the mobile development industry in Brazil: Awareness, knowledge, adoption, motivations and barriers. **Journal of Systems and Software**, v. 177, p. 110942, 2021. ISSN 0164-1212. Available at: https://www.sciencedirect.com/science/article/pii/S016412122100039X. Accessed on: 9 June 2025.

LI, Y.; CHOI, D.; CHUNG, J.; KUSHMAN, N.; SCHRITTWIESER, J.; LEBLOND, R.; ECCLES, T.; KEELING, J.; GIMENO, F.; LAGO, A. D.; HUBERT, T.; CHOY, P.; D'AUTUME, C. de M.; BABUSCHKIN, I.; CHEN, X.; HUANG, P.-S.; WELBL, J.; GOWAL, S.; CHEREPANOV, A.; MOLLOY, J.; MANKOWITZ, D. J.; ROBSON, E. S.; KOHLI, P.; FREITAS, N. de; KAVUKCUOGLU, K.; VINYALS, O. Competition-level code generation with alphacode. **Science**, American Association for the Advancement of Science (AAAS), v. 378, n. 6624, p. 1092–1097, Dec. 2022. ISSN 1095-9203. Available at: http://dx.doi.org/10.1126/science.abq1158. Accessed on: 9 June 2025.

LI, Y.; SHI, J.; ZHANG, Z. An approach for rapid source code development based on chatgpt and prompt engineering. **IEEE Access**, v. 12, p. 53074–53087, 2024.

LIMA, V. C.; NETO, A. G. S.; EMER, M. C. Experimental investigation and agile practices: Threats to the validity of experiments involving the pair programming agile practice. **Revista Eletrônica de Sistemas de Informação<BR>Information Systems Electronic Journal<BR>ISSN16773071 doi:10.21529/RESI**, v. 13, n. 1, 2014. ISSN 1677-3071. Available at: https://periodicosibepes.org.br/index.php/reinfo/article/view/1500. Accessed on: 9 June 2025.

LIU, Y.; LE-CONG, T.; WIDYASARI, R.; TANTITHAMTHAVORN, C.; LI, L.; LE, X.-B. D.; LO, D. Refining ChatGPT-generated code: Characterizing and mitigating code quality issues. **ACM Trans. Softw. Eng. Methodol.**, Association for Computing Machinery, New York, NY, USA, v. 33, n. 5, jun 2024. ISSN 1049-331X. Available at: https://doi.org/10.1145/3643674. Accessed on: 9 June 2025.

LOPES, R.; FAÇANHA, A.; VIANA, W. I can't pay! accessibility analysis of mobile banking apps. In: **XXVIII Brazilian Symposium on Multimedia and Web (WebMedia)**. Porto Alegre, RS, Brasil: SBC, 2022. p. 269–273. Available at: https://sol.sbc.org.br/index.php/webmedia/article/view/22118. Accessed on: 9 June 2025.

MARITACAAI. **Sabia-3**. 2024. Available at: https://chat.maritaca.ai/. Accessed on: 9 September 2024.

MARTINS, R. S.; RABELO, D. M.; ARAÚJO, M. d. C. C.; CARVALHO, W. V. de. Where is the description? investigating accessibility issues in portuguese versions of smart home apps. In: **Proceedings of the 21st Brazilian Symposium on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2022. (IHC '22). ISBN 9781450395069. Available at: https://doi.org/10.1145/3554364.3559133. Accessed on: 9 June 2025.

MASTROPAOLO, A.; PASCARELLA, L.; GUGLIELMI, E.; CINISELLI, M.; SCALABRINO, S.; OLIVETO, R.; BAVOTA, G. On the robustness of code generation techniques: An empirical study on github copilot. In: **Proceedings of the 45th International Conference on Software Engineering**. IEEE Press, 2023. (ICSE '23), p. 2149–2160. ISBN 9781665457019. Available at: https://doi.org/10.1109/ICSE48619.2023.00181. Accessed on: 9 June 2025.

MOWAR, P.; PENG, Y.-H.; WU, J.; STEINFELD, A.; BIGHAM, J. P. **CodeA11y: Making AI Coding Assistants Useful for Accessible Web Development**. 2025. Available at: https://arxiv.org/abs/2502.10884. Accessed on: 9 June 2025.

