Image-to-UI Accessibility: Assessing ChatGPT's Effectiveness in Generating Accessible Android Screens from Figma Templates

Lucas Emanuel Oliveira Xavier
lucaseoxavier@gmail.com
Federal University of Ceará
Fortaleza, Ceará, Brazil

Daniel Mesquita Feijó Rabelo
dmesquita861@gmail.com
Federal University of Ceará
Fortaleza, Ceará, Brazil

Ribamar de Souza Martins
ribamarop@gmail.com
Federal University of Ceará
Fortaleza, Ceará, Brazil

Windson Viana
windson@virtual.ufc.br
Federal University of Ceará
Fortaleza, Ceará, Brazil

ABSTRACT

As accessibility remains a persistent challenge in mobile applications, recent advances in Large Language Models (LLMs) offer promising support for addressing these issues in the development process. This study investigates the potential of ChatGPT-4 to generate accessible Android user interfaces directly from screen mockups. Using a dataset of 60 screens in 8 Figma-based UI templates, we prompted ChatGPT to produce Jetpack Compose code solely from screen images. We then evaluated the generated code using Google's Accessibility Scanner, identifying a total of 302 accessibility violations. Through iterative prompting guided by the scanner's feedback, we achieved a reduction of more than 50% in the reported issues. However, the process also introduced new violations in some cases, highlighting important limitations in the consistency and reliability of the model. Our findings suggest that while LLMs like ChatGPT can support accessibility improvements in mobile User Interface (UI) development, human oversight remains essential. This study contributes to ongoing discussions on the role of AI in inclusive software design and the future of image-to-code generation pipelines.

KEYWORDS

Large Language Models, Mobile Accessibility, Jetpack Compose, Figma Templates, ChatGPT-4

1 INTRODUCTION

Advances in Artificial Intelligence (AI) have transformed how society interacts with technology. From speech interpretation to video generation, AI systems can learn from large datasets and emulate human decision making [10, 16, 28, 42]. In software development and multimedia creation, AI-powered tools are increasingly used to improve productivity by generating assets, code, hypermedia documents, and metadata [12, 14, 26, 42].

As AI reshapes development processes, its role in promoting accessibility is gaining attention. Digital accessibility aims to eliminate barriers and ensure equal access for all users, regardless of physical, perceptual, or social conditions [7]. According to the World Health Organization (WHO), 1.3 billion people live with disabilities and 2.2 billion experience visual impairments [47, 48]. Guidelines such as W3C's WCAG [50] and Google's Android Accessibility Guide [17] offer best practices to help developers design inclusive applications, including support for assistive technologies such as TalkBack[1].

Despite the availability of established accessibility guidelines for mobile and web development, ensuring that mobile and Web applications are accessible remains an ongoing challenge [15, 30, 32, 36, 44]. Accessibility is often overlooked during the development process, leading to widespread usability barriers for people with disabilities [4, 8, 27, 52]. These shortcomings highlight key issues: lack of awareness, limited tools, and poor integration of accessibility practices throughout the development lifecycle.

Modern User Interface (UI) frameworks, such as Jetpack Compose[2], aim to streamline mobile development through a declarative approach and built-in tools for accessibility, including semantic APIs and compatibility with TalkBack [19]. In parallel, Large Language Models (LLMs), such as OpenAI's GPT[3], have shown the potential to process natural language and generate structured code [29, 37, 39]. Given their ability to process multimodal input [2, 33, 35] and suggest code snippets [13], these models present a promising opportunity to support developers in creating accessible user interfaces from visual design artifacts, such as UI mockups.

However, despite this potential, the effectiveness of LLMs in generating accessible mobile interfaces directly from image-based input remains largely unexplored. To the best of our knowledge, no study has so far evaluated the accessibility of LLM-generated mobile application interfaces from image input. To address this gap in mobile software development, the main objective of this study is to evaluate the ability of ChatGPT-4 to generate accessible Android UI code in Jetpack Compose from image input.

The model was prompted to create screens based on eight Figma templates, and the generated output had its accessibility assessed using Google's Accessibility Scanner. The initial outputs contained 302 accessibility issues, including 172 low contrast texts and 71 item description errors. We then asked ChatGPT to fix the identified issues and re-ran the scanner, finding that the total number of issues was reduced by more than half. To verify the reliability of these

In: Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia'2025). Rio de Janeiro, Brazil. Porto Alegre: Brazilian Computer Society, 2025.

ISSN 2966-2753

[1] https://support.google.com/accessibility/android/answer/6283677
[2] https://developer.android.com/compose
[3] https://openai.com/index/gpt-4-research/

results, we then selected the nine screens with the most issues and re-ran the prompt twice. None of the retries produced identical results, but only one showed significant variation over the original.

The remainder of this paper is structured as follows: Section 2 offers background information on LLMs, accessibility in user interfaces, Jetpack Compose, and relevant related work. Section 3 details our methodology, describing the approach used to evaluate the accessibility of code generated from Figma templates. Section 4 presents the results obtained from applying this methodology. Section 5 discusses the results in relation to our research questions, and Section 6 concludes the paper with final remarks and directions for future work.

2 BACKGROUND

2.1 Mobile Software Development

Native mobile development involves building apps for a specific operating system using its supported languages. Kotlin or Java for Android and Swift or Objective-C for iOS [6]. Google's Jetpack Compose has recently become the standard for creating Android UIs. It introduces a declarative approach using composable functions written in Kotlin, enabling reactive updates based on app state changes. This contrasts with the earlier XML-based method, which required manual management of the state of the UI [18].

Compose improves accessibility by embedding accessibility attributes directly in its components, unlike XML where developers needed to manually configure them following Android's guidelines [17]. It also provides native support for screen readers, keyboard navigation, and other assistive tools [19]. Lassfolk highlights how semantic modifiers improve compatibility with TalkBack and similar technologies. However, accessibility issues persist in modern Android apps [8, 30, 32], even after the official release of Jetpack Compose[4].

2.2 Mobile Accessibility

Digital accessibility aims to eliminate barriers so that all users, including those with disabilities, can interact effectively with digital content [7]. The W3C's Web Content Accessibility Guidelines (WCAG) define international standards to support users with various disabilities [50].

Mobile accessibility adapts these principles to devices such as phones, tablets, and wearables, addressing challenges such as small screens, touch interaction, and use in varying environments [49]. Tools such as Google's Accessibility Scanner help identify interface issues, although they are limited in scope and not fully aligned with WCAG [1, 20].

