AI-Assisted Accessible Web Interface Generation and WCAG Validation: A Review of LLM-Based Code Generation, Automated Auditing, Repair, and Human Oversight

Sinthiya Dongre
UG Student
Dept. of Computer Science and Engineering (Data Science)
Guru Nanak Institute of Technology
Nagpur, India
dongresinthiya@gmail.com

Vedashri Banpurkar
UG Student
Dept. of Computer Science and Engineering (Data Science)
Guru Nanak Institute of Technology
Nagpur, India
vedashribanpurkar644@gmail.com

Tejal Mankar
UG Student
Dept. of Computer Science and Engineering (Data Science)
Guru Nanak Institute of Technology
Nagpur, India
tejalmankar200@gmail.com

Kartik Chacherkar
Project Guide
Dept. of Computer Science and Engineering (Data Science)
Guru Nanak Institute of Technology
Nagpur, India
Kartikchacherkar866@gmail.com

***Abstract*—**Most websites still fail basic accessibility checks, and tools that write web pages with AI are now common. If these tools copy the habits of the code they learned from, they may repeat the same barriers at a much larger scale. This review looks at research on AI-assisted generation of accessible web interfaces and their validation against the Web Content Accessibility Guidelines (WCAG). We group the work into eight themes: the state of web accessibility and its standards, the limits of automated checkers, large language models (LLMs) as web code generators, design-to-code and visual models, LLM-based detection of WCAG problems, automated repair, alternative text and visual content, and mobile interfaces with human oversight and governance. We summarise thirty-five studies, the methods they used, the results their authors report, and the limits they admit. We then compare the approaches and benchmarks, and we describe a generate, validate and repair pipeline as a reference architecture. LLMs handle simple issues such as colour contrast and missing alternative text fairly well, but they struggle with ARIA, keyboard behaviour and checks that depend on meaning. Automated tools alone also miss many real barriers. We end with open problems and directions for future work.

***Keywords*—**Web accessibility, WCAG, large language models, accessible code generation, automated accessibility testing, ARIA, alternative text, accessibility repair, design-to-code.

I. Introduction

The web is now the main way to reach services such as banking, education, health care and government support. About 16% of the world's population lives with some form of disability [1], so a page that cannot be used with a screen reader, a keyboard or a magnifier shuts out a large group of people. The Web Content Accessibility Guidelines (WCAG) are the shared standard for avoiding this [2], and many regions now ask for accessible digital services in law or policy, for example the European Accessibility Act [3], India's Rights of Persons with Disabilities Act [4] and the guidelines for Indian government websites [5]. Even so, a yearly analysis of one million home pages found detectable WCAG failures on 94.8% of them in 2025 [6].

At the same time, large language models (LLMs) have become part of everyday web development. Models built on the transformer design [7], such as GPT-3 [8], Codex [9] and GPT-4 [10], can write HTML, CSS and JavaScript from a short description, and multimodal versions can turn a screenshot into a page [11]. This saves time, but it raises a simple worry. These models learn from public code, and most public pages are not accessible. Early evidence supports the worry: when developers asked ChatGPT to build sites without mentioning accessibility, most of the sites had violations [12]. Generated code can also be weak in other ways, such as security [13], which is a reminder that code from a model needs checking before it is trusted.

The same models can also read a page, explain a violation and suggest a fix, and several recent tools use them to find and repair WCAG problems [14], [15], [16]. The research is spread over software engineering, human-computer interaction and machine learning venues, and earlier surveys usually treat accessibility testing and AI code generation separately. This paper puts the two together. It reviews thirty-five studies, most of them from 2020 to 2026, with three aims: (i) to organise the work into a simple taxonomy; (ii) to compare methods, benchmarks and reported results; and (iii) to point out open problems. A generate, validate and repair pipeline is used as a reference case.

A. Review Methodology

The review follows the planning, conducting and reporting steps used for structured reviews in software engineering [17]. We searched Google Scholar, the ACM Digital Library, IEEE Xplore and arXiv with terms such as web accessibility, WCAG, large language model, accessible code generation, automated accessibility testing, alt text and design-to-code. A paper was kept if it fitted at least one theme of the taxonomy, reported an evaluation, and had a source that can be checked. The thirty-five core studies in Tables I to III were read in full. They come from venues such as CHI, ASSETS, ICSE, ISSTA, FSE, W4A, NAACL, IEEE Access and *Universal Access in the Information Society*, together with a few preprints, which are marked as such in the reference list. Standards, tools and foundation papers add to the list. Every number quoted for a study is the one its authors report, and the limits in the tables are those the authors state or, where none is stated, those that follow from the scope of the study. The studies use different datasets, tools and WCAG versions, so the numbers cannot be compared directly, and the review is a narrative synthesis, not a meta-analysis.

B. Contributions

1) An eight-theme taxonomy of AI-assisted accessible interface generation and WCAG validation (Fig. 1).
2) A thematic synthesis of thirty-five studies, with the results and limits reported by the authors (Tables I–III).
3) Comparisons of the main approaches (Table IV) and of the datasets and benchmarks used (Table V).
4) A reference generate–validate–repair architecture (Fig. 2, Table VI), together with open challenges and research directions.