MUNIZ, J. H.; RABELO, D. M. F.; VIANA, W. Assessing accessibility levels in mobile applications developed from figma templates. In: **Proceedings of the 17th International Conference on PErvasive Technologies Related to Assistive Environments**. New York, NY, USA: Association for Computing Machinery, 2024. (PETRA '24), p. 316–321. ISBN 9798400717604. Available at: https://doi.org/10.1145/3652037.3652075. Accessed on: 9 June 2025.

OLIVEIRA, A. D. A.; SANTOS, P. S. H. D.; JúNIOR, W. E. M.; ALJEDAANI, W. M.; ELER, D. M.; ELER, M. M. Analyzing accessibility reviews associated with visual disabilities or eye conditions. In: **Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2023. (CHI '23). ISBN 9781450394215. Available at: https://doi.org/10.1145/3544548.3581315. Accessed on: 9 June 2025.

OLIVEIRA, G. A. A. de; BETTIO, R. W. de; FREIRE, A. P. Accessibility of the smart home for users with visual disabilities: an evaluation of open source mobile applications for home automation. In: **Proceedings of the 15th Brazilian Symposium on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2016. (IHC '16). ISBN 9781450352352. Available at: https://doi.org/10.1145/3033701.3033730. Accessed on: 9 June 2025.

OPENAI. **GPT-3**. 2020. Available at: https://www.openai.com/research/gpt-3. Accessed on: 9 June 2025.

OPENAI. **GPT-4**. 2023. Available at: https://openai.com/index/hello-gpt-4o/. Accessed on: 9 June 2025.

OPENAI. **ChatGPT**. 2024. Available at: https://chat.openai.com/. Accessed on: 9 June 2025.

OPPENLAENDER, J.; LINDER, R.; SILVENNOINEN, J. **Prompting AI Art: An Investigation into the Creative Skill of Prompt Engineering**. 2024. Available at: https://arxiv.org/abs/2303.13534. Accessed on: 9 June 2025.

OTHMAN, A.; DHOUIB, A.; JABOR, A. N. A. 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**. New York, NY, USA: Association for Computing Machinery, 2023. (PETRA ’23), p. 707–713. ISBN 9798400700699. Available at: https://doi-org.ez11.periodicos.capes.gov.br/10.1145/3594806.3596542. Accessed on: 9 June 2025.

PENG, Q.; CHAI, Y.; LI, X. **HumanEval-XL: A Multilingual Code Generation Benchmark for Cross-lingual Natural Language Generalization**. 2024. Available at: https://arxiv.org/abs/2402.16694. Accessed on: 9 June 2025.

PENG, S.; KALLIAMVAKOU, E.; CIHON, P.; DEMIRER, M. **The Impact of AI on Developer Productivity: Evidence from GitHub Copilot**. 2023. Available at: https://arxiv.org/abs/2302.06590. Accessed on: 9 June 2025.

PEREIRA, L. S.; MATOS, M.; DUARTE, C. Exploring mobile device accessibility: Challenges, insights, and recommendations for evaluation methodologies. In: **Proceedings of the CHI Conference on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2024. (CHI ’24). ISBN 9798400703300. Available at: https://doi.org/10.1145/3613904.3642526. Accessed on: 9 June 2025.

PEREZ, F.; RIBEIRO, I. **Ignore Previous Prompt: Attack Techniques For Language Models**. 2022. Available at: https://arxiv.org/abs/2211.09527. Accessed on: 9 June 2025.

PIRES, R.; ABONIZIO, H.; ALMEIDA, T. S.; NOGUEIRA, R. Sabiá: Portuguese large language models. In: SPRINGER. **Brazilian Conference on Intelligent Systems**. [S.l.], 2023. p. 226–240.

REYNOLDS, L.; MCDONELL, K. **Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm**. 2021. Available at: https://arxiv.org/abs/2102.07350. Accessed on: 9 June 2025.

SAHOO, P.; SINGH, A. K.; SAHA, S.; JAIN, V.; MONDAL, S.; CHADHA, A. **A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications**. 2025. Available at: https://arxiv.org/abs/2402.07927. Accessed on: 9 June 2025.

