Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Computer Engineering

A.a. 2025/2026

Graduation Session March 2026

Design and Development of a Framework for Evaluating AI-Generated User Interfaces

Relatori:

Luigi De Russis

Qusay H. Mahmoud

Candidati:

Gizem Irmak

Abstract

Generative AI–driven no-code platforms can automatically produce user interfaces from natural language prompts, yet the resulting designs frequently contain accessibility violations, usability inconsistencies, and ethically problematic interaction patterns. Existing evaluation tools focus primarily on isolated accessibility checks and assume complete structural access to the interface, limiting their use for modern multi-modal AI-generated outputs.

This thesis presents a framework for evaluating AI-generated user interfaces that unifies accessibility (Web Content Accessibility Guidelines), usability heuristics, and ethical design principles within a single evaluation taxonomy. The framework is modality-aware, adapting its evaluation strategy to different interface representations, including source code, repository artifacts, and rendered interface images with varying levels of structural observability. The architecture combines static interface analysis with constrained large-language-model reasoning, while an epistemic severity model categorizes findings as confirmed, potential, or non-evaluable.

A prototype implementation generates structured corrective prompts and introduces a convergence-based mechanism to measure reductions in confirmed violations across iterations. The evaluation results demonstrate systematic identification of recurring deficiencies and measurable improvement of generated interfaces, indicating that reliable evaluation of AI-generated interfaces requires hybrid reasoning and explicit uncertainty modeling rather than purely automated checking.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Prof. Luigi De Russis and Prof. Qusay H. Mahmoud, for their guidance, valuable feedback, and continuous support throughout the development of this thesis.

I am grateful to Prof. Qusay H. Mahmoud for his mentorship during my visiting research period at Ontario Tech University. His encouragement and understanding created a welcoming research environment and made this experience both enriching and memorable.

I would also like to thank Ontario Tech University for the opportunity to conduct part of this research as a visiting researcher, and Politecnico di Torino for the education and support that made this work possible.

My sincere thanks go to the friends I met in Canada during my mobility period, who made this experience truly special. I am also grateful to my friends in Italy and Turkey for their encouragement and support throughout this time.

Finally, I would like to thank my family for their constant support, patience, and encouragement during my studies. Their motivation and understanding made this journey possible.

Table of Contents

List of Figures

Chapter 1

Introduction

Low-code/no-code (LCNC) platforms have transformed software development by offering visual environments and pre-built components that require minimal or no programming knowledge. This accessibility has contributed to the "democratization" of software creation, enabling non-technical professionals, often referred to as citizen developers, to construct applications quickly and cost-effectively [1]. LCNC solutions are widely adopted because they accelerate development, reduce labor costs, and improve agility, thereby addressing traditional challenges such as delayed delivery and high implementation effort [2]. Adoption is expected to continue growing, with projections indicating use in nearly 70% of enterprise applications by 2025 and approximately 65% of software projects overall [3, 4].

The integration of LCNC platforms with Generative Artificial Intelligence (GenAI) represents a further shift in software creation workflows. GenAI tools use deep learning models to generate new content and automate development tasks, enhancing LCNC environments by supporting ideation, prototyping, and feature refinement [1]. In user interface (UI) development, Multimodal Large Language Models (MLLMs) can translate sketches, screenshots, or natural-language descriptions directly into functional code (UI2Code) [5]. Commercial systems such as Bolt and Replit, along with similar platforms, illustrate this emerging paradigm in which interfaces can be produced from textual instructions [6, 7].

However, reliance on generative no-code tools introduces challenges related to design quality, consistency, and predictability. Natural-language prompts are inherently ambiguous, making it difficult for users to precisely specify design intent. Furthermore, because large language models operate probabilistically, outputs may vary across executions. Studies evaluating systems including Bolt, Replit, and comparable platforms report that identical prompts can produce substantially different layouts, styling decisions, and component structures across tools and across repeated runs within the same tool [8, 9]. As a result, users frequently engage in iterative prompting to refine outputs, reducing expected efficiency gains.

In response to these challenges, this thesis investigates how AI-generated user interfaces can be systematically evaluated through a structured, multi-dimensional framework integrating accessibility, usability, ethical design considerations, and iteration-based convergence analysis. By combining a systematic literature review with the design and implementation of a deterministic-first, hybrid rule-based evaluation architecture, this work seeks to bridge the gap between automated UI generation and established human-centered design principles.

1.1 Goal

The primary goal of this thesis is to design, develop, and validate a structured framework for evaluating AI-generated user interfaces produced by generative no-code tools. As large language models increasingly automate UI generation from natural-language prompts, there is a growing need for systematic evaluation methods capable of assessing accessibility, usability, ethical design considerations, and reproducibility across iterations and tools.

To address this need, this thesis first conducts a systematic literature review to identify evaluation gaps and recurring limitations in generative UI research. Based on these findings, it proposes a multi-dimensional rule taxonomy and implements a deterministic-first hybrid evaluation architecture capable of distinguishing structurally verifiable violations from interpretive design risks. A proof-of-concept system is developed to operationalize this framework and is applied in a comparative evaluation of selected generative no-code platforms (Bolt, Lovable, and Replit). The broader objective is to contribute toward more transparent, reproducible, and human-centered evaluation practices in AI-driven UI generation.

1.2 Research Questions

This study was guided by one main research question (MRQ) and four supporting sub-questions designed to examine how generative AI–based no-code tools make UI design decisions, how consistent these decisions are, and how well they align with established design principles.

- MRQ: How can AI-generated user interfaces produced by generative no-code tools be systematically evaluated with respect to accessibility, usability, and ethical design principles?

- RQ1: What underlying design principles, layout structures, or color strategies can be observed in the generated interfaces?

- RQ2: How do generative design tools (e.g., Bolt, Lovable, Replit) vary in the user interfaces they generate when given the same prompt, both across different tools and across repeated generations within the same tool?

- RQ3: To what extent do the generated interfaces adhere to established usability and accessibility guidelines (e.g., WCAG and Nielsen's heuristics)?

- RQ4: What limitations, biases, or inconsistencies exist in the current generative UI design systems?

1.3 Contributions

This thesis contributes to the emerging field of AI-assisted user interface generation by proposing a structured framework for evaluating interfaces produced by generative no-code tools. The main contributions of this work are:

- A modality-aware evaluation framework capable of analyzing AI-generated interfaces from source code archives, GitHub repositories, and screenshots.

- A unified rule taxonomy integrating accessibility standards (WCAG), usability heuristics, and ethical design principles.

- A hybrid evaluation architecture combining deterministic rule-based analysis with LLM-assisted reasoning for higher-level design interpretation.

- An iteration-based convergence mechanism for tracking improvements across successive prompt-driven interface refinements.

- A proof-of-concept implementation (UI Critic) demonstrating the practical application of the proposed framework.

1.4 Methodology

This study employed a Systematic Literature Review (SLR) methodology to identify, evaluate, and synthesize research on automated user interface design in generative no-code tools. The review followed a structured protocol [10] to ensure transparency and reproducibility and focused on design decisions, aesthetic behavior, and consistency across generated interfaces.