II. Background

A. WCAG, ARIA and the Legal Setting

WCAG is organised around four principles: content must be perceivable, operable, understandable and robust. Each principle has guidelines, and each guideline has testable success criteria at level A, AA or AAA. Version 2.2 adds criteria on target size, focus visibility and accessible login to version 2.1 [2], [18]. WAI-ARIA adds roles and states so that scripted widgets such as menus and dialogs can be understood by assistive technology [19]. It helps only when it is used correctly, and wrong ARIA can leave a page worse than plain HTML. In Europe the technical rules sit in EN 301 549 [20] and are tied to law by the European Accessibility Act [3]. Lazar [21] gives an overview of web accessibility policy and law. For checking a whole site, W3C describes a conformance evaluation method called WCAG-EM [22].

B. How Accessibility Is Checked Today

Most teams begin with rule-based checkers such as axe-core [23] and the IBM Equal Access Accessibility Checker [24]. These tools read the page structure and test rules that code can decide, for example whether an image has an alt attribute, whether a form field has a label, or whether text contrast passes the threshold. They are fast and can run in a build pipeline. What they cannot decide is whether an alt text is *useful*, whether a heading order makes sense, or whether a person can finish a task with only a keyboard. Those points need manual checks with a keyboard and a screen reader, as WCAG-EM describes [22], and ideally tests with disabled users.

C. Generative AI for Interface Creation

LLMs generate code by predicting the next token, and the quality of the result depends a lot on the prompt. Few-shot examples [8], step-by-step reasoning [25] and reason-then-act loops such as ReAct [26] are used to get better output. Retrieval-augmented generation [27] can place the exact WCAG wording in the prompt. Vision transformers [28] allow models to read screenshots, and an LLM can also act as a judge of another model's output [29]. The main risk is hallucination: a model can state something wrong in a confident tone [30]. For accessibility this means a made-up fix or a wrong image description can look fine while still hurting the user.

III. Taxonomy

Fig. 1 shows the eight themes. They run from the problem itself (the state of the web and the limits of automated checkers), through AI for producing interfaces (code generation, design-to-code) and for checking and fixing them (detection, repair, alternative text), to the setting around the tools (mobile interfaces, human oversight and governance).

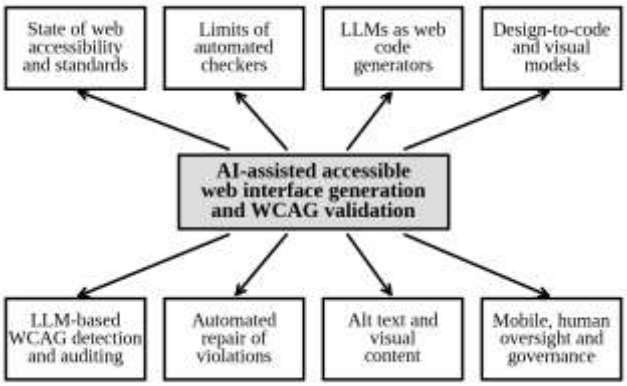


Fig. 1. Taxonomy of AI-assisted accessible web interface generation and WCAG validation.

IV. Thematic Review

A. State of Web Accessibility and Standards

Accessibility failures are common across sectors and countries. The yearly WebAIM analysis found failures on 94.8% of one million home pages in 2025 [6]. Sector studies tell the same story. Campoverde-Molina *et al.* [31] reviewed empirical studies on educational websites and reported that conformance problems were common. University sites in Kuwait [32] and in Spain, Chile and Mexico [33], mobile government sites in Saudi Arabia [34], Indian e-government sites [35] and Korean health-care sites [36] all show barriers; the Korean study tested with visually impaired users and not only with tools. Inside software projects, Bi *et al.* [37] found that about 70% of the GitHub projects they studied had reported accessibility issues, mostly about colour use and layout. Alshayban *et al.* [38] studied 1,000 Android apps together with developers' views and showed that issues are widespread. Mack *et al.* [39] looked at accessibility papers in CHI and ASSETS from 1994 to 2019 and describe how the field has grown. These studies matter here because the code a model learns from reflects this state of the web.

B. Limits of Automated Checkers

Checkers are the base of most validation, but research shows how much they miss. Brajnik *et al.* [40] studied the validity and reliability of WCAG 2.0 evaluation and found that conformance judgements were not stable: evaluators missed real problems and also reported some that were not real. Vigo *et al.* [41] compared several tools on the same pages and warned against relying on automated tests alone, because the tools differ in coverage and correctness. Power *et al.* [42] watched 32 blind users on real sites and found that only about half of the problems they met were covered by WCAG 2.0 success criteria. More recently, Tafreshipour *et al.* [43] built Ma11y, which injects 25 kinds of accessibility faults into pages, and showed that current tools miss nearly half of the injected bugs. A generated page that passes one tool is not proven accessible, and a repair loop that only chases tool warnings can be no better than the tool.