Studies have explored the use of such tools to provide a more objective analysis of accessibility issues in mobile applications [5, 8, 30, 36, 38]. Andrade et al. evaluated eight popular Android apps in Portuguese, English, and Spanish, detecting more than 1,100 touch-target size issues and nearly 600 contrast failures. For example, in their study, the Spanish mode alone revealed 828 accessibility errors.

In addition to these tools, Chen et al. analyzed 2,270 Android apps using an automated exploration tool and documented more than 86,000 accessibility issues. Yan and Ramachandran found that nearly all 479 Android apps analyzed on Google Play had accessibility issues and more than 94% contained direct violations of accessibility guidelines. Furthermore, Krainz et al. found widespread inaccessibility in applications on the Google Play Store, including major platforms such as WhatsApp and Amazon.

Additional research [4, 21, 25], based on surveys with mobile developers, suggests that these accessibility shortcomings stem from a general lack of familiarity with accessibility standards and the rare inclusion of such concerns during app development. The respondents also highlighted that common development tools tend to neglect accessibility aspects, leading to improper implementation.

UI design also influences accessibility outcomes. For example, in [31], the authors identified more than 700 issues in mobile templates in Figma, underscoring how design stage decisions can propagate into the usability of the final product.

2.3 Generative Artificial Intelligence

Modern generative AI is based on the Transformer architecture, introduced by Google in 2017 [37]. This model uses attention mechanisms to focus on relevant parts of the input, improving contextual understanding and output quality [16, 43]. It underpins LLMs such as GPT, which are widely used in natural language processing [11].

LLMs are trained on massive text datasets using autoregressive prediction, where the model learns to predict the next word based on the prior context [53]. Due to their dependence on the context of input, the structure and clarity of user prompts significantly affect the quality of the output [54]. GPT (Generative Pre-trained Transformer), developed by OpenAI, follows this approach and powers tools like ChatGPT. Although it can produce fluent and contextually appropriate responses, it can also generate incorrect or incoherent output, called hallucinations [3, 37].

Beyond text generation, LLMs have shown potential in practical domains such as language learning [40], automated UI testing [51], and research assistance [37]. In particular, they have also been explored to improve digital accessibility. Othman et al. found that ChatGPT could resolve up to 94% accessibility issues on websites. However, the study stresses the importance of combining automation with manual validation for reliable results.

Recent research has assessed how LLMs generate accessible mobile UIs. For example, Rabelo et al. found that even when prompts explicitly requested accessible interfaces, the resulting Jetpack Compose and XML code often contained an increased number of accessibility errors. Similarly, Suh et al. showed that more detailed prompts improved accessibility outcomes when generating web content, but emphasized the need for iterative feedback to ensure robust results.

2.4 Related Work

To the best of our knowledge, no prior work has assessed the accessibility of mobile app interfaces generated by LLMs from image input. Although LLMs have been widely explored for code generation and developer support, their ability to produce accessible UIs, especially from visual design templates, remains largely unexamined. This is a critical gap, given the increasing use of design-to-code workflows and the importance of inclusive design in modern app development.

[4]https://android-developers.googleblog.com/2021/07/jetpack-compose-announcement.html

Recent studies [9, 35, 46] have investigated how visual input, such as screenshots or mockups, can guide LLM-based development. For example, Chen et al. propose a self-correcting framework built on GPT-4 that incorporates UI mockups and design metadata to improve front-end code generation. By decomposing complex interfaces, the system achieves a more than 30% higher accuracy compared to general-purpose LLMs without domain-specific tuning.

Park et al. leverage multimodal LLMs to extract semantic information from UI images, allowing designers to retrieve examples aligned with both functional and aesthetic goals. Their results show that GPT-4 outperforms baselines in screen category prediction, reaching a top-1 accuracy of 59.21%, and producing richer UI descriptions than prior datasets.

Focusing on testing, Wang et al. use LLMs to analyze UI screenshots to solve rendering issues such as missing labels, dark mode violations, and incomplete map regions. Their method simulates manual QA workflows and reduces reliance on hand-written checks, while improving coverage and accuracy. They highlight the importance of prompt engineering and curated error datasets in enabling effective model behavior.

Although these works demonstrate the potential of LLMs to interpret visual UI artifacts and support various development tasks, they do not address the accessibility of the generated interfaces. Whether such models can create UIs that adhere to accessibility guidelines and promote inclusive interaction remains an open question.

3 METHODOLOGY

Writing code from UI images is already a common task in software development. The use of LLMs can streamline this process by reducing manual effort, but it remains essential to assess their effectiveness in generating accessible code from visual inputs. Therefore, the main objective of this study is to evaluate the ability of Large Language Models to generate an accessible Android UI using Jetpack Compose based on **visual design inputs**, while also evaluating their ability to improve accessibility through iterative prompts. To address this objective, three research questions were created:

RQ1 - To what extent are LLMs capable of generating accessible Jetpack Compose UI from image inputs?
RQ2 - How effectively can LLMs suggest solutions to accessibility issues in Android applications developed with Jetpack Compose?
RQ3 - What are the most common categories of accessibility errors found in UI screens generated by LLMs?

Therefore, the methodology of this paper was organized into nine stages, as presented in Figure 1. This process was designed to address the research questions.

3.1 LLM Selection

The decision to use ChatGPT-4 in this study is based on its advanced natural language processing capabilities, its ability to understand complex instructions, and its adaptability to different contexts. As a more recent iteration of the ChatGPT model family, it offers improved coherence, accuracy, and a better understanding of technical requirements [33].

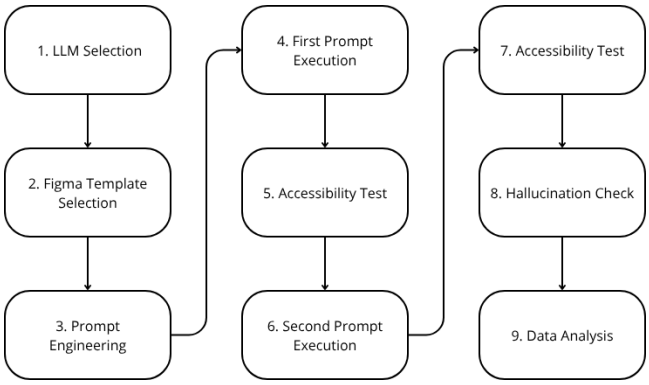

Figure 1: Methodology workflow

3.2 Figma Template Selection

The selection of Figma templates was based on a previous study [31], in which the authors initially identified 12 mobile application templates. After applying selection criteria such as popularity, structured navigation, and a minimum of four screens, they narrowed the set down to 10 templates. However, at the time of our study, two of these templates were no longer available in the Figma library. As a result, we used the remaining 8 templates, which are listed in Table 1. The purpose of using the same set of templates was to enable a direct comparison between our results and those of the original study. This allows us to evaluate how the LLM's performance aligns with or diverges from previously established baselines.