1.4.1 Review Protocol and Planning

The review followed an a priori protocol based on Kitchenham's guidelines and PRISMA 2020 reporting standards [10, 11]. The protocol defined the research questions, search strategy, eligibility criteria, quality assessment, and data extraction procedures before the search was conducted. This process aimed to reduce selection bias and ensure replicability of the study.

Literature searches were conducted in IEEE Xplore, ACM Digital Library, and Google Scholar (see Table 1.1), selected for coverage of human–computer interaction, software engineering, and artificial intelligence research. Search expressions combined terms related to low-code/no-code platforms, generative AI or large language models, and user interface design. Essentially: ("no-code" OR "low-code") AND ("generative AI" OR "large language model" OR LLM) AND ("user interface" OR UI OR UX).

Searches were restricted to peer-reviewed publications between 2021 and 2025. Academic theses were included when they provided empirical evidence directly related to generative UI design.

Table 1.1: Summary of database searches and initial screening results.

Database	**Search Strings Used**	**Total Retrieved**	**Records After Initial Screening**
IEEE Xplore	No-code / low-code AND generative AI / LLM AND UI design	54	44
ACM Digital Library	No-code / low-code AND user experience / UI design	60	27
Google Scholar	No-code UI design AND evaluation / prompt-based tools	30	18

1.4.2 Inclusion and Exclusion Criteria

Studies were included if they examined AI-based low-code or no-code tools for user interface generation and evaluated design, usability, accessibility, or related quality aspects. Eligible sources consisted of peer-reviewed publications or reputable academic theses presenting empirical or structured analytical evidence. Studies were excluded if they lacked a UI/UX focus, did not involve generative or automated design approaches, or consisted of non-academic sources such as tutorials, blogs, or white papers. The criteria were applied during both title–abstract screening and full-text review.

1.4.3 Study Selection Process

The selection process followed PRISMA guidelines [11]. After removing duplicates, titles and abstracts were screened for relevance. Remaining papers underwent full-text assessment using the eligibility criteria. Twenty studies satisfied all criteria and were included in the final synthesis. The overall study selection process is summarized using a PRISMA flow diagram (Figure 1.1) that illustrates the screening reduction from initial retrieval to final inclusion.

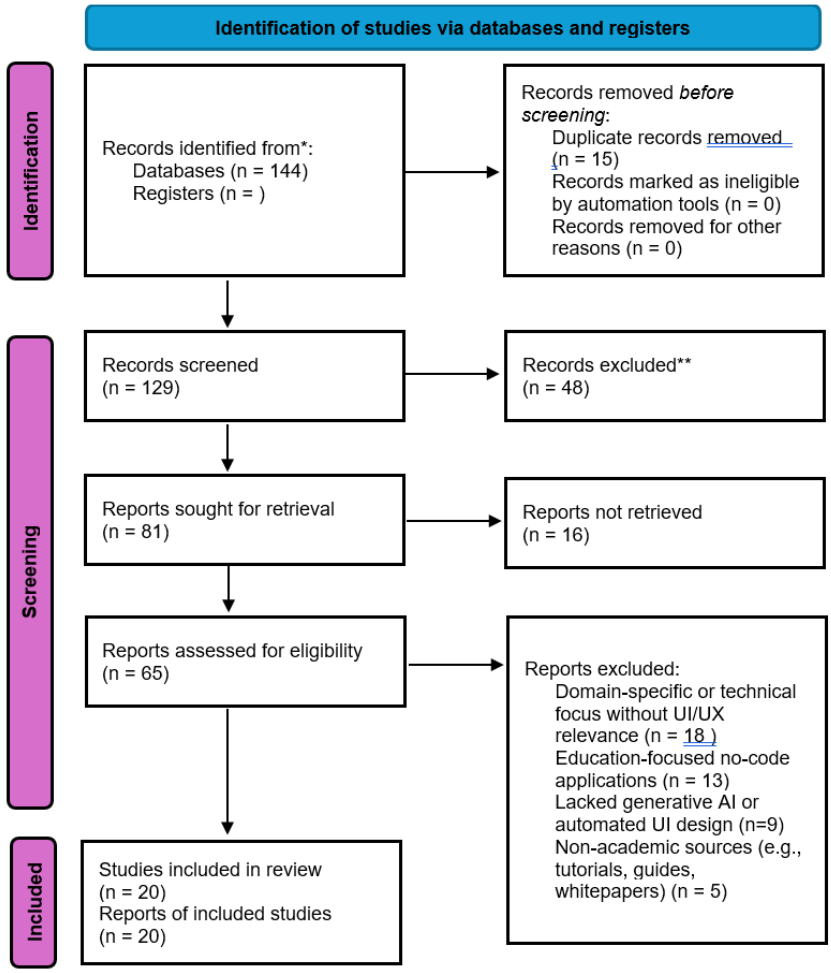

Figure 1.1: PRISMA-Based Study Selection Flow Diagram.

The review process adhered to the PRISMA 2020 reporting guidelines for systematic reviews. Figure 1.1 presents the PRISMA flow diagram detailing the

identification, screening, eligibility assessment, and inclusion of studies.

1.4.4 Data Extraction and Categorization

A structured extraction form recorded publication details, tool type, generative approach, evaluation method, assessed UI quality dimensions, and reported findings. Extracted data were categorized to enable comparison across studies and support thematic synthesis of design behavior and variability.

1.4.5 Quality Assessment Criteria

Study quality was evaluated using criteria adapted from Kitchenham [10], assessing relevance, methodological clarity, and transparency of evaluation procedures. Studies were classified as high, medium, or low relevance (Table 1.2). The classification informed interpretation but did not exclude studies from the synthesis.

Table 1.2: Quality assessment criteria for included studies.

Criterion	Description
QA1	Relevance to generative no-code or low-code UI design
QA2	Clarity of research objectives and contribution
QA3	Appropriateness of the methodological approach
QA4	Transparency of UI evaluation or analysis procedures
QA5	Clarity and validity of reported findings

1.4.6 Synthesis and Analysis Approach

The synthesis combined qualitative comparison with descriptive aggregation. Studies were grouped by UI quality dimensions, including layout structure, aesthetics, usability, accessibility, consistency, and developer effort. Frequency analysis (Table 1.3) identified commonly evaluated aspects, while cross-study comparison revealed recurring patterns and limitations in generative UI behavior.

Individual studies may contribute to multiple research questions, reflecting the multifaceted nature of generative UI design and evaluation. Table 1.4 maps the included studies to the research questions addressed in this review.

Table 1.3: Frequency of UI quality dimensions assessed across included studies.