C. LLMs as Web Code Generators

Aljedaani *et al.* [12] were among the first to test this. Eighty-eight web developers asked ChatGPT (GPT-3.5) for websites in plain language, without asking for accessible code. Only 14 of the 88 sites had no violations, which means 84% had at least one. When the developers then asked ChatGPT to fix the problems, it repaired about 70% of the violations in its own sites (806 of 1,153) and about 73% in open-source projects

from GitHub. Abu Doush and Kassem [44] tested ChatGPT 4o, Copilot Pro, Claude 3.7 Sonnet and Grok 3 on eleven WCAG 2.1 components, checking the results by hand with a keyboard and a screen reader. All models wrote valid markup, but most needed follow-up prompts and human checks to reach full compliance; Claude 3.7 Sonnet and ChatGPT 4o did best. Othman *et al.* [45] reported a case study in which ChatGPT was used to remediate the accessibility of existing pages.

Suh *et al.* [46] used a different design. They regenerated the interface files of ten real web projects with GPT-4o and Qwen2.5-Coder and measured the share of elements with violations. With the IBM checker, plain generation scored 0.347 and 0.348, against 0.425 for the original human code, so the models did slightly better than the human authors. Contrast problems made up 80% of all violations, and GPT-4o cut them by nearly half and cut alt-text problems by over 70%. But both models did worse than humans on ARIA-related rules. Zero-shot, few-shot and self-criticism prompts did not beat plain generation, and few-shot was the worst at 0.449. This does not conflict with the 84% figure above: one study counts sites with *any* violation, the other counts the share of *elements* with a violation, so they measure different things.

D. Design-to-Code and Visual Models

A second route to a web page is to start from an image. pix2code [47] was an early attempt, using a convolutional and a recurrent network to produce interface code from a screenshot; it reported accuracy above 77% on synthetic interfaces for three platforms, but the designs were simple. WebSight [48] supplied about 0.8 million synthetic page and code pairs, and WebCode2M [49] and Web2Code [50] followed with real or mixed webpages for training and testing. Design2Code [11] built a benchmark of 484 real webpages and showed how far multimodal models still were from reliable screenshot-to-code work; its test pages have about 158 HTML tags on average, against 19 in WebSight. Pix2Struct [51] and ScreenAI [52] pretrain on screenshots and interface data to help models read screens, and Rico [53] gave the field a large collection of mobile screens.

None of these works treats accessibility as the goal. Design2Code, for example, scores how close the rendered page looks to the target screenshot [11], and as far as we could tell the other datasets report visual and structural similarity as well. A page can therefore score very well and still have no heading structure, no labels and no keyboard support. This gap is one reason why accessibility needs its own validation step after generation.

E. LLM-Based Detection and Auditing

Rule-based tools cover only a part of WCAG, so several groups use LLMs to widen the check. López-Gil and Pereira [54] turned success criteria that are normally checked by hand into automatic checks with an LLM. He *et al.* [14] built GenA11y: for each testable success criterion it pulls the related elements out of a page and asks an LLM to find problems. It reached 94.5% precision and 87.61% recall, and on real sites it found on average eight more kinds of violation than a combination of existing tools. AccessGuru [15] joins existing testing tools with LLM prompts and sorts violations into syntactic, layout and semantic types. Calò *et al.* [55] looked at 300 interfaces written by three commercial models and counted 541 semantic violations of six fault types; they checked an LLM-as-judge by injecting faults and got recall between 80% and 92%. These results are encouraging, but an LLM judge can be wrong or invent a problem [30], [29], so its findings still need a second check.

F. Automated Repair of Violations

Repair is harder than detection, because a fix must keep the page working. Huang *et al.* [16] changed the page DOM in real time with an LLM and prompt engineering and reported more than a 51% drop in violation errors on their ACCESS benchmark. AccessGuru [15] used prompts matched to the violation type and reported an average drop of about 84% in the violation score. WebAccessVL [56] takes a different path: a vision-language model gets both the HTML and its rendering, is conditioned on the number of WCAG 2 violations, and learns to edit the HTML from a dataset of manually corrected pages. FeedA11y [46] feeds checker reports back to the model in a ReAct loop. It had the lowest inaccessibility rate in its study (about 0.29 to 0.30 with the IBM checker), needed 2.4 rounds on average, and was capped at three rounds. The authors also report that general instructions such as “make this accessible” sometimes created new problems because added attributes clashed with the existing code. Targeted feedback worked better than broad advice.

G. Alternative Text and Visual Content

Missing or poor image descriptions are a long-standing barrier. Gleason *et al.* [57] showed how often user-provided descriptions were absent on Twitter. Kreiss *et al.* [58] argued that a good description depends on context, which makes automatic scoring of captions without human references hard. Mohanbabu and Pavel [59] built a Chrome extension that gives GPT-4V the page text, title and URL along with the image. In a study with 12 blind and low-vision participants, the context-aware descriptions were often preferred and rated higher for quality, imaginability, relevance and plausibility. On mobile, Li *et al.* [60] created a widget-captioning task, Chen *et al.* [61] trained a deep model to predict labels for image buttons that have none, and Haque and Csallner [62] fine-tuned an LLM on about 1.4 thousand icons using the icon’s class, bounds, recognised text and neighbouring nodes. Screen Recognition [63] went one step further and built accessibility metadata for mobile apps from pixels, using a model trained on tens of thousands of iOS screens. A wrong description is worse than none for a screen-reader user, so these tools work best when a person can review the text.