Table 1: Selected Figma templates.

Template Name	Category
Animal Wiki Mobile App	Education
BCA Mobile	Finance
Bestbuy App Design	Shopping
Food Ordering App	Food and Drink
Kapuha Music - Mobile App	Music and Audio
Music App Prototype	Music and Audio
RSPCA Mobile App	Medical
ToDo Mobile App	Productivity

3.3 Prompt Engineering

Figure 2 presents an example of the final prompt used in this experiment as the result of an iterative prompt engineering process aimed at ensuring that the LLM could generate the most functional version of the UI [54]. To assess the prompt's effectiveness, each output was tested by compiling the generated code in Android Studio and verifying the overall UI structure. Early versions of the prompt produced code that either failed to compile or resulted in incomplete

UI implementations (e.g., non-functional buttons or missing scrollable content). Through multiple refinement sessions, the prompt evolved to include specific instructions, such as strict use of Jetpack Compose, limited visual assets, and a defined structure for composable functions. This process guided the model toward generating stand-alone code that required minimal to no further intervention.

Despite the overall improvement in the output due to the refined prompt, the model still displayed limitations in replicating the original UI design with the same precision as a human developer. In addition, some hallucinations occasionally led to non-compilable code, which had to be manually commented out to avoid further intervention and preserve the integrity of the evaluation process.

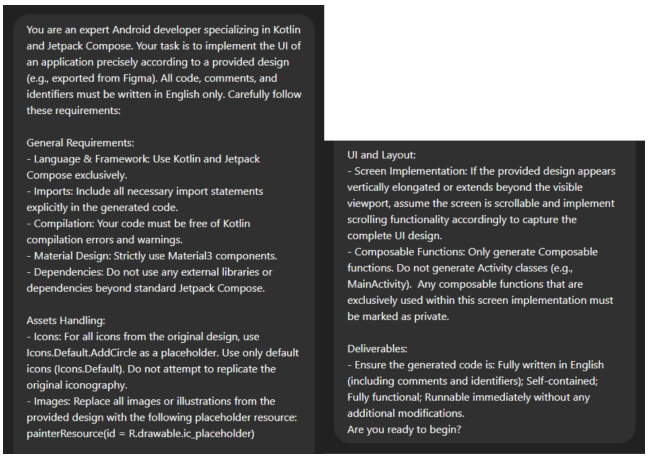

Figure 2: Prompt example for generating screens with Jetpack Compose.

3.4 Execution

Each generated screen was exported to Android Studio. We created a Compose Navigation Graph to organize them within a single application flow. To ensure test integrity and avoid introducing accessibility errors through code interference, no visual assets from templates were imported. Instead, a high-contrast drawable was used as a placeholder and a default icon was defined. Furthermore, to avoid context contamination from previous iterations, each prompt was submitted in a new chat session. Despite these precautions, the LLM occasionally ignored the prompt instructions and attempted to import custom assets, which had to be commented out. Some syntax errors introduced by the model also broke the UI structure and required manual correction.

3.4.1 Accessibility Test. To identify accessibility issues, we used Google's Accessibility Scanner, a tool that provides a visual and textual report containing suggestions on how to improve the accessibility of the app by scanning UI elements on the screen based on content labels, touch target size, clickable items, text and image contrast, as shown in Figure 3. Based on this report, we then prompted the LLM to refactor the code and create an improved version of the same screen. This iterative process of displaying the screen, scanning it, and prompting a refactoring was repeated for each template.

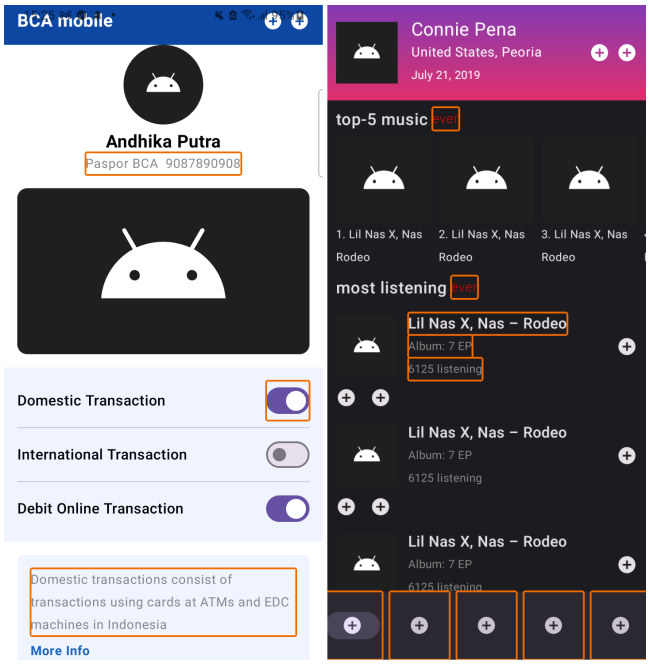

Figure 3: Example of accessibility reports for BCA and Kapuha Music templates with accessibility issues highlighted.

3.4.2 Hallucination Check. After generating and evaluating the accessibility of all screens, we selected the ones with more than 10 accessibility issues (total) to ensure the reliability of the results by running a hallucination check. In this stage, the 9 selected screens were generated again by re-running the original prompt two more times for each screen. LLMs are non-deterministic models, which means that they rely on probability and can produce different responses to the same input by exploring multiple possibilities in each execution [43]. This behavior can also lead them to generate output that appears valid, but is incorrect or hallucinated [3]. The objective of this step was to evaluate whether the issues were inherent in the templates or were hallucinated by the LLM, since running the prompt multiple times helps reveal inconsistencies and makes it easier to assess which outputs are more trustworthy by comparing the results. Once the new version of the screens was ready for testing, we ran the scanner one more time to compare the results and identify the root of the lack of accessibility.

4 RESULTS

In this study using ChatGPT-4, each of the 60 screens from the 8 templates was generated in both an initial version and a refactored version for accessibility. Furthermore, 9 additional screens were prompted two more times, and the corresponding refactored versions were created to check for hallucinations, resulting in a total of 156 screens. As shown in Table 2, a total of 302 accessibility errors were identified among the 120 initial screens before refactoring, with an average of 5.02 errors per screen. After prompting the LLM to improve accessibility, 138 issues were detected in the refactored versions, averaging 2.3 errors per screen. The additional 36 screens used for hallucination checks were not included in this analysis.

Table 2: Accessibility issues per initial and refactored screens versions.