UI Quality Dimension	**Number of Studies** ($n = 20$)
Layout & Structure	10
Aesthetics & Visual Design	5
Usability & Learnability	5
Accessibility	4
Consistency & Variability	7

Table 1.4: Mapping of included studies to research questions

Research Question	**Number of Studies**	**Studies (Author [ref])**
RQ1	11	Gui Y. [5], Vaithilingam P. [6], Leung A. [8], Zhou T. [12], Wu F. [13], Wan Y. [**wan2025divide**], Lu Y. [14], Kamnerddee C. [15], Feng S. [16], Lee J. [17]
RQ2	9	Gui Y. [5], Duan P. [18], Huang T. [19], Zhou T. [12], Wu F. [13], Kamnerddee C. [15], Monteiro M. [20], Lee J. [17], Owen E. [21]
RQ3	6	Duan P. [18], Gurita A. [22], Shahab M. A. [23], Doush I. A. [24], Fischer M. [25]
RQ4	10	Gui Y. [5], Duan P. [18], Wu F. [13], Wan Y. [**wan2025divide**], Kamnerddee C. [15], Shahab M. A. [23], Fischer M. [25], Feng S. [16], Lee J. [17], Owen E. [21]

1.5 Thesis Structure

The remainder of this thesis is structured as follows.

Chapter 2 presents the background and related work. It introduces the foundations of low-code/no-code development and generative UI systems and reviews existing research on accessibility, usability, and evaluation approaches for AI-generated interfaces.

Chapter 3 introduces the proposed evaluation framework. It describes the overall

framework architecture, the rule taxonomy covering accessibility, usability, and ethical design principles, and the severity and convergence model used to classify and interpret evaluation findings.

Chapter 4 describes the proof-of-concept implementation of the proposed framework. It presents the system architecture, the evaluation pipeline, and the reporting interface used to analyze and present rule-based findings.

Chapter 5 presents the experimental results and evaluation. It reports the comparative analysis of interfaces generated by different generative no-code tools, including cross-tool comparisons, modality consistency experiments, and real-world repository evaluations.

Chapter 6 concludes the thesis by summarizing the main contributions, discussing the implications of the findings, and outlining limitations and directions for future work.

Chapter 2

Background and Related Work

The study of generative no-code tools for automated user interface (UI) design sits at the intersection of two accelerating trends in software engineering: the mainstream adoption of Low-Code/No-Code (LCNC) development and the rapid integration of Generative Artificial Intelligence (GenAI) into end-user programming workflows. LCNC platforms have historically aimed to reduce the technical barrier to application development through visual abstractions and reusable components, enabling "citizen developers" to produce functional software with limited or no programming expertise. More recently, Large Language Models (LLMs) and Multimodal LLMs (MLLMs) have extended this paradigm by enabling prompt-driven generation of complete front-end interfaces and code from natural-language descriptions and visual inputs. This convergence increases development speed and accessibility, but it also introduces new concerns around reliability, reproducibility, accessibility compliance, usability quality, and aesthetic consistency of generated UIs. Accordingly, this chapter first establishes the conceptual and technological foundations of LCNC and generative UI synthesis, and then reviews prior research on platform characteristics, UI generation systems, and empirical evaluations of AI-generated interface quality, culminating in the research gaps that motivate the systematic review and evaluation framework developed in this thesis.

2.1 Background

The evaluation of no-code tools for automated user interface (UI) design operates within the broader context of the evolving software industry, driven by the emergence of Low-Code/No-Code (LCNC) platforms and the subsequent integration of Generative Artificial Intelligence (GenAI). This background outlines the historical

development of LCNC systems, the factors driving their adoption, the conceptual distinctions between low- and no-code approaches, and the transformative impact of GenAI, particularly Large Language Models (LLMs) and Multimodal LLMs (MLLMs), on automated UI generation. It also highlights the technical and aesthetic challenges that underpin the need for a systematic evaluation.

2.1.1 The Rise of Low-Code/No-Code Development

Low-code/no-code (LCNC) platforms emerged from earlier visual and rapid application development approaches that aimed to reduce programming effort through abstraction and visual interaction [26, 27]. These platforms allow users to build applications using graphical components and declarative modeling rather than traditional coding, enabling participation from non-programmers often referred to as citizen developers [1].

Adoption has been driven by increasing software demand, developer shortages, and the need for faster delivery cycles [28, 29]. Industry reports suggest LCNC solutions can significantly accelerate development and reduce costs compared to conventional approaches [3, 4]. While they improve accessibility and agility, they also introduce limitations related to scalability, customization, and control [30].

2.1.2 Defining Low-Code and No-Code

Although often grouped together, low-code and no-code platforms target different levels of technical expertise. As shown in Table 2.1, low-code platforms support developers or technically skilled citizen developers by allowing limited scripting and customization, whereas no-code platforms rely primarily on visual workflows and pre-built components for users with minimal programming background [26, 27, 30, 31].

Both approaches abstract implementation details through graphical development environments, enabling faster application creation while reducing reliance on traditional programming. This study focuses on generative no-code systems, which extend this abstraction by using large language models to produce user interfaces directly from natural-language prompts.

2.1.3 The Convergence with Generative AI in UI Design

The integration of Generative Artificial Intelligence into low-code/no-code platforms enables interfaces to be produced directly from natural-language descriptions rather than manually assembled components [1]. Using large language models (LLMs) and multimodal LLMs (MLLMs), these systems can translate prompts, sketches,

Table 2.1: Low-Code versus No-Code platform characteristics.

Feature	**Low-Code (LC) Platforms**	**No-Code (NC) Platforms**
Primary goal	Accelerate development with minimal manual coding [27].	Eliminate the need for traditional coding altogether [27].
Target user	IT professionals, developers, or "citizen developers" with some coding knowledge [30].	Business users or "citizen developers" who have minimal or zero programming background [26].
Customization	Allows a high degree of customisation and flexibility, often enabling users to write some custom scripts/code [31].	Relies heavily on predesigned components, templates, and ready-made functions [31].
Technical Approach	Acts as a middle ground, speeding up development with minimal coding [27].	Utilizes visual programming entirely through a graphical interface and drag-and-drop functions [27].

or screenshots into functional UI code, commonly referred to as UI2Code or design-to-code generation [19].

This shift changes interface creation from deterministic configuration to probabilistic generation. While it accelerates prototyping and lowers technical barriers, it also introduces uncertainty in how design decisions are inferred, motivating the need to examine consistency, reliability, and alignment with human-centered design principles.

Multimodal Large Language Models (MLLMs)

Multimodal large language models extend text-based generation by incorporating visual understanding, allowing UI code to be generated from mockups, screenshots, or other visual artifacts [12, 13]. Compared with earlier vision-based approaches, these models can produce more complete interface structures but still struggle with complex layouts and interaction logic [32]. As a result, generated interfaces may contain missing elements, incorrect component mappings, or inconsistent behaviors across pages.

Structured Approaches for Managing UI Complexity

Because direct prompt-based generation struggles with complex layouts, several approaches introduce intermediate structural representations to guide UI synthesis [5, 13, 32]. These methods decompose interfaces into hierarchical regions or layout trees before code generation, improving structural accuracy and reducing omissions. However, while structured pipelines improve reliability in controlled scenarios, prompt-driven no-code tools still expose variability because internal representations remain hidden from end users.