H. Mobile Interfaces, Human Oversight and Governance

Testing on mobile shows where AI helps with the checking process itself. Latte [64] uses assistive services such as a screen reader to run use cases, and its authors report that it completed more use cases and exposed more accessibility problems than earlier automated tools. AXNav [65] lets testers write accessibility test steps in plain language and replays them on an app, and it was evaluated with accessibility testing professionals. Both tools support a human tester; neither replaces one. The same view appears in the evidence from users. Power *et al.* [42] and Yi [36] both found barriers that only tests with disabled people revealed.

Governance follows from this. A claim of conformance needs evidence, and WCAG-EM [22] describes the steps for gathering it. Laws such as the European Accessibility Act [3] place the duty on the provider, not on the tool. Code from an AI assistant should be treated like any other third-party code, the way work on Copilot security suggests [13]: checked, logged and reviewed. Teams that publish AI-generated pages should record the model, the prompt, the checker versions and the manual tests that were done.

TABLE I
SUMMARY OF REVIEWED STUDIES ON LLMS FOR ACCESSIBLE WEB CODE, AUDITING, REPAIR AND ALT TEXT

Study	Method and reported result	Limit
Aljedaani *et al.* [12]	88 developers prompted ChatGPT; 84% of sites had violations; about 70% of violations fixed	GPT-3.5 only; two tools disagree
Suh *et al.* [46]	GPT-4o, Qwen2.5-Coder on 10 projects; rate 0.347 vs 0.425 human; FeedA11y about 0.30	ARIA rules still weak
Abu Doush and Kassem [44]	4 chatbots, 11 WCAG 2.1 components; manual keyboard and screen reader tests	Needs follow-up prompts and human checks
Othman *et al.* [45]	Case study of ChatGPT for automatic remediation of pages	Single case; no general claim
López-Gil and Pereira [54]	LLM turns manual success criteria into automatic checks	Limited to chosen criteria
He *et al.* [14]	GenA11y; 94.5% precision, 87.61% recall; 8 more violation types than tools	Depends on prompts and extracted elements
Calò *et al.* [55]	300 UIs from 3 models; 541 semantic violations; LLM judge recall 80–92%	Short paper; small human study
Fathallah *et al.* [15]	AccessGuru: tools + LLM prompts; about 84% drop in violation score	Needs human review of fixes
Huang *et al.* [16]	ACCESS: DOM edits by LLM; over 51% fewer violations	Preprint; benchmark scope
Zheng *et al.* [56]	Violation-conditioned VLM edits HTML using page image	Preprint; trained on one dataset
Mohanbabu and Pavel [59]	Context-aware GPT-4V alt text; 12 BLV users preferred it	Small user study; risk of wrong detail
Haque and Csallner [62]	Fine-tuned LLM labels UI icons using context; about 1.4k icons	Small training set; preprint

TABLE II
SUMMARY OF REVIEWED STUDIES ON DESIGN-TO-CODE, VISUAL MODELS AND MOBILE GUI ACCESSIBILITY

Study	Method and reported result	Limit
Beltramelli [47]	CNN + LSTM from screenshot to code; above 77% accuracy	Simple synthetic UIs
Laurençon *et al.* [48]	WebSight: about 0.8M synthetic page–code pairs	Synthetic, simple pages
Gui *et al.* [49]	WebCode2M real-page dataset; ViT-based WebCoder baseline	Visual match, not WCAG
Si *et al.* [11]	Design2Code: 484 real pages; 5 automatic similarity metrics	Scores looks, not accessibility
Yun *et al.* [50]	Web2Code: webpage-to-code data and evaluation framework	Not accessibility-focused
Lee *et al.* [51]	Pix2Struct: pretraining on screenshot parsing	General visual language task
Baechler *et al.* [52]	ScreenAI: vision-language model for UI and infographics	No WCAG evaluation
Zhang *et al.* [63]	Screen Recognition: metadata from pixels; tens of thousands of iOS screens	iOS only; fixed element types
Chen *et al.* [61]	Deep model predicts labels for unlabelled image buttons	Android; label quality varies
Taeb *et al.* [65]	AXNav: plain-language test steps replayed on app; evaluated with accessibility testers	Needs human tester
Salehnamadi *et al.* [64]	Latte: runs use cases through assistive services	Android only
Alshayban *et al.* [38]	1,000 Android apps plus developer views; issues widespread	Apps only, not web
Bi *et al.* [37]	GitHub projects; about 70% reported accessibility issues	Based on reported issues