Screen Version	Errors	Mean	Standard deviation
Initial Version	302	5.02	4.07
Refactored Version	138	2.30	2.81

To analyze the difference between the two approaches, we compared the 60 screens generated with the initial prompt and their corresponding refactored versions. Applying a paired Student's t-test to these comparisons, we obtained the following results: $t = 6.113, p = 0.00000008$. The result ($p < 0.05$) indicates that the differences are statistically significant.

Figure 4 summarizes the distribution of accessibility errors by category in each screen template, combining the results before and after refactoring. The most frequent issue was Text Contrast, with a total of 222 instances observed in all templates. Item Description errors were also common, totaling 108 occurrences, followed by Image Contrast with 46 instances. On the other hand, Unexposed Text and Touch Target were among the least frequent issues, with only 12 and 3 occurrences, respectively.

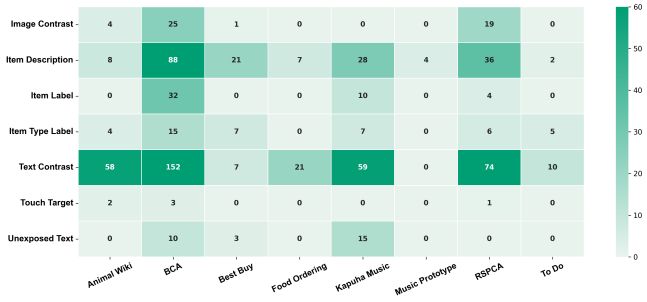

Figure 4: Accessibility errors by error category and template in all 120 screens.

Table 3 presents the number of accessibility issues identified in each template, both before and after the LLM was prompted to refactor the screens based on the accessibility scanner report. The template with the most errors initially was the BCA app, with 123 issues in total and a mean of 5.85 errors per screen. The most frequent issue was Text Contrast, which accounted for more than 50% of its total errors. After accessibility refactoring, the total number of errors in the BCA app decreased to 54, with Text Contrast issues representing less than 30% of the total. However, the number of Item Type Label issues rose from 0 to 11, indicating that the LLM introduced these errors during the refactoring process.

The introduction of new types of errors was not limited to the BCA scenario, as was observed in 5 of the 8 templates, accounting for a total of 40 additional errors across them. After requesting a regeneration of the code for accessibility improvements, 4 templates presented new Item Type Label issues, 3 presented new Unexposed Text issues, and 1 presented new Item Label issues. Table 3 shows that the Best Buy template initially had no Item Type Label occurrences, but showed 7 such violations after refactoring. Similarly, the Kapuha Music template, which originally had no Unexposed Text issues, exhibited 3 after the update. Figure 5 shows the distribution of accessibility errors per screen for each template, comparing initial and refactored versions. Most templates, such as Animal Wiki, BCA, and RSPCA, show a reduction in both the median number of errors and the overall variability after refactoring. However, Kapuha Music stands out with an increase in errors, suggesting that the request for improvement sometimes introduced new issues.

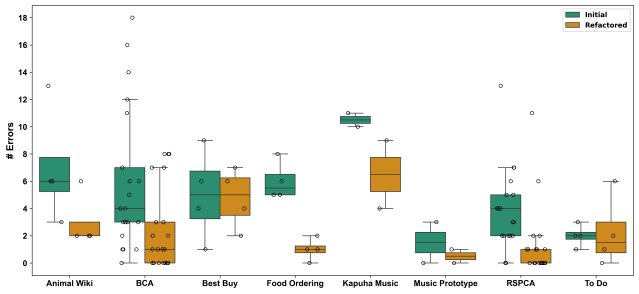

Figure 5: Distribution of errors per screen in each template.

Figure 6 presents an overview of the number of accessibility errors per screen in all templates, comparing the initial and refactored versions. Each subplot corresponds to a template and shows the error counts for its individual screens. Among the 60 initial screens, 9 had the highest number of accessibility errors, together accounting for almost 40% of the 302 reported issues. After the prompted refactoring, 5 of these 9 screens showed an improvement of at least 50%. Separately, some other screens introduced new errors after the refactoring. For example, the To Do Screen in the To Do template and the Walk Record Screen in RSPCA had more issues after being refactored.

To analyze the likelihood of hallucinations in the generated screens, we selected all screens with an initial error count greater than or equal to 10, resulting in the 9 listed in Table 4. Each screen was generated twice, producing versions V2 and V3 for comparison. The table presents the error count for each version along with its standard deviation. The Profile Screen from the BCA template had 18 errors in the initial version, making it the screen with the highest number of issues among all 120. It also showed the highest standard deviation (6.34), suggesting that most of its initially reported errors were probably hallucinated. Although none of the other screens exhibited such high variability, the Transaction Screen, also from the BCA template, showed the second highest deviation at 3.26. All remaining screens had standard deviations below 3, but still indicated signs of hallucination.

5 DISCUSSION

5.1 RQ1 - To what extent are LLMs capable of generating accessible Jetpack Compose UI from image inputs?

Main Result: All screens generated from the Figma UI templates exhibited accessibility issues, mainly related to Text Contrast and Item Descriptions.

Table 3: Accessibility issues per template without prompting and prompting for accessibility.

Template	Prompt	Text Contrast	Touch Target	Item Description	Image Contrast	Item Type Label	Item Label	Unexposed Text	Total	Mean
Animal Wiki	Initial	23	1	2	2	0	0	0	**28**	**7.00**
	Refactored	10	1	1	0	0	0	0	**12**	**3.00**
BCA	Initial	63	0	30	18	0	11	1	**123**	**5.85**
	Refactored	16	0	15	6	11	1	5	**54**	**2.57**
Best Buy	Initial	6	0	13	1	0	0	0	**20**	**5.00**
	Refactored	1	0	8	0	7	0	3	**19**	**4.75**
Food Ordering	Initial	17	0	7	0	0	0	0	**24**	**6.00**
	Refactored	4	0	0	0	0	0	0	**4**	**1.00**
Kapuha Music	Initial	12	0	4	0	0	5	0	**21**	**10.50**
	Refactored	6	0	4	0	0	0	3	**13**	**6.5**
Music Prototype	Initial	0	0	3	0	0	0	0	**3**	**1.50**
	Refactored	0	0	1	0	0	0	0	**1**	**0.50**
RSPCA	Initial	44	1	11	18	1	0	0	**75**	**3.94**
	Refactored	10	0	7	1	4	4	0	**26**	**1.36**
To Do	Initial	7	0	1	0	0	0	0	**8**	**2.00**
	Refactored	3	0	1	0	5	0	0	**9**	**2.25**

