Evaluating Large Language Models for Accessible Web Code Generation: A Comparative Study Using the FeedA11y Framework

Diya Singh[1], Prof. Sanjay Kumar Sharma[2]

[1]Student, University School of Information & Communication Technology, Gautam Buddha University
[2]Professor, University School of Information & Communication Technology, Gautam BuddhaUniversity

Abstract
The role of Large Language Models (LLMs) in software development has become quite common, integrating them more deeply into the daily routines of developers. This makes it increasingly important for LLMs to generate code that is not only coherent but also easy for humans to read, especially for creating different web applications. While previous studies have focused on individual LLMs regarding their functionality or security, there's a noticeable lack of research comparing multiple LLM platforms in terms of accessibility, correctness, and usability. This gap is significant and serves as the driving force behind our work. We conducted a comprehensive empirical study to evaluate the coding capabilities of eight leading LLMs: Chat GPT, Claude Ai, Gemini, Mistral, Qwen Ai, Phind, Grok and Blackbox Ai. Our assessment not only looked at how well these models generate forms, user interfaces, and semantic ARIA roles for webpages but also measured five key criteria: compliance with WCAG 2.0, code accuracy, responsiveness to prompts, cost-effectiveness, and user-friendliness. To enhance efficiency, we also modified the FeedA11y framework to automate evaluations using tools like AChecker for automated web accessibility assessments. Our findings highlight significant differences among the LLMs. Lastly, we outline the proposed solutions and frameworks and discuss the implications of LLMs on inclusive development processes.

Keywords: LLM, FeedAlly.

1. Introduction

The way Large Language Models (LLMs) are being used in Software Engineering is evolving faster than ever, fundamentally changing how developers brainstorm, write, test, and maintain their code. With tools like GitHub Copilot, ChatGPT, and Claude, developers can now code with minimal input, boosting productivity and speeding up development timelines. However, this rapid adoption of LLMs raises important concerns, particularly about the quality and inclusivity of their outputs. This is especially relevant in the realm of web accessibility, which fundamentally relies on the WCAG 2.0 standards to ensure fair access for everyone.

While some studies have examined LLMs in terms of their performance in code generation, debugging, or security assessments, fewer have explored how well these models adhere to accessibility guidelines or inclusive design principles. To address this gap, we're taking a closer look at eight of the leading LLMs available today, including general-purpose models like GPT, Claude and Gemini, as well as code-specific models like Qwen Ai. Our aim is to assess how effectively they generate HTML, CSS, and JavaScript code, while also capturing the nuances of prompting and feedback in iterative processes. We're expanding the FeedA11y framework, originally designed to enhance web code sourced from LLMs through ReAct-style accessibility critiques, to include support for cross-model comparison analyses.

Our research aims to tackle some key questions:

- RQ1: Which LLMs make it easiest to access web code?
- RQ2: How do different platform prompting strategies (like Zero-shot, Few-shot, and Self-Criticism) impact accessibility outcomes?
- RQ3: Which platforms strike the best balance between accessibility, accuracy, cost, and responsiveness?
- RQ4: What common accessibility issues or failure patterns do LLMs tend to exhibit?

This paper presents a thorough, reproducible benchmark for evaluating LLM accessibility, featuring cross-architecture empirical assessments and practical tips for choosing the best LLMs for inclusive code generation.

2. Related Work

The latest improvements in Large Language Models (LLMs) have greatly impacted the software development cycle, particularly with the automation of coding tasks. Research such as Suh et al. [1] and Anand et al. [2] has noted the increasing importance of LLMs like ChatGPT and GitHub Copilot in aiding development automation. Yet, most of these works centre on general code synthesis, security, or productivity and not on the outputs that are generated and their accessibility.