2.1.4 Challenges in Generative AI and No-Code UI Automation

The integration of generative AI into no-code environments introduces challenges related to predictability, correctness, and design quality. Because large language models operate probabilistically, identical prompts may produce different outputs, complicating testing and reproducibility [9]. The ambiguity of natural language further affects reliability by making precise specification difficult and increasing prompt interpretation variability [19, 33].

At the implementation level, generated interfaces often contain structural and functional errors, including missing elements, incorrect component mappings, navigation inconsistencies, and compilation issues requiring manual correction [12, 32]. These problems are particularly evident in multi-page applications, where consistent behavior across screens is difficult to maintain.

Beyond functional correctness, design quality also varies. Generated interfaces may exhibit homogenized layouts, lack pixel-level precision, or require manual visual adjustments to meet professional standards [18, 14, 34]. Finally, limitations inherent to LCNC platforms, such as scalability constraints, hidden platform bugs, and accumulated technical debt, can propagate into generated applications and further reduce reliability [3, 35].

2.2 Related Work

Generative no-code UI research spans three primary directions: platform-oriented studies on low-code/no-code ecosystems, empirical evaluations of AI-generated interfaces, and accessibility or usability assessments. While these works (see Table 2.2) provide valuable insights into productivity and compliance, they rarely examine design consistency, aesthetic reasoning, or reproducibility across generated interfaces. Consequently, existing literature offers fragmented perspectives on UI quality rather than an integrated understanding of automated design behavior.

Table 2.2: Key related studies on generative no-code and AI-assisted UI design.

Study	**Year**	**Main Contribution**	**Limitations in UI Evaluation**
Zhou T. [12]	2025	This paper presents a structured design-to-code pipeline that uses computer vision, multimodal LLMs, and intermediate representations to generate declarative UI code from mockups and screenshots.	The review focuses on functional correctness and code reliability, without evaluating user-facing UI quality, aesthetics, usability, accessibility, or generative variability.
Wan Y. [32]	2025	This paper introduces a divide-and-conquer pipeline that decomposes UI screenshots into semantic regions and uses multimodal LLMs to generate and assemble HTML/CSS with improved visual and code similarity.	The review emphasizes design-to-code fidelity and does not assess human-centered UI qualities such as aesthetics, usability, accessibility, interaction clarity, or generative variability.
Leung A. [8]	2025	This paper presents an interactive UI-authoring system that combines LLM-based generation with a structured intermediate representation to support guided exploration and controlled refinements in UI prototyping.	The review focuses on prototyping workflows and does not evaluate UI design quality, including aesthetics, layout consistency, usability, accessibility, or broader user experience.

Study	Year	Main Contribution	Limitations in UI Evaluation
Liu D. [3]	2024	This paper empirically analyzes bugs from major low-code platforms, showing that many defects originate during design and frequently involve UI graphics or interaction logic.	The review focuses on platform-level bugs and does not evaluate UI design quality, layout structure, aesthetics, usability, accessibility, or design decision-making.
Duan P. [18]	2024	This paper introduces UICrit, a large-scale dataset of expert UI critiques and ratings for mobile interfaces, demonstrating that visually guided few-shot prompting improves LLM-generated design feedback.	The review focuses on issue-level critiques and does not provide a systematic evaluation of layout structure, aesthetics, consistency, usability, reproducibility, or developer effort.
Gui Y. [5]	2025	This paper presents UICopilot, a hierarchical UI-to-code generation system that decouples structural prediction from HTML/CSS synthesis to improve layout accuracy and visual similarity.	The review emphasizes structural and visual similarity metrics and does not assess human-centered UI qualities such as aesthetics, usability, accessibility, design consistency, or developer experience.
Khalajzadeh H. [36]	2024	This paper presents a systematic literature review of low-code development accessibility, synthesizing strategies, evaluation methods, and research gaps for inclusive low-code IDE design.	The review focuses on accessibility of low-code development environments and does not evaluate AI-generated UI artifacts including layout quality, aesthetics, design consistency, reproducibility, or end-user interaction design.

Study	Year	Main Contribution	Limitations in UI Evaluation
Somer P. [37]	2025	This paper presents a structured literature review of studies on algorithmic accountability in low-code/no-code AI platforms, identifying governance risks and mitigation mechanisms.	The review focuses on governance and accountability rather than user-facing interface design and does not examine UI layout, aesthetics, usability, accessibility, reproducibility, or design variability.
Kamouch H. E. [38]	2023	This paper presents a PRISMA-guided systematic literature review of low-code/no-code platforms, synthesizing technologies, domains, benefits, and challenges supporting citizen development.	The review provides a high-level overview of LCNC platforms but does not empirically evaluate UI design quality, generative behavior, reproducibility, or tool-specific design characteristics.
Gao D. [39]	2024	This paper introduces EDEQ-LCDP, a questionnaire instrument for measuring episodic developer experience during short interactions with low-code development platforms.	The review focuses on developer experience within a single platform and does not evaluate UI layout, aesthetics, usability, accessibility, or generative design behavior.
Roy et al. [40]	2025	This paper benchmarks multiple generative AI tools across software engineering tasks, highlighting workflow trade-offs and implications for educational use.	The review limits UI evaluation to high-level scaffolding and diagram accuracy, and does not analyze layout quality, aesthetics, usability, accessibility, reproducibility, or tool-specific design behavior across repeated generations.

Study	Year	Main Contribution	Limitations in UI Evaluation
Lively et al. [41]	2023	This paper examines how text- and image-based generative AI tools influence creativity, aesthetic decision-making, and productivity in web design and UX education.	The review focuses on educational outcomes and ideation support and does not systematically evaluate UI layout, design consistency, usability, accessibility, reproducibility, or autonomous generative behavior in no-code UI systems.
Xu H. [42]	2025	This paper proposes a knowledge-augmented framework that converts UI wireframes into React-based front-end applications using visual prompting and retrieval-augmented generation to improve code structure and scalability.	The review emphasizes code correctness and architectural quality rather than evaluating UI layout, aesthetics, usability, accessibility, reproducibility, or comparative generative behavior across tools and prompts.
Dibia V. [43]	2024	This paper introduces AUTOGEN STUDIO, a no-code environment that uses structured, declarative representations to improve controllability, transparency, and reproducibility in LLM-based multi-agent workflows.	The review does not address user-facing UI generation and does not evaluate visual layout, aesthetics, usability, accessibility, or design consistency, limiting its relevance to human-centered UI quality analysis in generative no-code tools.