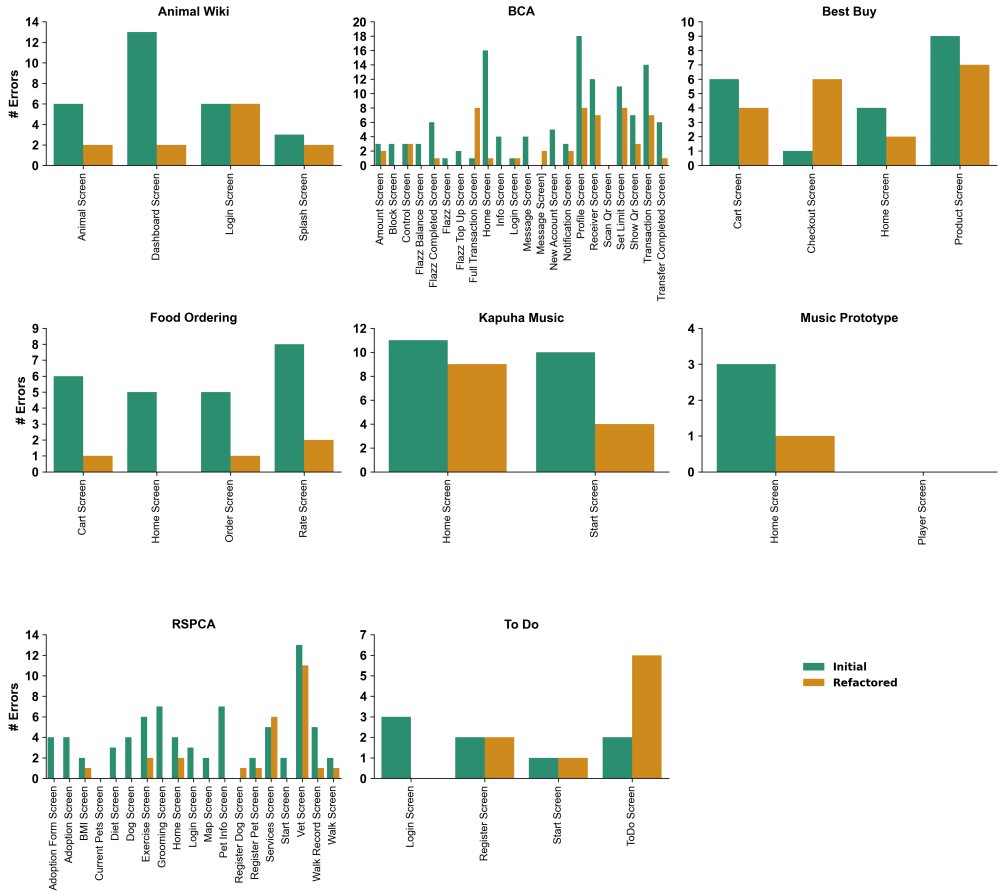

Figure 6: Accessibility errors across templates for each screen.

Table 4: Error comparison with standard deviation as a measure of hallucination

Template	Screen	V1	V2	V3	Std
BCA	Profile	18	3	7	6.34
BCA	Home	16	11	14	2.05
BCA	Transaction	14	10	18	3.26
Animal Wiki	Dashboard	13	16	14	1.24
RSPCA	Vet	13	12	15	1.24
BCA	Receiver	12	11	13	0.81
BCA	Set Limit	11	16	10	2.62
Kapuha Music	Home	11	14	17	2.44
Kapuha Music	Start	10	8	11	1.24

The results of this study show that LLMs are capable of generating Jetpack Compose screens based on image input. However, when accessibility was tested, there were 302 violations on 60 screens of 8 templates, with an average of 5.02 issues per screen.

In contrast, the study by Muniz et al. found 738 issues on 97 screens, averaging 7.61 issues per screen. Although both studies are using roughly the same templates, the screens generated by LLMs with Jetpack Compose showed a lower average number of errors. However, this comparison is not as direct as it seems, as other factors, such as the number of screens, evaluation tools, programming language, and screen complexity, can also affect the results. For example, in Text Contrast issues, the most frequent type of accessibility issue in our results, the study by Muniz et al. performed better, with an average of 2.34 issues per screen compared to our 2.87. Furthermore, in the hallucination check, which involved nine selected screens, half of the generated versions contained fewer accessibility issues than the originals. This suggests that while the overall results are promising, LLMs can still deviate from the input image and introduce new errors during generation.

Another study that presented similar results was that of Suh et al.. In their work, LLMs were prompted to generate user interfaces based on textual descriptions extracted from original Web pages, which were then evaluated for accessibility. Their results align with ours, as color contrast issues were reported as the most common accessibility problem. However, a key difference in our studies is that, compared to human-written code, their findings show that ChatGPT-4 generated code with nearly 49% fewer contrast issues. Furthermore, they observed that LLM-generated code occasionally introduced new accessibility issues not present in the original designs, a pattern also reflected in our results.

5.2 RQ2 - How effectively can LLMs suggest solutions to accessibility issues in Android applications developed with Jetpack Compose?

Main Result: Results showed that, when prompted with the Accessibility Scanner's report, the LLM was able to resolve 50% of the identified accessibility issues.

The analysis of the results demonstrates that LLMs can be a valuable tool in resolving accessibility issues when guided correctly. In our study, providing the model with a detailed report of the Accessibility Scanner as input led to a significant improvement of approximately 54%, reducing the initial 302 errors to just 138. This highlights the potential of targeted prompting. The importance of this specific, feedback-driven approach is underscored in contrast to the findings of Rabelo et al.. Their research showed that using a generic prompt was not only ineffective, but often counterproductive, frequently increasing the number of accessibility errors.

These results are consistent with the findings of Othman et al., who evaluated ChatGPT's ability to automatically fix web accessibility issues. In their study, ChatGPT corrected 37 of 39 violations identified on two real websites, achieving a success rate of 94%. Although the studies diverge in context as theirs was based on HTML web content, both studies show that LLMs perform well when given structured feedback, being able to resolve accessibility related issues.

However, despite showing a positive performance in solving accessibility issues in both scenarios, the results also show that only 164 of the 302 reported errors were corrected, giving a success rate of just barely more than 50%. This reinforces the idea that LLMs are tools meant to assist developers in achieving this objective, but still require manual evaluation to ensure quality.

Furthermore, when asked to correct the errors reported by the scanner, the LLM introduced 40 additional violations in three different categories. These findings show that, while the model is unable to resolve 100% of the reported issues, it is also prone to introduce new violations in the process. This suggests that, in order to effectively address known issues, developers should consistently evaluate the model's output and work iteratively to ensure that the expected results are achieved.