Multiple frameworks have been developed to evaluate the ability of LLMs to produce code that is syntactically and semantically executable at a given level [6][8]. However, very few address the practical compliance with web accessibility standards such as WCAG 2.0/2.1. For example, Suh et al. [1] did try to compare human coded accessible web pages and LLM generated counterparts, but the focus was on a narrow range of models and tasks. Also, Cai et al. [3] investigated the use of LLMs for GUI design but did not frame accessibility benchmarks either systematically or comparatively**.**

Agarwal et al.'s works [6] describe the creation of automated evaluation harnesses for assessing LLM-guided programming. Such tools are relevant, but focus mainly on correctness and efficiency, neglecting critical aspects such as usability, inclusive design, or other vital ergonomic factors. Other comparative evaluation works, like those by Liusie et al. [5], have investigated the capabilities of zero-shot generation, but mostly from the NLG perspective. Up to this moment, no single study has assessed all available LLMs in an organized manner using structured prompts focused on accessibility compliance. Gaps like these are addressed in this paper, where eight LLMs are benchmarked with WCAG-aligned HTML/CSS tasks using a tailored version of FeedA11y for automated feedback refinements.

3. Methodology

This section dives into the experimental design we used to compare and evaluate the performance of eight large language models (LLMs) in creating web code that is both accessible and accurate. We'll cover everything from task selection and model settings to prompting techniques, the code generation pipeline, and the evaluation methods and tools we employed.

3.1 LLM Platforms

We chose eight different LLM platforms that include both general-purpose and code-focused models. Our selection was based on their popularity, availability, and how well they fit into today's software development workflows.

Qwen: Alibaba's open-source multilingual LLM family with superior Chinese/English reasoning and coding capabilities.

Mistral: French startup famous for very powerful open-source LLMs such as Mixtral MoE comparable to bigger models.

Phind: Search engine for developers with AI-driven, LLM-powered, and cited solutions to coding and tech-related questions.

ChatGPT: A hit chatbot app by OpenAI (based on GPT series) for conversations, writing, and projects.

Claude: Anthropic's assistant AI prioritizing safety with industry-leading context windows and sound reasoning

Gemini: Google's multimodal chatbot embedded within the company's ecosystem, excellent at searching and facts.

Blackbox: Autocomplete, search, and chat coding assistant supporting over 20 programming languages.

Grok: xAI's chatbot within X (Twitter) with the reputation of having less filtering and real-time access.

3.2 Task Design

To assess how well LLMs perform, we came up with a practical coding task that really focus on accessibility and usability:

1. UI Code Regeneration from code summaries (inspired by FeedA11y).
2. Fixing Form Accessibility: This involves adding missing labels, ARIA roles, and alt text.
3. Correcting Table Accessibility: We look at scope, headers, and captions.
4. Applying ARIA Landmarks: This means ensuring semantic roles are applied correctly and are unique.
5. Integrating Skip Navigation: We want to make sure keyboard users can easily skip to the main content.

Each of these tasks is based on common failure points identified in WCAG 2.1 from previous studies.

3.3 Prompting Strategies

To understand how large language models (LLMs) react to varying levels of instruction, we can break it down like this:

- Naive Prompting: This approach doesn't mention accessibility at all.
- Zero-Shot: Here, we provide basic instructions aimed at achieving WCAG compliance. • Few-Shot: This involves outlining WCAG rules along with examples of what's correct and what's not.
- Self-Criticism: In this case, the model takes a moment to review and enhance its own code.
- FeedA11y: This method focuses on refining the model based on feedback, using a ReAct-style approach that incorporates evaluator reports.

We'll assess each model using the Naive, Zero-Shot, and FeedA11y configurations.

3.4 Code Generation Pipeline

We've got a well-organized process inspired by FeedA11y that goes like this:

1. Summarization Phase: An LLM creates code summaries based on the input files.
2. Generation Phase: Another LLM instance takes those summaries and generates the actual code.
3. Accessibility Evaluation: We put the code to the test using AChecker and QualWeb.
4. Optional Refinement: FeedA11y takes the accessibility feedback and refines the code in an iterative manner.