Study	Year	Main Contribution	Limitations in UI Evaluation
Liu Y. F. [44]	2024	This paper evaluates an AI-enhanced mobile learning application using expert-based usability heuristics, identifying strengths in visual design and feedback alongside issues in user control and consistency.	The review is limited to expert-based heuristic analysis of a single handcrafted educational application and does not assess generative or prompt-driven UI systems, formal accessibility compliance, reproducibility, or tool-specific design behavior.
Gurita A. E. [22]	2025	This paper evaluates AI-generated user interfaces for accessibility compliance by analyzing 90 interfaces and examining how prompting strategies influence accessibility and design outcomes.	The review focuses primarily on accessibility and does not evaluate broader UI qualities such as layout structure, aesthetics, usability, or learnability, while limiting analysis to static outputs from two AI tools without considering developer effort, code quality, or real-world deployment.
Gurita A. E. [45]	2025	This paper evaluates 200 AI-generated user interfaces from five design tools against WCAG criteria, analyzing how interactive prompt engineering affects accessibility and visual consistency.	The review prioritizes accessibility compliance over holistic UI quality and does not systematically assess layout structure, aesthetics, usability, code quality, or reproducibility across identical prompts.

Study	Year	Main Contribution	Limitations in UI Evaluation
Iman J. A. [46]	2025	This paper compares AI-generated minimalist UI refinements with human-designed interfaces, showing that AI-refined designs can outperform human refinements in usability, efficiency, and cognitive load under constrained minimalist settings.	The review is limited to minimalist UI scenarios and does not examine layout variability, accessibility compliance, reproducibility across prompts or tools, developer effort, or long-term design maintainability.
Kamnerddee C. [15]	2024	This paper compares human-designed and AI-generated mobile UI prototypes using usability testing and interviews, and proposes the AID-UX framework to support effective human–AI collaboration in UI design.	The review focuses on prototype-level evaluations rather than deployed no-code tools and does not assess layout reproducibility, aesthetic consistency, accessibility compliance, or cross-tool variability.
Shahab M. A. [23]	2024	This paper investigates the perceived impact of generative AI tools on mobile UI design workflows using the SPACE framework, based on a survey of professional designers and developers.	The review emphasizes perceived efficiency based on self-reported data and does not objectively evaluate generated UI layouts, aesthetics, accessibility, reproducibility, or artifact-level design quality across generative no-code tools.

Study	Year	Main Contribution	Limitations in UI Evaluation
Doush I. A. [24]	2024	This paper evaluates AI-generated web code from ChatGPT and Microsoft Copilot using metric-driven measures of accessibility compliance, correctness, cross-browser compatibility, and standards conformance.	The review focuses on code-level accessibility metrics and does not evaluate UI layout, visual aesthetics, usability, design consistency, or variability across multiple generated interfaces.
Monteiro M. [20]	2025	This paper introduces NoCodeGPT, a task-oriented no-code interface built on GPT models that guides non-experts through structured application-building workflows and improves task completion compared to chat-based prompting.	The review is limited to small-scale applications and student participants and does not assess UI design quality, layout consistency, accessibility, reproducibility, or developer effort beyond task completion.
Fischer M. [25]	2024	This paper evaluates how generative AI tools support multiple software-development phases using a user-centered lens, combining a literature review with an 18-student evaluation focused on usability and workflow integration.	The review does not assess UI layout quality, visual aesthetics, or design consistency, and relies on a small student sample while focusing on tool support rather than comparative analysis of generated UI artifacts.

Study	Year	Main Contribution	Limitations in UI Evaluation
Paliwal G. [1]	2024	This paper reviews the convergence of low-code/no-code platforms and generative AI, highlighting how GenAI supports rapid development, automation, and prototyping through a small illustrative case study using Noodl with GPT-4o.	The review remains conceptual and descriptive, lacking systematic evaluation of UI design quality, layout, usability, accessibility, or reproducibility, and relies on a limited case study without scalability or comparative analysis.
Guthardt T. [47]	2024	This paper empirically compares programmers and citizen developers using a prototype no-code builder, finding comparable task correctness, completion time, and perceived usability between the two groups.	The review relies on a simplified custom tool and small tasks and does not assess UI layout, visual quality, aesthetics, accessibility, or design consistency.
Liu S. [48]	2022	This paper describes the design and usability evaluation of Pathverse, a no-code platform enabling researchers to build mHealth applications through drag-and-drop configuration and behavioral logic authoring.	The review focuses on platform development rather than UI design quality and is based on usability testing with a very small sample, limiting generalizability.
Pinho D. [49]	2023	This paper systematically reviews 38 studies on low-code platforms to analyze how usability is addressed, identifying common platform features and recurring usability challenges.	The review focuses on platform-level usability rather than evaluating generated UI artifacts or AI-driven UI generation.

Study	**Year**	**Main Contribution**	**Limitations in UI Evaluation**
Velásquez A. P. [4]	2024	This paper systematically reviews 15 studies on low-code development, identifying common platform features, key benefits, and recurring challenges such as limited customization and scalability.	The review focuses on general LCDP features rather than UI layout, aesthetics, usability, accessibility, or design consistency.
Rokis K. [50]	2023	This paper provides a comprehensive literature review of low-code development, synthesizing definitions, lifecycles, platform features, benefits, challenges, and adoption patterns.	The review focuses on general low-code development concepts rather than UI layout, visual design, usability, accessibility, or design-related challenges.
Ajimati M. O. [51]	2025	This paper presents a multi-phase systematic literature review of 40 empirical studies on low-code/no-code adoption.	The review focuses on organizational adoption rather than UI quality or technical output and does not evaluate layout, usability, aesthetics, accessibility, or generated artifacts.
Sufi F. [52]	2023	This paper presents a practical review of 47 LCNC studies, identifying common usage patterns and highlighting how LCNC tools enable non-technical users to perform AI-driven tasks.	The review focuses on algorithmic workflows rather than generative UI design or reproducibility.

Study	Year	Main Contribution	Limitations in UI Evaluation
Zhang L. [27]	2024	This paper evaluates low-code/no-code platforms using the ISO 25010 quality model.	The review focuses on general software quality rather than UI layout, aesthetics, consistency, accessibility, or generative UI behavior.
Cui J. [35]	2024	This paper qualitatively reviews academic and industry sources to examine how LCNC platforms affect development efficiency and delivery quality.	The review focuses on productivity outcomes rather than UI layout, usability, accessibility, or design quality.
Abahussain O. [30]	2025	This paper reviews the benefits and risks of LCNC platforms, highlighting rapid development advantages and challenges such as scalability and integration.	The review does not examine UI/UX aspects such as layout, usability, accessibility, or visual design.
Upadhyaya N. [53]	2023	This paper reviews how LCNC platforms influence traditional software development, summarizing benefits and challenges related to governance and integration.	The review focuses on organizational impacts rather than UI design and does not analyze usability, accessibility, visual layout quality, or developer experience.

2.2.1 Systematic Literature Reviews and Surveys on Low-Code/No-Code Platforms

Prior systematic reviews primarily examine LCNC platforms from technological and organizational perspectives, focusing on adoption, productivity, and governance [50, 51, 4, 30, 53]. These studies consistently report faster development cycles and improved accessibility for non-programmers but also identify scalability limitations, customization constraints, and vendor lock-in risks.