5.3 RQ3 - What are the most common categories of accessibility errors found in UI screens generated by LLMs?

Main Result: The most frequent accessibility issues in LLM-generated UIs were related to Text Contrast, Item Descriptions, and Image Contrast, with Text Contrast being the most frequent overall.

The most frequent error category was Text Contrast, with 222 occurrences across all template iterations, accounting for 73.5% of all issues. Other significant occurrences were for Item Description (108) and Image Contrast (46) categories. On the other hand, Unexposed Text (12) and Touch Target (3) were the less frequently reported categories.

These findings align with similar studies [4, 30, 36, 44], such as Alshayban et al., which conducted a study on accessibility issues in more than 1,000 Android applications. Their results also highlight Text Contrast as the most prevalent issue, with 22.81% occurrences, followed by Touch Target Size (19.78%) and Image Contrast (12.85%).

These similarities may suggest that the LLM is not necessarily introducing new types of error but rather replicating patterns it has learned from training data. Given that real-world applications still present high accessibility issues rates, as reported by studies [4, 23, 36] it is possible to assume that the model has been exposed to examples with such flaws. As a result, the recurrence of

inaccessibility in LLM-generated UIs might reflect common practices found in the source material it was trained on, reinforcing the need to explicitly instruct the LLM with well-structured prompts to guide it toward implementing good practices.

5.4 Implications and Takeaways

The findings of this study suggest that LLMs can offer real support in the development of user interfaces accessible from image input using Jetpack Compose for Android applications, especially when paired with feedback from evaluation tools. Although none of the generated screens was fully accessible on the first attempt, the model was able to reduce the number of issues by more than 50% after being prompted with the scanner report. That alone points to its potential as an assistive tool in UI development workflows.

At the same time, the results also highlight that these models are far from reliable when used in isolation. In several cases, ChatGPT failed to fix all reported issues and even introduced new ones. This reinforces the importance of human review and iterative testing, especially when the goal is to meet accessibility standards. Developers should approach LLMs not as end-to-end solutions but as starting points that still require critical oversight. Some of the reported errors, such as Text Contrast and missing Item Descriptions, are also common in manual developed apps [4, 30, 44], suggesting that LLMs may be replicating existing development patterns. This raises questions about the training data these models were exposed to and how that influences the accessibility of their output.

Together, the results of this study reinforce that LLMs can contribute meaningfully to accessibility-focused development workflows, but only when used with intention and care. The quality of the prompt played a crucial role in guiding the model toward usable outputs, which highlights the importance of prompt engineering as part of the process. At the same time, developers must be familiar with accessibility standards to evaluate and refine the prompt and generated code effectively. It is also important to note that some accessibility issues observed in the outputs were already present in the input images [31], suggesting that the quality of the original design directly impacts the result. For that reason, when using image-exported UI designs to generate accessible app screens with LLMs, developers should critically review their image inputs to ensure their accessibility in order to achieve the best results.

5.5 Threats to Validity

In terms of construct validity, accessibility evaluation was based on Google's Accessibility Scanner. Although useful for detecting issues such as missing labels, touch targets, and contrast problems, it covers only a limited range of accessibility barriers and does not assess their severity. Involving users with disabilities would have provided an essential context for understanding the real impact of these issues.

Another limitation is that screen generation used a zero-shot prompt, focusing only on instructions to the LLM. Other prompt strategies were not tested, so different approaches could have reduced the number of detected errors.

For internal validity, the prompt was designed to minimize human edits, but some manual adjustments were needed to compile the code. Although accessibility logic was unchanged, these edits may have influenced the results. The hallucination check showed that the model can produce varying outputs for the same input, causing differences in error counts. Each prompt was re-run only twice, so more repetitions could have provided more reliable data.

Regarding external validity, the results are limited to the scope of this study: 8 Figma templates and 60 screens designed for Android and Jetpack Compose. These do not cover the full diversity of mobile UIs, especially customized applications or other technologies. The exclusive use of English prompts also limits generalizability, as LLM performance can vary by language [22, 36, 45]. Therefore, the types and frequency of generated errors may not apply directly to other platforms or languages.

6 FINAL CONSIDERATIONS

This study examined the ability of LLMs to generate accessible Jetpack Compose screens from Figma-based UI templates. The results suggest that, although LLMs can generate code from designed UIs, they often reproduce existing accessibility issues present in the original designs and may even introduce new errors. However, with a designed prompt and feedback from tools such as the Accessibility Scanner, the model showed the ability to resolve more than half of the accessibility issues.

These results support the idea that LLMs can help developers during the development of user interfaces. However, to develop accessible applications, **human review** and adjustments **are required** to ensure that accessibility guidelines are followed.

Future studies should investigate whether LLMs can generate fully accessible screens using accessible real-world applications as input and compare the results. This includes evaluating how well LLMs can fix accessibility issues of varying severity. These evaluations should not be limited to issues flagged by automated accessibility scanners but should also include problems identified by real users with disabilities. Furthermore, the impact of prompt language warrants future investigation, especially regarding its performance and consistency across multiple languages. Moreover, evaluating the use of Jetpack Compose Multiplatform could help measure the model's ability to produce accessible UIs across different platforms, including web, desktop, and iOS, from a single codebase.

ACKNOWLEDGMENTS

This research was partially supported by the Brazilian National Council for Scientific and Technological Development (Conselho Nacional de Desenvolvimento Científico e Tecnológico [CNPq]), under grant number 307831/2025-6.

ChatGPT-4 was used in the preparation of this paper to help with text revision.

REFERENCES

[1] Patricia Acosta-Vargas, Rasa Zalakeviciute, Sergio Luján-Mora, and Wilmar Perdomo. 2019. *Accessibility Evaluation of Mobile Applications for Monitoring Air Quality: Helping Teachers Develop Research Informed Practice.* Springer Nature, 638–648. https://doi.org/10.1007/978-3-030-11890-7_61

[2] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millicah, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. 2022. Flamingo: a visual

language model for few-shot learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1723, 21 pages.