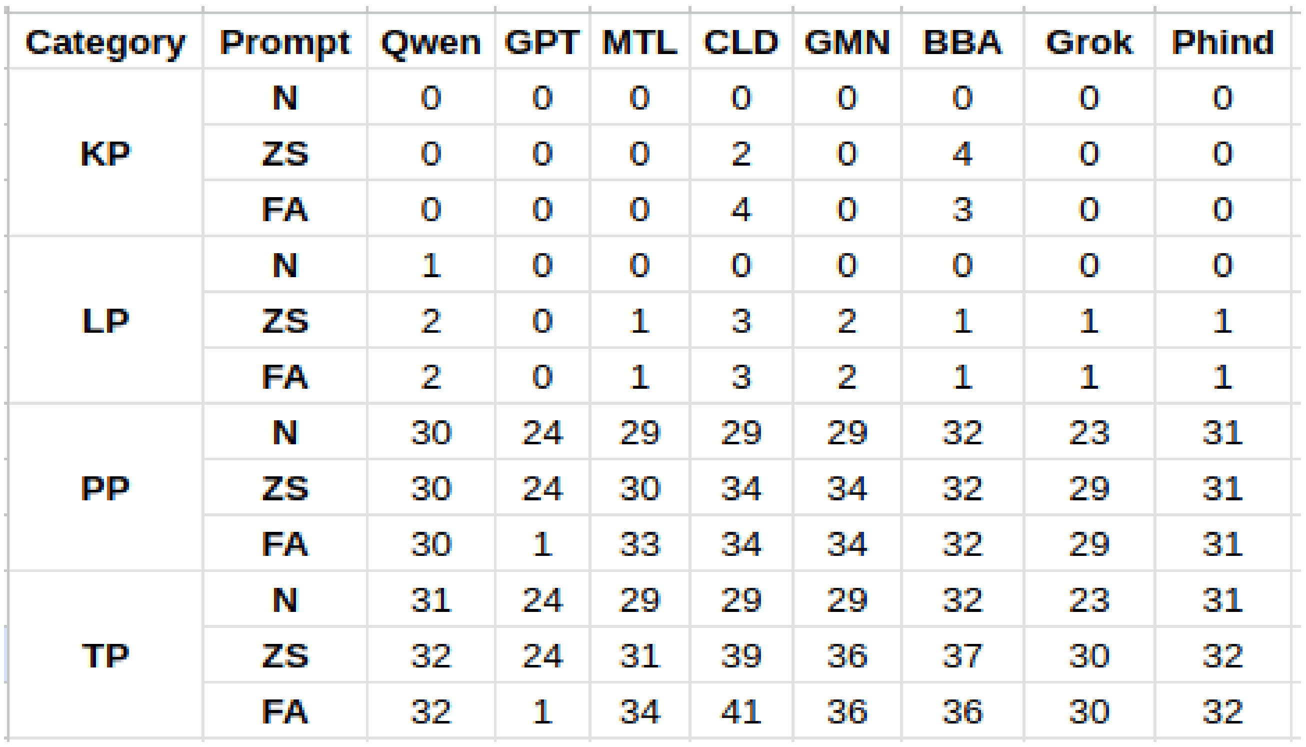
We focus on block-level generation—like HTML sections, JavaScript functions, and CSS rules—to enhance rendering accuracy and pinpoint any errors.

3.5 Tooling and Automation

- Accessibility Evaluators: AChecker.
- Rendering Checks: Headless browser comparison (like Puppeteer)
- Linting Tools: ESLint, Prettier, CSSLint
- Environment: We'll keep prompt and generation logs for reproducibility. All LLM APIs will be accessed using standard settings (temperature, max tokens).

4. Result

Our empirical analysis measured the performance of eight of the top Large Language Models (LLMs) for web code generation on accessibility, in terms of total "known," "likely," and "potential" issues. As Figure 1 shows, total issues by LLM and prompting strategy, enormous differences were seen. ChatGPT, and especially when using the FeedA11y feedback loop, had record-breaking performance, with a paltry 1 total issue. This phenomenal reduction from its baseline (24 issues, Naive/Zero-Shot) illustrates the revolution in performance that can be gained from iterative improvement for highly advanced LLMs. By contrast, Grok had the highest intrinsic capacity, which reached the lowest level of issues (23) with Naive prompting and showed consistently low performance for all strategies, suggesting a high baseline for accessible code generation.