However, UI is typically treated as part of platform usability rather than as

a design artifact. Evaluations therefore rely on perceived ease of use, workload reduction, or workflow efficiency rather than analysis of generated layouts, visual hierarchy, or aesthetic consistency [49, 52]. As a result, the visual and behavioral properties of generated interfaces remain largely unexplored.

2.2.2 Empirical Studies Evaluating AI-Generated User Interface Designs

Empirical studies comparing AI-generated and human-designed interfaces report comparable baseline usability and efficiency but highlight differences in perceived intuitiveness and engagement [15, 46]. Other investigations emphasize productivity improvements and faster ideation workflows rather than direct assessment of design quality [23, 20].

Although these studies evaluate usability metrics such as task completion time, SUS scores, and cognitive workload, they rarely analyze aesthetic logic, layout consistency, or variability across repeated generations. Most experiments also rely on simplified tasks or single outputs, limiting insight into real-world generative behavior.

2.2.3 Systems for UI Authoring and Structured Representations

Several systems introduce intermediate structural representations to improve controllability and reliability in AI-assisted UI generation [**wan2025divide**, 8, 12, 42, 43]. By constraining generation through layout decomposition or guided refinement, these approaches reduce structural errors and improve code correctness. However, they focus primarily on functional fidelity and implementation accuracy rather than human-centered design qualities such as aesthetic reasoning, usability, or cross-generation consistency. Consequently, they provide technical stability but limited understanding of emergent design behavior in prompt-driven no-code tools.

2.2.4 Accessibility-and Usability-Focused Evaluations of AI-Generated Interfaces

Accessibility-focused research evaluates AI-generated interfaces using WCAG and ARIA compliance metrics, generally finding acceptable baseline accessibility but frequent minor violations requiring iterative correction [22, 45, 24]. These systems often achieve compliance through standardized patterns, which can reduce creative diversity and produce visually similar interfaces.

Usability studies similarly report adequate task performance but persistent preference for human-designed interfaces, suggesting that technical correctness

does not fully capture perceived design quality [15]. Across both research streams, accessibility and usability are typically assessed independently, leaving broader design coherence and reproducibility insufficiently examined.

2.2.5 Existing UI Evaluation Tools

Several automated and semi-automated tools have been developed to support the evaluation of user interface accessibility and usability. Accessibility auditing tools such as Axe, Lighthouse, Pa11y, and WAVE automatically detect violations of Web Content Accessibility Guidelines (WCAG) through static or runtime analysis of HTML structures and ARIA attributes.

While these tools are effective for identifying technical accessibility issues, they primarily focus on code-level compliance and provide limited support for broader usability and ethical design considerations. Usability evaluation is often performed through heuristic evaluation methods based on established principles such as Nielsen's usability heuristics, which require manual inspection by experts.

Consequently, existing evaluation approaches remain fragmented. Most automated tools operate only on source code or rendered web pages, do not support screenshot-based analysis, and do not track iterative improvements during UI development. Ethical design considerations, such as transparency, choice architecture, and user control, are also rarely included in automated UI evaluation pipelines.

To illustrate these limitations, Table 2.3 compares common UI evaluation approaches with the proposed UI Critic framework.

Table 2.3: Comparison of existing UI evaluation tools and the proposed UI Critic framework.

Tool	**Accessibility**	**Usability**	**Ethical Design**	**Screenshot Input**	**Iteration Tracking**
Axe DevTools	✓	–	–	–	–
Google Lighthouse	✓	Limited	–	–	–
Pa11y	✓	–	–	–	–
WAVE	✓	–	–	–	–
Heuristic Evaluation	Limited	✓	Limited	✓	–
UI Critic (Proposed)	✓	✓	✓	✓	✓

2.2.6 Motivation for Further Systematic Review

Existing literature demonstrates that generative UI systems can support development efficiency and baseline usability but provides limited insight into design consistency, aesthetic variability, and reproducibility. Moreover, prior evaluations treat accessibility and usability as isolated properties and rarely analyze how automated tools interpret design intent across tools or repeated generations. These gaps motivate a systematic review focused specifically on layout structure, visual aesthetics, consistency, usability, accessibility, and prompt-to-output reliability in generative no-code interface design.

2.2.7 Summary

This chapter situated generative no-code UI tools within the broader evolution of LCNC development and the shift toward prompt-driven interface synthesis enabled by LLMs and MLLMs. While LCNC platforms are widely studied in terms of adoption and productivity, generated UI artifacts are rarely examined as primary objects of systematic, comparative analysis.

Existing studies report baseline usability and, in controlled contexts, acceptable perceptual performance. However, findings remain fragmented across accessibility, usability, aesthetics, and reproducibility. Accessibility research further indicates that WCAG compliance is often achieved through conservative patterns, sometimes requiring iterative refinement.

Two key gaps emerge: the absence of an integrated quality model spanning accessibility, usability, and ethical design, and the lack of reproducible mechanisms for cross-tool and cross-iteration evaluation. These gaps motivate the systematic review and inform the design of the UI Critic framework introduced in Chapter 3.

Chapter 3

Proposed Framework

This chapter introduces the proposed evaluation framework and constitutes the central methodological contribution of this thesis. Building upon the systematic literature review and the identified limitations of current generative no-code systems, the chapter formalizes a structured, modality-aware framework for assessing AI-generated user interfaces. The framework translates accessibility standards, usability heuristics, and ethical design principles into executable evaluation logic, enabling systematic and reproducible validation of front-end artifacts. The chapter first outlines the design objectives and architectural structure of the framework, followed by the definition of the rule taxonomy across accessibility, usability, and ethical dimensions. It then presents the severity classification and convergence model that govern how findings are interpreted and integrated into iterative evaluation cycles. Together, these components establish a hybrid deterministic–interpretive evaluation methodology designed to support transparent benchmarking, structured refinement, and longitudinal analysis of generative interface systems.

3.1 Framework Overview

This section introduces the conceptual and architectural foundations of the proposed evaluation framework for AI-generated user interfaces. It outlines the system's objectives, structural design, and methodological positioning, clarifying how rule-based logic, modality-aware analysis, and controlled AI-assisted reasoning are integrated into a unified evaluation pipeline. Rather than functioning as a conventional checklist-based auditor, the framework operates as a reproducible, iteration-aware instrument for systematically assessing compliance, usability quality, and ethical interaction properties in generative UI outputs.

3.1.1 Design Objectives and Scope

The proposed framework is designed to provide a structured and reproducible evaluation mechanism for AI-generated user interfaces. Its objectives are to operationalize design principles into executable logic, preserve evaluation reproducibility, accommodate multiple input modalities, generate actionable corrective feedback, and support iterative convergence analysis.

First, the framework operationalizes established design principles into rule-based evaluation logic. Accessibility rules (A1–A6) formalize WCAG-aligned structural requirements into deterministic checks, such as luminance contrast computation and attribute validation. Usability (U1–U6) and ethical design rules (E1–E3) extend the evaluation beyond compliance toward interaction clarity, cognitive ergonomics, and manipulation risk detection. Rather than functioning as a static checklist, each principle is encoded into a rule-specific evaluation pipeline that produces structured diagnostic evidence.