TABLE III
SUMMARY OF REVIEWED STUDIES ON CHECKER VALIDITY AND THE STATE OF THE WEB

Study	Method and reported result	Limit
Brajnik *et al.* [40]	Validity and reliability of WCAG 2.0 judgements by evaluators	WCAG 2.0 era
Vigo *et al.* [41]	Compared automated tools on the same pages; warned against sole reliance	Older tools
Power *et al.* [42]	32 blind users; about half of problems not covered by WCAG 2.0	Small user group
Tafreshipour *et al.* [43]	Ma11y: 25 fault operators; tools miss nearly 50%	Injected, not natural, faults
Campoverde-Molina *et al.* [31]	Review of studies on educational websites	Depends on the studies included
Paul and Das [35]	Accessibility and usability of Indian e-government sites	Sample of sites only
Al-Sakran and Alsudairi [34]	Saudi mobile e-government websites assessed	One country
AlMeraj *et al.* [32]	Higher education sites in Kuwait evaluated	One country and sector
Máñez-Carvajal *et al.* [33]	Top university sites in Spain, Chile, Mexico	Top-ranked sites only
Yi [36]	Korean health-care sites; user test with visually impaired people	One sector; one user group

V. Comparative Analysis

Table IV compares the main families of approaches discussed above, and Table V lists the datasets and benchmarks that appear in the reviewed work. The comparison is qualitative. It rests on what the original authors report and is not a new experiment.

TABLE IV
COMPARISON OF APPROACHES

Approach	Strengths	Weaknesses
Rule-based checkers [23], [24]	Fast; repeatable; fits a build pipeline	Covers only rules that code can decide [43], [41]
Plain LLM generation [12], [46]	Quick; handles contrast and alt text fairly well	Weak on ARIA; most sites still had violations
Prompted accessibility rules [46]	Easy to try	Did not beat plain generation; can add clashes
Checker feedback loop [46], [16]	Targeted fixes; best score in its study	Only as good as the checker; extra rounds
LLM detection [14], [54], [55]	Finds meaning-based issues tools miss	May miss or invent issues [30]
Vision-language repair [56]	Uses both code and rendered page	Needs corrected training data
Context-aware alt text [59]	Preferred by blind and low-vision users	Wrong detail is harmful
User and expert testing [42], [22]	Finds real barriers	Slow and costly

TABLE V
DATASETS AND BENCHMARKS IN THE REVIEWED WORK

Resource	Type	Notes
WebSight [48]	Page–code pairs	About 0.8M synthetic pages
WebCode2M [49]	Page–code pairs	Real webpages; ViT baseline
Design2Code [11]	Benchmark	484 real pages; similarity metrics
Web2Code [50]	Dataset + framework	Webpage-to-code for MLLMs
Rico [53]	Mobile screens	Large Android UI collection
ACCESS [16]	Violation + fix	Built for web accessibility repair

Resource	Type	Notes
WebAccessVL data [56]	Violation + fix	Manually corrected pages
Ma11y mutants [43]	Injected faults	25 operators to test tools

Four patterns appear. First, models do well on simple, rule-like issues. Contrast and alt text improve in generated code [46], while ARIA, keyboard behaviour and focus stay hard [46], [44], [55]. Second, results depend on how accessibility is measured. The share of sites with any violation [12] and the share of elements with a violation [46] can point in different directions, and tools disagree with each other [41], [43]. Third, feedback beats instruction. Prompts that only say "be accessible" gave little [46], while loops that return checker reports gave more [46], [16], [15]. Fourth, the evidence from real users is thin. Only a few of the studies involve disabled participants [59], [36], [42], and we did not find a study that measures whether an AI-generated page is easier for them to use in a full task. For a practical system these patterns favour a layered design: generate, check with more than one tool, add LLM checks for meaning, repair with targeted feedback, and finish with a human review.

VI. Reference Case: A Generate–Validate–Repair Pipeline

To show how the themes fit together, we describe a reference architecture. It is a design based on the reviewed work. It is not an implemented system, and this paper reports no measured results for it. Fig. 2 shows the flow and Table VI maps each review theme to a module.

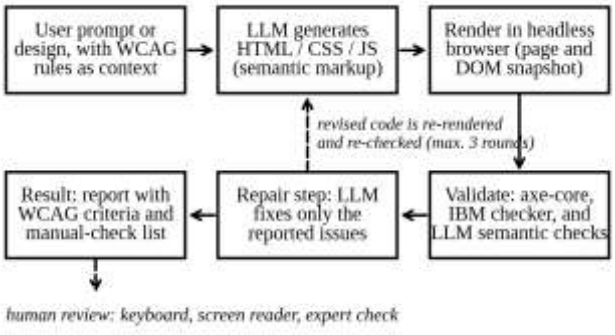


Fig. 2. Reference generate–validate–repair pipeline for accessible web interfaces.