[3] Hussam Alkaissi and Samy I. McFarlane. 2023. Artificial Hallucinations in ChatGPT: Implications in Scientific Writing. *Cureus* 15, 2 (2023), e35179. https://doi.org/10.7759/cureus.35179

[4] Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek. 2020. Accessibility issues in Android apps: state of affairs, sentiments, and ways forward. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1323–1334. https://doi.org/10.1145/3377811.3380392

[5] Matheus Andrade, Daniel Rabelo, Ribamar Martins, and Windson Viana. 2024. Investigating the accessibility of popular mobile Android apps: a prevalence, category, and language study. In *Proceedings of the 30th Brazilian Symposium on Multimedia and the Web* (Juiz de Fora/MG). SBC, Porto Alegre, RS, Brasil, 400–404. https://doi.org/10.5753/webmedia.2024.242041

[6] Andreas Biørn-Hansen, Christoph Rieger, Tor-Morten Grønli, Tim A. Majchrzak, and Gheorghita Ghinea. 2020. An Empirical Investigation of Performance Overhead in Cross-Platform Mobile Development Frameworks. *Empirical Software Engineering* 25, 4 (2020), 2997–3040. https://doi.org/10.1007/s10664-020-09827-6

[7] Brasil. 2019. *Acessibilidade Digital*. https://www.gov.br/governodigital/pt-br/acessibilidade-e-usuario/acessibilidade-digital

[8] Sen Chen, Chunyang Chen, Lingling Fan, Mingming Fan, Xian Zhan, and Yang Liu. 2022. Accessible or Not? An Empirical Investigation of Android App Accessibility. arXiv:2203.06422 [cs.SE] https://arxiv.org/abs/2203.06422

[9] Yunnong Chen, Shixian Ding, YingYing Zhang, Wenkai Chen, Jinzhou Du, Lingyun Sun, and Liuqing Chen. 2025. DesignCoder: Hierarchy-Aware and Self-Correcting UI Code Generation with Large Language Models. arXiv:2506.13663 [cs.SE] https://arxiv.org/abs/2506.13663

[10] Jingwen Cheng, Kshitish Ghate, Wenyue Hua, William Yang Wang, Hong Shen, and Fei Fang. 2025. REALM: A Dataset of Real-World LLM Use Cases. arXiv:2503.18792 [cs.HC] https://arxiv.org/abs/2503.18792

[11] Cole Stryker and Jim Holdsworth, IBM. 2024. *What Is NLP (Natural Language Processing)?* IBM. https://www.ibm.com/think/topics/natural-language-processing Accessed: 2025-07-07.

[12] Daniel de Sousa Moraes, Polyana Bezerra da Costa, Antonio JG Busson, José Matheus Carvalho Boaro, Carlos de Salles Soares Neto, and Sergio Colcher. 2023. On the Challenges of Using Large Language Models for NCL Code Generation. In *Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia)*. SBC, 151–156.

[13] Giovanni Delnevo, Manuel Andruccioli, and Silvia Mirri. 2024. On the Interaction with Large Language Models for Web Accessibility: Implications and Challenges. In *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*. 1–6. https://doi.org/10.1109/CCNC51664.2024.10454680

[14] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2024. Evaluating Large Language Models in Class-Level Code Generation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 81, 13 pages. https://doi.org/10.1145/3597503.3639219

[15] Reinaldo Ferraz, Ana Duarte, João Bárbara, Adriano CM Pereira, and Wagner Meira. 2023. A platform to check website compliance with web accessibility standards. In *Proceedings of the 20th International Web for All Conference*. 75–78.

[16] Ana Cristina Bicharra Garcia. 2020. Ética e Inteligência Artificial. *Computação Brasil* 43 (nov. 2020), 14–22. https://doi.org/10.5753/compbr.2020.43.1791

[17] Google. 2024. *Principles for Improving App Accessibility*. Google. https://developer.android.com/guide/topics/ui/accessibility/principles

[18] Google. 2024. *Why adopt Compose*. Google. https://developer.android.com/develop/ui/compose/why-adopt

[19] Google. 2025. *Accessibility in Jetpack Compose*. Google. https://developer.android.com/develop/ui/compose/accessibility

[20] Google LLC. 2025. *Get started with Accessibility Scanner*. Google. https://support.google.com/accessibility/android/answer/6376570 Accessed: 2025-07-07.

[21] Marianna Di Gregorio, Dario Di Nucci, Fabio Palomba, and Giuliana Vitiello. 2022. The making of accessible Android applications: an empirical study on the state of the practice. *Empirical Software Engineering* 27, 6 (2022), 145. https://doi.org/10.1007/s10664-022-10182-x Open access under CC BY 4.0.

[22] Kei Koyanagi, Dong Wang, Kotaro Noguchi, Masanari Kondo, Alexander Serebrenik, Yasutaka Kamei, and Naoyasu Ubayashi. 2024. Exploring the Effect of Multiple Natural Languages on Code Suggestion Using GitHub Copilot. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. IEEE, 481–486.

[23] Elmar Krainz, Klaus Miesenberger, and Johannes Feiner. 2018. Can We Improve App Accessibility with Advanced Development Methods?. In *Computers Helping People with Special Needs: 16th International Conference, ICCHP 2018, Linz, Austria, July 11-13, 2018, Proceedings, Part I* (Linz, Austria). Springer-Verlag, Berlin, Heidelberg, 64–70. https://doi.org/10.1007/978-3-319-94277-3_12

[24] Elsa Lassfolk. 2023. *User Experience App Design for Visually Impaired Elderly*. Technical Report. Metropolia University of Applied Sciences, Helsinki, Finland.

[25] Manoel Victor Rodrigues Leite, Lilian Passos Scatalon, André Pimenta Freire, and Marcelo Medeiros Eler. 2021. Accessibility in the mobile development industry in Brazil: Awareness, knowledge, adoption, motivations and barriers. *Journal of Systems and Software* 177 (2021), 110942. https://doi.org/10.1016/j.jss.2021.110942

[26] Guilherme Lima, Rodrigo Costa, and Marcio Moreno. 2019. Semantics and multimedia: an introduction to symbolic artificial intelligence applied to multimedia. In *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*. 7–9.

[27] Matheus Lima, Guido Lemos, and Raoni Kulesza. 2021. A Proposal for an Accessible Videoconferencing Interface for People with Hearing Impairments. In *Proceedings of the Brazilian Symposium on Multimedia and the Web*. 29–36.