Second, reproducibility is prioritized wherever technically feasible. The deterministic layer, based on explicit conditional logic, static code parsing, and formal contrast computation, guarantees consistent outputs for identical inputs. This separation between deterministic analysis and probabilistic interpretation ensures methodological transparency when evaluating iterative outputs of generative systems.

Third, the framework adopts modality-aware evaluation. Source code inputs (ZIP archives or public GitHub repositories) enable structural inspection of elements, attributes, and styles, whereas screenshot inputs restrict evaluation to perceptual cues and may require controlled LLM-assisted reasoning. When structural artifacts are unavailable, the system assigns a "Not Evaluated" classification to prevent unreliable conclusions.

Fourth, the framework generates structured corrective prompts for confirmed violations. These prompts include contextualized diagnostic explanations and recommended remediation strategies designed to be tool-agnostic and reusable across generative platforms. This transforms the evaluator into an iterative refinement support mechanism.

Finally, the framework incorporates convergence measurement across iterations. Confirmed violations are tracked relative to a predefined threshold, enabling quantitative analysis of improvement patterns, violation persistence, and stabilization behavior. This convergence logic supports empirical benchmarking of generative UI systems under structured feedback conditions.

Together, these objectives define the scope of the proposed framework as a reproducible, modality-aware, and feedback-driven evaluation system for AI-generated user interfaces.

3.1.2 Architectural Design and Evaluation Workflow

The proposed framework follows a layered architectural design that separates input handling, rule execution, reasoning logic, and structured reporting. This modular organization ensures reproducibility, traceability, and controlled integration of deterministic and interpretive evaluation mechanisms.

At a high level, the evaluation workflow consists of five components: input processing, rule orchestration, deterministic analysis, LLM-assisted reasoning, and structured reporting with iteration tracking. These components operate sequentially within a single evaluation cycle.

The process begins with modality-aware input ingestion. The framework accepts exported source code archives (ZIP), public GitHub repositories, and static screenshots. Source-based inputs enable structural inspection of HTML, JSX/TSX components, and CSS rules through static parsing, whereas screenshot inputs restrict evaluation to visual and perceptual cues. The system routes each input to appropriate rule pipelines based on artifact availability, preventing execution of rules that require structural evidence when such evidence is absent.

Following input routing, the rule orchestration layer activates the selected rule set. Each rule is bound to a predefined evaluation method (deterministic, hybrid, or LLM-assisted), severity model, and output schema. This binding ensures consistent rule execution and structured result generation across evaluation runs.

The deterministic analysis engine constitutes the reproducible core of the framework. It operates through explicit conditional logic, static code parsing, pattern detection, and mathematical computation (e.g., WCAG luminance ratio calculation). Structural accessibility checks and code-based usability signals are validated within this layer. Identical inputs therefore produce identical outputs, ensuring stability across iterative evaluations.

Certain higher-order properties, such as perceptual grouping, recognition, recall balance, and manipulative choice architecture, require contextual interpretation beyond static pattern detection. In these cases, the framework invokes a structured LLM-assisted reasoning stage using rule-specific rubric prompts. Findings derived from this layer are treated conservatively to preserve methodological transparency.

Because LLM-based reasoning introduces probabilistic interpretation, the framework incorporates several safeguards to maintain analytical reliability. Prior to each model invocation, deterministic extraction mechanisms generate structured evidence describing relevant interface elements and interaction context. The language model is therefore instructed to evaluate predefined evidence rather than performing unrestricted analysis. In addition, rule-specific prompts impose explicit evaluation rubrics and structured output schemas, constraining the reasoning process and reducing the risk of hallucinated interpretations.

To preserve methodological transparency, findings produced through LLM-assisted reasoning are conservatively classified as *Potential* rather than *Confirmed* violations to account for probabilistic interpretation and potential reasoning errors. Confirmed violations are reserved exclusively for deterministically verified structural conditions.

The final stage of the workflow transforms findings into structured, severity-aware reports. Violations are classified as Confirmed, Potential, or Not Evaluated depending on evidentiary strength and modality constraints. Confirmed findings generate corrective prompts, while evaluation results are recorded to support iteration-level comparison.

Beyond single-pass auditing, the architecture incorporates convergence tracking. Confirmed violation counts are evaluated against a predefined threshold, enabling quantitative measurement of improvement across iterations. This transforms the evaluation process into a benchmarking mechanism rather than a static compliance check.

The overall conceptual architecture and evaluation workflow of the proposed framework is illustrated in Figure 3.1. A detailed technical implementation of these components is presented in Chapter 4.

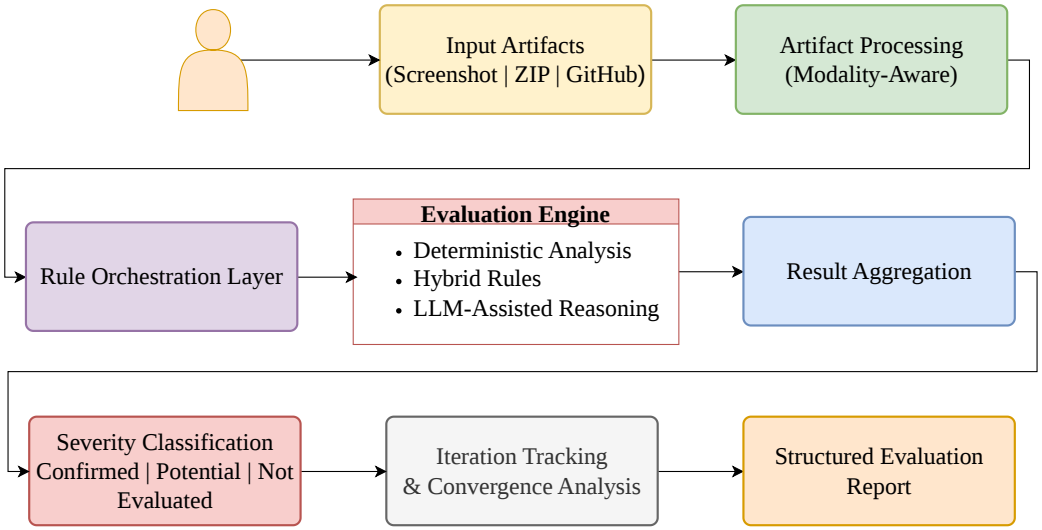

Figure 3.1: Conceptual architecture of the proposed UI Critic framework.

3.1.3 Input Modalities

The proposed framework supports multiple input modalities in order to evaluate AI-generated user interfaces under varying levels of structural visibility. Generative no-code systems may expose artifacts in different forms, including downloadable

project archives, repository links, or visual interface previews. To accommodate these variations, the framework accepts three input modalities: ZIP source archives, public GitHub repositories, and static interface screenshots.