SANTOS, T.; BRANCO, J. C. Percepções e desafios da acessibilidade digital: um estudo sobre conhecimento e aplicação de diretrizes acessíveis em projetos de desenvolvimento de artefatos digitais no brasil. **Blucher Design Proceedings**, v. 12, n. 1, p. 1263–1275, 2024. ISSN 2318-6968.

SASAKI, Y.; WASHIZAKI, H.; LI, J.; SANDER, D.; YOSHIOKA, N.; FUKAZAWA, Y. Systematic literature review of prompt engineering patterns in software engineering. In: **2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)**. [S.l.: s.n.], 2024. p. 670–675.

SERRA, L. C.; CARVALHO, L. P.; FERREIRA, L. P.; VAZ, J. B. S.; FREIRE, A. P. Accessibility evaluation of e-government mobile applications in brazil. **Procedia Computer Science**, v. 67, p. 348–357, 2015. ISSN 1877-0509. Available at: https://www.sciencedirect.com/science/article/pii/S1877050915031257. Accessed on: 9 June 2025.

SIDDIQ, M. L.; MAJUMDER, S. H.; MIM, M. R.; JAJODIA, S.; SANTOS, J. C. S. An empirical study of code smells in transformer-based code generation techniques. **2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)**, p. 71–82, 2022. Available at: https://api.semanticscholar.org/CorpusID:252125574. Accessed on: 9 June 2025.

SPIESS, C.; GROS, D.; PAI, K. S.; PRADEL, M.; RABIN, M. R. I.; ALIPOUR, A.; JHA, S.; DEVANBU, P.; AHMED, T. **Calibration and Correctness of Language Models for Code**. 2024. Available at: https://arxiv.org/abs/2402.02047. Accessed on: 9 June 2025.

STUDENT. The probable error of a mean. **Biometrika**, [Oxford University Press, Biometrika Trust], v. 6, n. 1, p. 1–25, 1908. ISSN 00063444, 14643510. Available at: http://www.jstor.org/stable/2331554. Accessed on: 9 June 2025.

SUH, H.; TAFRESHIPOUR, M.; MALEK, S.; AHMED, I. **Human or LLM? A Comparative Study on Accessible Code Generation Capability**. 2025. Available at: https://arxiv.org/abs/2503.15885. Accessed on: 9 June 2025.

SVYATKOVSKIY, A.; DENG, S. K.; FU, S.; SUNDARESAN, N. **IntelliCode Compose: Code Generation Using Transformer**. 2020. Available at: https://arxiv.org/abs/2005.08025. Accessed on: 9 June 2025.

SWEARNGIN, A.; WU, J.; ZHANG, X.; GOMEZ, E.; COUGHENOUR, J.; STUKENBORG, R.; GARG, B.; HUGHES, G.; HILLIARD, A.; BIGHAM, J. P.; NICHOLS, J. Towards automated accessibility report generation for mobile apps. **ACM Trans. Comput.-Hum. Interact.**, Association for Computing Machinery, New York, NY, USA, v. 31, n. 4, Sep. 2024. ISSN 1073-0516. Available at: https://doi.org/10.1145/3674967. Accessed on: 9 June 2025.

TANG, N.; CHEN, M.; NING, Z.; BANSAL, A.; HUANG, Y.; MCMILLAN, C.; LI, T. J.-J. **A Study on Developer Behaviors for Validating and Repairing LLM-Generated Code Using Eye Tracking and IDE Actions**. 2024. Available at: https://arxiv.org/abs/2405.16081. Accessed on: 9 June 2025.

TAZI, F.; SAKA, S.; OPP, G.; NEUPANE, S.; DAS, S.; CARLI, L. D.; RAY, I. Accessibility evaluation of IoT android mobile companion apps. In: **Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2023. (CHI EA '23). ISBN 9781450394222. Available at: https://doi.org/10.1145/3544549.3585652. Accessed on: 9 June 2025.

TIAN, H.; LU, W.; LI, T. O.; TANG, X.; CHEUNG, S.-C.; KLEIN, J.; BISSYANDÉ, T. F. **Is ChatGPT the Ultimate Programming Assistant – How far is it?** 2023. Available at: https://arxiv.org/abs/2304.11938. Accessed on: 9 June 2025.