[28] Yihao Liu, Xu Cao, Tingting Chen, Yankai Jiang, Junjie You, Minghua Wu, Xiaosong Wang, Mengling Feng, Yaochu Jin, and Jintai Chen. 2025. From screens to scenes: A survey of embodied AI in healthcare. *Information Fusion* 119 (July 2025), 103033. https://doi.org/10.1016/j.inffus.2025.103033

[29] Juan-Miguel López-Gil and Juanan Pereira. 2024. Turning manual web accessibility success criteria into automatic: an LLM-based approach. *Universal Access in the Information Society* 24 (2024), 837–852. https://doi.org/10.1007/s10209-024-01108-z

[30] Ribamar Martins, Daniel Rabelo, Maria Araújo, and Windson de Carvalho. 2022. Where is the description? Investigating accessibility issues in portuguese versions of smart home apps. In *Anais do XXI Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais* (Diamantina). SBC, Porto Alegre, RS, Brasil. https://sol.sbc.org.br/index.php/ihc/article/view/22279

[31] Júlia Holanda Muniz, Daniel Mesquita Feijó Rabelo, and Windson Viana. 2024. Assessing Accessibility Levels in Mobile Applications Developed from Figma Templates. In *Proceedings of the 17th International Conference on PErvasive Technologies Related to Assistive Environments* (Crete, Greece) *(PETRA '24)*. Association for Computing Machinery, New York, NY, USA, 316–321. https://doi.org/10.1145/3652037.3652075

[32] Alberto Dumont Alves Oliveira, Paulo Sérgio Henrique Dos Santos, Wilson Estécio Marcílio Júnior, Wajdi M Aljedaani, Danilo Medeiros Eler, and Marcelo Medeiros Eler. 2023. Analyzing Accessibility Reviews Associated with Visual Disabilities or Eye Conditions. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 37, 14 pages. https://doi.org/10.1145/3544548.3581315

[33] OpenAI. 2023. GPT-4 Technical Report. https://openai.com/research/gpt-4. Accessed: 2025-07-22.

[34] Achraf Othman, Amira Dhouib, and Aljazi Nasser Al Jabor. 2023. Fostering websites accessibility: A case study on the use of the Large Language Models ChatGPT for automatic remediation. In *Proceedings of the 16th International Conference on PErvasive Technologies Related to Assistive Environments* (Corfu, Greece) *(PETRA '23)*. Association for Computing Machinery, New York, NY, USA, 707–713. https://doi.org/10.1145/3594806.3596542

[35] Seokhyeon Park, Yumin Song, Soohyun Lee, Jaeyoung Kim, and Jinwook Seo. 2025. Leveraging Multimodal LLM for Inspirational User Interface Search. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. ACM, 1–22. https://doi.org/10.1145/3706598.3714213

[36] Daniel Mesquita Feijo Rabelo, Ribamar de Souza Martins, Isaac Santos, Paulo Henrique G. Da Silva, Kiev Gama, and Windson Viana. 2025. Breaking Barriers in Mobile Accessibility: A Study of LLM-Generated Native Android Interfaces . In *2025 IEEE/ACM 12th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE Computer Society, Los Alamitos, CA, USA, 20–31. https://doi.org/10.1109/MOBILESoft66462.2025.00010

[37] Anatália Saraiva Martins Ramos. 2023. Inteligência Artificial Generativa baseada em grandes modelos de linguagem - ferramentas de uso na pesquisa acadêmica. *SciELO Preprints* (may 2023). https://doi.org/10.1590/SciELOPreprints.6105

[38] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O. Wobbrock. 2020. An Epidemiology-inspired Large-scale Analysis of Android App Accessibility. *ACM Trans. Access. Comput.* 13, 1, Article 4 (April 2020), 36 pages. https://doi.org/10.1145/3348797

[39] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces* (Sydney, NSW, Australia) *(IUI '23)*. Association for Computing Machinery, New York, NY, USA, 491–514. https://doi.org/10.1145/3581641.3584037

[40] Sarang Shaikh, Sule Yildirim Yayilgan, Blanka Klimova, and Marcel Pikhart. 2023. Assessing the Usability of ChatGPT for Formal English Language Learning. *European Journal of Investigation in Health, Psychology and Education* 13 (09 2023), 1937–1960. https://doi.org/10.3390/ejihpe13090140

[41] Hyunjae Suh, Mahan Tafreshipour, Sam Malek, and Iftekhar Ahmed. 2025. Human or LLM? A Comparative Study on Accessible Code Generation Capability. arXiv:2503.15885 [cs.SE] https://arxiv.org/abs/2503.15885

[42] Michael Cheng-Tek Tai. 2020. The impact of artificial intelligence on human society and bioethics. *Tzu Chi Medical Journal* 32, 4 (Oct–Dec 2020), 339–343.

https://doi.org/10.4103/tcmj.tcmj_71_20

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[44] Christopher Vendome, Diana Solano, Santiago Liñán, and Mario Linares-Vásquez. 2019. Can Everyone use my app? An Empirical Study on Accessibility in Android Apps. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 41–52. https://doi.org/10.1109/ICSME.2019.00014

[45] Chaozheng Wang, Zongjie Li, Cuiyun Gao, Wenxuan Wang, Ting Peng, Hailiang Huang, Yuetang Deng, Shuai Wang, and Michael R. Lyu. 2024. Exploring Multi-Lingual Bias of Large Code Models in Code Generation. arXiv:2404.19368 [cs.SE] https://arxiv.org/abs/2404.19368

[46] Fei Wang, Kamakshi Kodur, Michael Micheletti, Shu-Wei Cheng, Yogalakshmi Sadasivam, Yue Hu, and Zening Li. 2024. Large Language Model Driven Automated Software Application Testing. Technical Disclosure Commons. https://www.tdcommons.org/dpubs_series/6815

[47] World Health Organization. 2023. *Blindness and Visual Impairment*. World Health Organization. https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment

[48] World Health Organization. 2023. *Disability*. World Health Organization. https://www.who.int/news-room/fact-sheets/detail/disability-and-health

[49] World Wide Web Consortium (W3C). 2023. *Mobile Accessibility at W3C*. World Wide Web Consortium (W3C). https://www.w3.org/WAI/standards-guidelines/mobile/ Accessed: 2025-07-07.

[50] World Wide Web Consortium (W3C). 2025. *Web Content Accessibility Guidelines (WCAG) 2.1*. World Wide Web Consortium (W3C). https://www.w3.org/TR/WCAG21/ Accessed: 2025-07-07.

[51] Zhuolin Xu, Qiushi Li, and Shin Hwei Tan. 2025. Guiding ChatGPT to Fix Web UI Tests via Explanation-Consistency Checking. arXiv:2312.05778 [cs.SE] https://arxiv.org/abs/2312.05778

[52] Shunguo Yan and P. G. Ramachandran. 2019. The Current Status of Accessibility in Mobile Apps. *ACM Trans. Access. Comput.* 12, 1, Article 3 (Feb. 2019), 31 pages. https://doi.org/10.1145/3300176

[53] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (LLM) security and privacy: The Good, The Bad, and The Ugly. *High-Confidence Computing* 4, 2 (2024), 100211. https://doi.org/10.1016/j.hcc.2024.100211

[54] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 437, 21 pages. https://doi.org/10.1145/3544548.3581388