Category	Prompt	Qwen	GPT	MTL	CLD	GMN	BBA	Grok	Phind
KP	N	0	0	0	0	0	0	0	0
	ZS	0	0	0	2	0	4	0	0
	FA	0	0	0	4	0	3	0	0
LP	N	1	0	0	0	0	0	0	0
	ZS	2	0	1	3	2	1	1	1
	FA	2	0	1	3	2	1	1	1
PP	N	30	24	29	29	29	32	23	31
	ZS	30	24	30	34	34	32	29	31
	FA	30	1	33	34	34	32	29	31
TP	N	31	24	29	29	29	32	23	31
	ZS	32	24	31	39	36	37	30	32
	FA	32	1	34	41	36	36	30	32

Table 1: Total accessibility problems identified in web code generated by various LLMs across Naive, Zero-Shot, and FeedA11y prompting strategies. Lower values indicate better accessibility performance.
Here in Table 1, N is Naïve code, ZS is Zero Shot code, FA is Feed Ally code, KP is Known Problem, LP is Likely Problem, PP is Potential Problem and TP is Total Accessibility Problems.

The impact of prompting strategies differed widely across LLMs. While FeedA11y significantly assisted

ChatGPT, with a drastic decrease in accessibility errors, in the other models, its impact was significantly less, and in several instances, even negative. Claude and Blackbox AI stood out by exhibiting a surprising increase in total problems under Zero-Shot and FeedA11y prompting over their Naive performance (e.g., Claude from 29 to 41 problems), which may reflect a difficulty in correctly interpreting or solving intricate accessibility constraints from direct instruction. Qwen, Mistral, Gemini, and Phind maintained low variability in problem counts across the different prompting strategies, reflecting a more uniform, but less adaptive, performance profile.

Lastly, a grouping of problem types was found to uncover the fact that "Potential Problems" always represented the overwhelming majority of problems found in nearly all LLMs, regardless of prompting strategy. This pervasive problem indicates a general shortcoming of LLMs to fully foresee and avoid the insidious and usually subtle accessibility issues that automated tools detect as "potential" issues. Whereas ChatGPT excelled by eliminating nearly all "Known" and "Likely" problems, the pervasive occurrence of "Potential Problems" across the board indicates a common area of future LLM research in the area of inclusive web code generation.

5. Limitations

While we conducted a thorough evaluation, there are a few limitations to this study that we should acknowledge:

1. Static Evaluation Only: We relied on automated tools like AChecker, which can’t really assess dynamic or behavioral accessibility issues, such as keyboard traps or modals.
2. Single Task Domain: Our focus was strictly on HTML/CSS-based UI code, so we haven’t tested how these findings might apply to other areas like mobile or backend code.
3. Prompt Dependency: The performance of the model can change with even slight variations in the prompts. This evaluation is based on fixed, idealized prompt formats.
4. Limited Human Evaluation: Although we gathered quantitative metrics, we didn’t conduct extensive assessments involving developers or users.
5. Black-box LLM APIs: We treated all models as black boxes, without considering the internal differences in how they handle context or tokenization, which could influence their performance.

6. Conclusion

This study performed a complete assessment of the eight most prominent Large Language Models (LLMs) regarding their ability to render code for webpages. Using a modified version of the FeedA11y framework, we examined how different prompting methods from simple to signature feedback loops impacted each model's accessibility performance.

As expected, we concluded that chat GPT consistently surpassed their peers in generating HTML documents due to their greater adherence to web standards. The advanced models, like ChatGPT, tended to show the most pronounced enhancement in accessibility outcomes with the assistance of the FeedA11y feedback loop, but still had severe difficulty navigating sophisticated accessibility problems such as the correct implementation of semantic ARIA roles and landmarks, even with step-by-step guidance provided. In our case, the primary contribution of the work is the prompt formulation and the subsequent re-prompting in the attempts to maximize turning accessibility compliance into a checklist. It raises the question of why some models do not accept step-by-step guidance on basic tasks such as inclusive website design.