TONMOY, S. M. T. I.; ZAMAN, S. M. M.; JAIN, V.; RANI, A.; RAWTE, V.; CHADHA, A.; DAS, A. **A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models**. 2024. Available at: https://arxiv.org/abs/2401.01313. Accessed on: 9 June 2025.

U.S. Congress. **The Americans with Disabilities Act of 1990**. 1990. Available at: https://www.ada.gov/. Accessed on: 9 June 2025.

VAITHILINGAM, P.; ZHANG, T.; GLASSMAN, E. L. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In: **Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems**. New York, NY, USA: Association for Computing Machinery, 2022. (CHI EA '22). ISBN 9781450391566. Available at: https://doi.org/10.1145/3491101.3519665. Accessed on: 9 June 2025.

VASILINIUC, M.-S.; GROZA, A. Case study: using ai-assisted code generation in mobile teams. In: **2023 IEEE 19th International Conference on Intelligent Computer Communication and Processing (ICCP)**. [S.l.: s.n.], 2023. p. 339–346.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. **Attention Is All You Need**. 2023. Available at: https://arxiv.org/abs/1706.03762. Accessed on: 9 June 2025.

VENDOME, C.; SOLANO, D.; LIÑÁN, S.; LINARES-VÁSQUEZ, M. Can everyone use my app? an empirical study on accessibility in android apps. In: **2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. [S.l.: s.n.], 2019. p. 41–52.

W3C. **Web Content Accessibility Guidelines (WCAG) 2.1**. World Wide Web Consortium, 2023. Available at: https://www.w3.org/TR/WCAG21. Accessed on: 9 June 2025.

WANG, C.; LI, Z.; GAO, C.; WANG, W.; PENG, T.; HUANG, H.; DENG, Y.; WANG, S.; LYU, M. R. **Exploring Multi-Lingual Bias of Large Code Models in Code Generation**. 2024. Available at: https://arxiv.org/abs/2404.19368. Accessed on: 9 June 2025.

WANG, J.; HUANG, Y.; CHEN, C.; LIU, Z.; WANG, S.; WANG, Q. Software testing with large language models: Survey, landscape, and vision. **IEEE Transactions on Software Engineering**, v. 50, n. 4, p. 911–936, 2024. Available at: https://doi.org/10.1109/TSE.2024.3368208. Accessed on: 9 June 2025.

WANG, W.; NING, H.; QIAN, S.; ZHANG, G.; WANG, Y. Characterizing developers' behaviors in llm -supported software development. In: **2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)**. [S.l.: s.n.], 2024. p. 1168–1177.

WEI, J.; WANG, X.; SCHUURMANS, D.; BOSMA, M.; ICHTER, B.; XIA, F.; CHI, E.; LE, Q.; ZHOU, D. **Chain-of-Thought Prompting Elicits Reasoning in Large Language Models**. 2023. Available at: https://arxiv.org/abs/2201.11903. Accessed on: 9 June 2025.

WHO *et al.* World report on vision. **World Health Organization (WHO)**, World Health Organization, 2019.

XU, Z.; PENG, K.; DING, L.; TAO, D.; LU, X. **Take Care of Your Prompt Bias! Investigating and Mitigating Prompt Bias in Factual Knowledge Extraction**. 2024. Available at: https://arxiv.org/abs/2403.09963. Accessed on: 9 June 2025.

ZAINA, L. A.; FORTES, R. P.; CASADEI, V.; NOZAKI, L. S.; PAIVA, D. M. B. Preventing accessibility barriers: Guidelines for using user interface design patterns in mobile applications. **Journal of Systems and Software**, v. 186, p. 111213, 2022. ISSN 0164-1212. Available at: https://www.sciencedirect.com/science/article/pii/S0164121221002831. Accessed on: 9 June 2025.

ZHANG, T.; XU, B.; THUNG, F.; HARYONO, S. A.; LO, D.; JIANG, L. Sentiment analysis for software engineering: How far can pre-trained transformer models go? In: **2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. [S.l.: s.n.], 2020. p. 70–80.

ZHONG, L.; WANG, Z. **Can ChatGPT replace StackOverflow? A Study on Robustness and Reliability of Large Language Model Code Generation**. 2024. Available at: https://arxiv.org/abs/2308.10335. Accessed on: 9 June 2025.