TABLE VI
MAPPING OF REVIEW THEMES TO THE REFERENCE PIPELINE

Theme	Pipeline module	Key technique
State of the web and standards	Prompt context	WCAG 2.2 criteria placed in the prompt [2], [27]
Limits of checkers	Validation	More than one tool; manual checklist [43], [22]
LLM code generation	Generator	Semantic HTML first; ARIA only when needed [46]
Design-to-code	Optional input	Screenshot or mockup to page [11]
LLM detection	Semantic checks	Per-criterion prompts [14]
Automated repair	Repair step	Fix reported issues only; 3 rounds [46]
Alt text	Media module	Page context in prompt; human approval [59]
Oversight and governance	Report	WCAG list, model and tool versions logged [22]

The flow works in this order. A user gives a prompt or a design, and the prompt carries the relevant WCAG criteria as context. The LLM writes HTML, CSS and JavaScript, with a preference for native elements such as button, nav and label before any ARIA. The page is rendered in a headless browser and checked with a rule-based tool such as axe-core [23]. A second tool, such as the IBM checker [24], gives an independent view, and LLM prompts check the points that rules cannot, for instance whether alt text matches the image and whether link text makes sense. Violations go back to the LLM, which is told to change only what was reported. The loop stops after three rounds or when no new change is made, as in FeedA11y [46]. The last output is a short report listing the criteria that passed, the criteria that failed, and a manual checklist for keyboard, focus and screen-reader tests.

Two design notes come from the reviewed studies. First, the tool used to guide repair should not be the only tool used to judge the result. Suh *et al.* [46] used rules from one checker for repair and the other for scoring, and the same split avoids a pipeline that learns to please its own checker. Second, a passing score is not a conformance claim [40], [42]. The report should say what was checked automatically and what still needs a human.

VII. Open Challenges and Research Directions

- **Beyond simple rules.** Models fix contrast and alt text but are weak on ARIA and keyboard behaviour [46], [44]. Better training data and tests for interactive widgets are needed.
- **Trustworthy measurement.** Different metrics and tools give different pictures [41], [43], [12], [46]. Shared benchmarks that combine several tools with manual results would make studies comparable.
- **Accessibility in design-to-code.** Current benchmarks reward visual similarity [11]. Accessibility scores should be added to them.
- **Reliable LLM judges.** Fault-injection tests give recall of 80–92% [55], which still leaves misses and possible false alarms [30]. Calibration against expert reviews is needed.
- **Safe automatic repair.** Fixes must not break the page or add new violations [46]. Regression tests and visual diffs should be part of repair.
- **Context-aware media text.** Alt text depends on context [58], [59]. Methods are needed to detect when a description is wrong or unsafe to publish.
- **Evidence with disabled users.** Studies that include screen-reader, keyboard and low-vision users on full tasks are rare [42], [36]. Such studies are the missing test for every tool in this review.
- **Governance and reporting.** Providers need clear logs of the model, prompt, tool versions and manual tests [22], [3].

VIII. Conclusion

Across the thirty-five studies reviewed here, AI helps at each stage of building an accessible page, but with a clear limit. Models are good at simple issues and weak at ARIA, keyboard use and anything that depends on meaning. Automated tools miss a large share of real barriers, so a page that passes them is not necessarily usable. Feedback loops that return checker reports work better than general instructions, and context makes image descriptions better. Evidence from disabled users is still scarce. A layered process that generates, checks with more than one tool, repairs with targeted feedback and ends with human review is the safest use of these tools today. For the reference pipeline, the next steps are to build it, test it on real projects against several checkers, add manual tests with

screen-reader and keyboard users, and report violations found, violations fixed, new violations created and time spent. No tool guarantees an accessible page. A checked, logged process at least keeps generated code from copying the barriers already on the web.

REFERENCES

[1] World Health Organization, "Global report on health equity for persons with disabilities," WHO, Geneva, Switzerland, 2022, ISBN 978-92-4-006360-0.

[2] World Wide Web Consortium (W3C), "Web Content Accessibility Guidelines (WCAG) 2.2," W3C Recommendation, Oct. 2023. [Online]. Available: https://www.w3.org/TR/WCAG22/

[3] European Parliament and Council of the European Union, "Directive (EU) 2019/882 on the accessibility requirements for products and services," *Off. J. Eur. Union*, vol. L 151, pp. 70–115, Jun. 2019.

[4] Government of India, "The Rights of Persons with Disabilities Act, 2016," Act No. 49 of 2016, Dec. 2016.

[5] National Informatics Centre, Government of India, "Guidelines for Indian Government Websites and Apps (GIGW) 3.0," 2023.

[6] WebAIM, "The WebAIM Million: An accessibility analysis of the top 1,000,000 home pages," 2025. [Online]. Available: https://webaim.org/projects/million/

[7] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 5998–6008.

[8] T. B. Brown *et al.*, "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[9] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv:2107.03374*, 2021.

[10] OpenAI, "GPT-4 technical report," *arXiv:2303.08774*, 2023.

[11] C. Si, Y. Zhang, R. Li, Z. Yang, R. Liu, and D. Yang, "Design2Code: Benchmarking multimodal code generation for automated front-end engineering," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics (NAACL)*, 2025.

[12] W. Aljedaani, A. Habib, A. Aljohani, M. Eler, and Y. Feng, "Does ChatGPT generate accessible code? Investigating accessibility challenges in LLM-generated source code," in *Proc. 21st Int. Web for All Conf. (W4A)*, Singapore, 2024, pp. 165–176, doi: 10.1145/3677846.3677854.