7. **Refrences**

1. H. Suh, M. Tafreshipour, S. Malek, and I. Ahmed, "Human or LLM? A comparative study on accessible code generation capability," arXiv preprint arXiv:2503.15885, 2025.
2. A. Anand, S. Chopra, and M. Arora, "Analysis of LLM code synthesis in software productivity," SCRS,2025.[Online].Available: https://www.publications.scrs.in/uploads/final_menuscript/3cba07c8efc40a1a583eb0d09e117f20.pdf
3. Y. Cai et al., "Low-code LLM: Graphical user interface over large language models," arXiv preprint arXiv:2304.08103, 2024.
4. R. Chew et al., "LLM-assisted content analysis: Using large language models to support deductive coding," arXiv preprint arXiv:2306.14924, 2023.
5. A. Liusie, P. Manakul, and M. J. F. Gales, "LLM comparative assessment: Zero-shot NLG evaluation through pairwise comparisons using large language models," arXiv preprint arXiv:2307.07889, 2023.
6. A. Agarwal et al., "Copilot evaluation harness: Evaluating LLM-guided software programming," arXiv preprint arXiv:2402.14261, 2024.
7. F. Lin, D. J. Kim, and T. Chen, "SOEN-101: Code generation by emulating software process models using large language model agents," arXiv preprint arXiv:2403.15852, 2024.
8. F. Liu et al., "Exploring and evaluating hallucinations in LLM-powered code generation," arXiv preprint arXiv:2404.00971, 2024.
9. A. S. Choi et al., "The LLM effect: Are humans truly using LLMs, or are they being influenced by them instead?" arXiv preprint arXiv:2410.04699, 2024.
10. D. Etsenake and M. Nagappan, "Understanding the human-LLM dynamic: A literature survey of LLM use in programming tasks," arXiv preprint arXiv:2410.01026, 2024.
11. Q. Wang et al., "What limits LLM-based human simulation: LLMs or our design?" arXiv preprint arXiv:2501.08579, 2025.
12. T. Yu et al., "Aligning multimodal LLM with human preference: A survey," arXiv preprint arXiv:2503.14504, 2025.
13. S. Lee et al., "Explore, select, derive, and recall: Augmenting LLM with human-like memory for mobile task automation," arXiv preprint arXiv:2312.03003, 2024.
14. H. Xiao et al., "LLM A*: Human-in-the-loop large language models enabled A* search for robotics," arXiv preprint arXiv:2312.01797, 2025.
15. J. Li et al., "Dissecting human and LLM preferences," arXiv preprint arXiv:2402.11296, 2024.
16. B. Abeysinghe and R. Circi, "The challenges of evaluating LLM applications: An analysis of automated, human, and LLM-based approaches," arXiv preprint arXiv:2406.03339, 2024.
17. P. D. Joshi et al., "HULLMI: Human vs LLM identification with explainability," arXiv preprint arXiv:2409.04808, 2024.
18. E. Artemova et al., "Hands-on tutorial: Labeling with LLM and human-in-the-loop," arXiv preprint arXiv:2411.04637, 2024.
19. S. Musker et al., "LLMs as models for analogical reasoning," arXiv preprint arXiv:2406.13803, 2024.
20. Q. Pan et al., "Human-centered design recommendations for LLM-as-a-judge," arXiv preprint arXiv:2407.03479, 2024.
21. H. Leung and Z. Wang, "LLM should think and act as a human," arXiv preprint arXiv:2502.13475, 2025.
22. V. K. Kommineni, B. König-Ries, and S. Samuel, "From human experts to machines: An LLM suppo

rted approach to ontology and knowledge graph construction," arXiv preprint arXiv:2403.08345, 2024.

23. N. McAleese et al., "LLM critics help catch LLM bugs," arXiv preprint arXiv:2407.00215, 2024.
24. R. Movva, P. W. Koh, and E. Pierson, "Annotation alignment: Comparing LLM and human annotations of conversational safety," arXiv preprint arXiv:2406.06369, 2024.
25. A. Elangovan et al., "Beyond correlation: The impact of human uncertainty in measuring the effectiveness of automatic evaluation and LLM-as-a-judge," arXiv preprint arXiv:2410.03775, 2024.
26. Y. Shao et al., "Character-LLM: A trainable agent for role-playing," arXiv preprint arXiv:2310.10158, 2023.
27. Y. R. Dong, T. Hu, and N. Collier, "Can LLM be a personalized judge?" arXiv preprint arXiv:2406.11657, 2024.
28. J. J. Sanchez-Medina, "Sentiment analysis and random forest to classify LLM versus human source applied to scientific texts," arXiv preprint arXiv:2404.08673, 2024.
29. L. Ibrahim et al., "Towards interactive evaluations for interaction harms in human-AI systems," arXiv preprint arXiv:2405.10632, 2024.
30. G. H. Chen et al., "Humans or LLMs as the judge? A study on judgement biases," arXiv preprint arXiv:2402.10669, 2024.