[13] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2022, pp. 754–768.

[14] Z. He, S. F. Huq, and S. Malek, "Enhancing web accessibility: Automated detection of issues with generative AI," *Proc. ACM Softw. Eng.*, vol. 2, no. FSE, pp. 2264–2287, 2025, doi: 10.1145/3729371.

[15] N. Fathallah, D. Hernandez, and S. Staab, "AccessGuru: Leveraging LLMs to detect and correct web accessibility violations in HTML code," in *Proc. 27th Int. ACM SIGACCESS Conf. Comput. Accessibility (ASSETS)*, Denver, CO, USA, 2025.

[16] C. Huang *et al.*, "ACCESS: Prompt engineering for automated web accessibility violation corrections," *arXiv:2401.16450*, 2024.

[17] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ., Keele, U.K., Tech. Rep. EBSE-2007-01, 2007.

[18] A. Kirkpatrick, J. O'Connor, A. Campbell, and M. Cooper, Eds., "Web Content Accessibility Guidelines (WCAG) 2.1," W3C Recommendation, Jun. 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/

[19] World Wide Web Consortium (W3C), "Accessible Rich Internet Applications (WAI-ARIA) 1.2," W3C Recommendation, Jun. 2023. [Online]. Available: https://www.w3.org/TR/wai-aria-1.2/

[20] European Telecommunications Standards Institute, "Accessibility requirements for ICT products and services," ETSI EN 301 549 V3.2.1, Mar. 2021.

[21] J. Lazar, "Web accessibility policy and law," in *Web Accessibility: A Foundation for Research*, 2nd ed., Y. Yesilada and S. Harper, Eds. London, U.K.: Springer, 2019, pp. 247–261.

[22] E. Velleman and S. Abou-Zahra, Eds., "Website Accessibility Conformance Evaluation Methodology (WCAG-EM) 1.0," W3C Working Group Note, Jul. 2014. [Online]. Available: https://www.w3.org/TR/WCAG-EM/

[23] Deque Systems, "axe-core: Accessibility engine for automated web UI testing." [Online]. Available: https://github.com/dequelabs/axe-core

[24] IBM, "IBM Equal Access Accessibility Checker." [Online]. Available: https://github.com/IBMa/equal-access

[25] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 35, 2022, pp. 24824–24837.

[26] S. Yao *et al.*, "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[27] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[28] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[29] L. Zheng *et al.*, "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS) Datasets Benchmarks Track*, 2023.

[30] Z. Ji *et al.*, "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, Art. no. 248, 2023.

[31] M. Campoverde-Molina, S. Luján-Mora, and L. Valverde García, "Empirical studies on web accessibility of educational websites: A systematic literature review," *IEEE Access*, vol. 8, pp. 91676–91700, 2020.

[32] Z. AlMeraj, F. Boujarwah, D. Alhuwail, and R. Qadri, "Evaluating the accessibility of higher education institution websites in the State of Kuwait: Empirical evidence," *Univers. Access Inf. Soc.*, vol. 20, no. 1, pp. 121–138, 2021.

[33] C. Máñez-Carvajal, J. F. Cervera-Mérida, and R. Fernández-Piqueras, "Web accessibility evaluation of top-ranking university web sites in Spain, Chile and Mexico," *Univers. Access Inf. Soc.*, vol. 20, pp. 179–184, 2021.

[34] H. O. Al-Sakran and M. A. Alsudairi, "Usability and accessibility assessment of Saudi Arabia mobile e-government websites," *IEEE Access*, vol. 9, pp. 48254–48275, 2021.

[35] S. Paul and S. Das, "Accessibility and usability analysis of Indian e-government websites," *Univers. Access Inf. Soc.*, vol. 19, no. 4, pp. 949–957, 2020.

[36] Y. J. Yi, "Web accessibility of healthcare web sites of Korean government and public agencies: A user test for persons with visual impairment," *Univers. Access Inf. Soc.*, vol. 19, no. 1, pp. 41–56, 2020.

[37] T. Bi, X. Xia, D. Lo, and A. Aleti, "A first look at accessibility issues in popular GitHub projects," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, 2021, pp. 390–401.

[38] A. Alshayban, I. Ahmed, and S. Malek, "Accessibility issues in Android apps: State of affairs, sentiments, and ways forward," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng. (ICSE)*, Seoul, South Korea, 2020, pp. 1323–1334.

[39] K. Mack, E. McDonnell, D. Jain, L. L. Wang, J. E. Froehlich, and L. Findlater, "What do we mean by "accessibility research"? A literature survey of accessibility papers in CHI and ASSETS from 1994 to 2019," in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2021.

[40] G. Brajnik, Y. Yesilada, and S. Harper, "Is accessibility conformance an elusive property? A study of validity and reliability of WCAG 2.0," *ACM Trans. Access. Comput.*, vol. 4, no. 2, Art. no. 8, 2012.

[41] M. Vigo, J. Brown, and V. Conway, "Benchmarking web accessibility evaluation tools: Measuring the harm of sole reliance on automated tests," in *Proc. 10th Int. Cross-Disciplinary Conf. Web Accessibility (W4A)*, 2013, Art. no. 1.

[42] C. Power, A. Freire, H. Petrie, and D. Swallow, "Guidelines are only half of the story: Accessibility problems encountered by blind users on the web," in *Proc. SIGCHI Conf. Human Factors Comput. Syst. (CHI)*, 2012, pp. 433–442.

[43] M. Tafreshipour, A. Deshpande, F. Mehralian, I. Ahmed, and S. Malek, "Ma1ly: A mutation framework for web accessibility testing," in *Proc. 33rd ACM SIGSOFT Int. Symp. Softw. Testing Anal. (ISSTA)*, Vienna, Austria, 2024, pp. 100–111.

[44] I. Abu Doush and R. Kassem, "Can generative AI create accessible web code? A benchmark analysis of AI-generated HTML against accessibility standards," 2025. [Online]. Available: http://hdl.handle.net/11675/14489

[45] A. Othman, A. Dhouib, and A. N. Al Jabor, "Fostering websites accessibility: A case study on the use of the large language models ChatGPT for automatic remediation," in *Proc. 16th Int. Conf. PErvasive Technol. Related Assistive Environ. (PETRA)*, 2023, pp. 707–713.

[46] H. Suh, M. Tafreshipour, S. Malek, and I. Ahmed, "Human or LLM? A comparative study on accessible code generation capability," *arXiv:2503.15885*, 2025.

[47] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," in *Proc. ACM SIGCHI Symp. Eng. Interactive Comput. Syst. (EICS)*, 2018, Art. no. 3.

[48] H. Laurençon, L. Tronchon, and V. Sanh, "Unlocking the conversion of web screenshots into HTML code with the WebSight dataset," *arXiv:2403.09029*, 2024.

[49] Y. Gui *et al.*, "WebCode2M: A real-world dataset for code generation from webpage designs," in *Proc. ACM Web Conf. (WWW)*, 2025.

[50] S. Yun *et al.*, "Web2Code: A large-scale webpage-to-code dataset and evaluation framework for multimodal LLMs," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS) Datasets Benchmarks Track*, 2024.

[51] K. Lee *et al.*, "Pix2Struct: Screenshot parsing as pretraining for visual language understanding," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 18893–18912.

[52] G. Baechler *et al.*, "ScreenAI: A vision-language model for UI and infographics understanding," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 2024.

[53] B. Deka *et al.*, "Rico: A mobile app dataset for building data-driven design applications," in *Proc. ACM Symp. User Interface Softw. Technol. (UIST)*, 2017, pp. 845–854.

[54] J.-M. López-Gil and J. Pereira, "Turning manual web accessibility success criteria into automatic: An LLM-based approach," *Univers. Access Inf. Soc.*, 2024.

[55] T. Calò, A.-E. Gurita, and L. De Russis, "Measuring the semantic accessibility gap in LLM-generated web UIs," Politecnico di Torino, Turin, Italy, 2026, pp. 1–5. [Online]. Available: https://iris.polito.it/handle/11583/3008330

[56] A. Y. Zheng, J. J. Lee, B. Benes, and R. A. Yeh, "WebAccessVL: Making an accessible web via violation-conditioned VLM," *arXiv:2602.03850*, 2026.

[57] C. Gleason *et al.*, "'It's almost like they're trying to hide it': How user-provided image descriptions have failed to make Twitter accessible," in *Proc. World Wide Web Conf. (WWW)*, 2019, pp. 549–559.

[58] E. Kreiss, C. Bennett, S. Hooshmand, E. Zelikman, M. R. Morris, and C. Potts, "Context matters for image descriptions for accessibility: Challenges for referenceless evaluation metrics," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2022, pp. 4685–4697.

[59] A. G. Mohanbabu and A. Pavel, "Context-aware image descriptions for web accessibility," in *Proc. 26th Int. ACM SIGACCESS Conf. Comput. Accessibility (ASSETS)*, St. John's, NL, Canada, 2024.

[60] Y. Li, G. Li, L. He, J. Zheng, H. Li, and Z. Guan, "Widget captioning: Generating natural language description for mobile user interface elements," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2020, pp. 5495–5510.

[61] J. Chen *et al.*, "Unblind your apps: Predicting natural-language labels for mobile GUI components by deep learning," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng. (ICSE)*, 2020, pp. 322–334.

[62] S. Haque and C. Csallner, "Infering alt-text for UI icons with large language models during app development," *arXiv:2409.18060*, 2024.

[63] X. Zhang et al., "Screen Recognition: Creating accessibility metadata for mobile applications from pixels," in Proc. CHI Conf. Human Factors Comput. Syst., 2021.

[64] N. Salehnamadi, A. Alshayban, J.-W. Lin, I. Ahmed, S. Branham, and S. Malek, "Latte: Use-case and assistive-service driven automated accessibility testing framework for Android," in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2021.

[65] M. Taeb, A. Swearngin, E. Schoop, R. Cheng, Y. Jiang, and J. Nichols, "AXNav: Replaying accessibility tests from natural language," in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2024.