The selection of these modalities reflects both the artifact formats typically produced by generative UI tools and the practical workflows of developers working with such systems. In common development scenarios, generated interfaces are accessed in one of three ways: exporting the generated project as a source package, sharing the implementation through a repository, or distributing visual previews during early design review. Supporting these formats enables the framework to evaluate both early-stage interface concepts and fully implemented applications. Screenshot support also accommodates situations in which developers cannot or prefer not to share source code, such as proprietary projects, private repositories, or preliminary design discussions.

ZIP archives and GitHub repositories represent source-based modalities that provide direct access to the structural representation of the interface. In these contexts, the framework performs static inspection of interface artifacts by parsing HTML, JSX/TSX components, and associated styling rules. Structural access allows rule evaluation to rely on verifiable programmatic evidence, enabling deterministic validation of properties such as semantic structure, keyboard operability, labeling associations, and accessible naming. Because these properties can be confirmed through structural inspection, violations detected under source-based modalities may be classified as deterministically verified findings.

GitHub repositories are operationally treated as an alternative representation of the same structural artifact as ZIP archives. When a repository URL is provided, the framework retrieves a snapshot of the project and processes it through the same normalization pipeline used for uploaded archives. This shared ingestion process ensures that repository-based and archive-based evaluations produce equivalent internal project representations, allowing rule execution to remain consistent across both modalities.

Screenshot inputs represent a visual-only modality in which structural artifacts are unavailable. In this context, the framework relies on LLM-assisted visual analysis to infer interface properties observable from rendered layouts. Because screenshots do not expose DOM elements, attributes, or event handlers, rules that require structural inspection cannot be executed. Such rules are therefore explicitly classified as Not Evaluated within the severity model. Findings derived from screenshot analysis are conservatively categorized as Potential in order to reflect the probabilistic nature of perceptual interpretation.

From a methodological perspective, the framework therefore distinguishes between structural evaluation contexts and visual evaluation contexts. Source-based inputs enable deterministic rule validation grounded in programmatic evidence, whereas screenshot-based inputs restrict evaluation to visually inferable signals.

Table 3.1: Supported input modalities and their evaluation capabilities.

Input Modality	**Structural Access**	**Evaluation Method**	**Limitations**
ZIP archive	Full source code	Deterministic + Hybrid + LLM	None
GitHub repository	Full source code	Deterministic + Hybrid + LLM	None
Screenshot	Visual layout only	LLM-assisted	Some rules are classified as Not Evaluated

This modality-aware design ensures that rule execution remains aligned with the available evidence and prevents incorrect assumptions of compliance when structural information is unavailable.

To maintain consistent evaluation behavior across modalities, the framework incorporates rule eligibility and suppression mechanisms. Certain rules are executed only when the required structural evidence is present, while others are omitted when modality limitations prevent reliable verification. By explicitly encoding these constraints, the framework preserves epistemic transparency and ensures that evaluation outcomes accurately reflect the evidentiary conditions under which analysis is performed.

To clarify the differences between the supported input modalities and their evaluation capabilities, Table 3.1 summarizes the level of structural access, evaluation method, and typical usage scenarios associated with each modality.

As shown in Table 3.1, source-based modalities (ZIP archives and GitHub repositories) provide full access to the interface structure and therefore enable deterministic and hybrid rule evaluation. In contrast, screenshot inputs provide only visual information, restricting analysis to LLM-assisted interpretation and preventing the execution of rules that require programmatic attributes. This distinction ensures that rule evaluation remains aligned with the evidence available in each modality.

3.2 Rule Taxonomy

This section presents the structured rule taxonomy that underpins the proposed evaluation framework. Building upon the findings of the systematic literature review, the taxonomy translates recurring compliance gaps, usability inconsistencies, and ethical risks in AI-generated interfaces into an operational rule-based model. Rather than serving as a generic checklist, the taxonomy formalizes theoretically grounded

design principles into executable evaluation logic, ensuring that each rule is both academically justified and computationally enforceable.

The overall structure of the rule taxonomy and its three evaluation dimensions is illustrated in Figure 3.2.

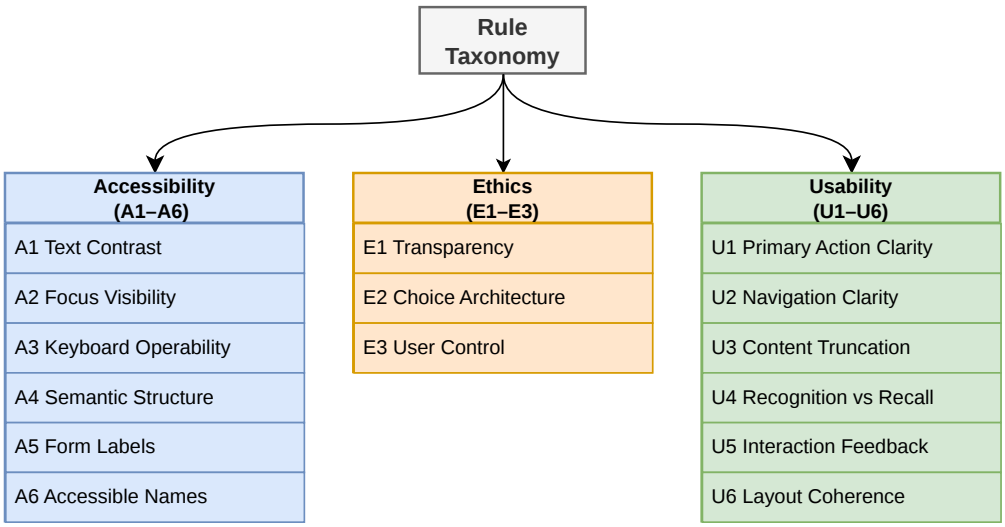

Figure 3.2: Rule taxonomy of the proposed framework organized into accessibility, usability, and ethical design dimensions.

Each dimension groups conceptually related rules that operationalize design principles derived from accessibility standards, usability heuristics, and ethical design literature.

3.2.1 Accessibility Rules (A1–A6)

The accessibility rule set operationalizes formal requirements derived from WCAG 2.1 into executable validation logic. While generative no-code platforms frequently produce visually coherent interfaces, compliance with structured accessibility standards cannot be assumed. In particular, deeper requirements such as semantic robustness, keyboard operability, and assistive technology compatibility require deterministic verification.

Six accessibility rules (A1–A6) are defined to collectively address perceptibility, operability, structural integrity, and programmatic robustness. When source artifacts (ZIP archives or GitHub repositories) are available, validation relies on static parsing and rule-based structural inspection. In screenshot-only contexts, evaluation is constrained to perceptual signals, and structurally dependent rules are explicitly marked as Not Evaluated (Input Limitation).

Accessibility functions as the compliance backbone of the framework. Confirmed violations within this category directly affect convergence status due to their grounding in formal WCAG requirements.

A1 – Text Contrast Compliance

A1 operationalizes WCAG 2.1 Success Criterion 1.4.3 (Contrast - Minimum, Level AA). It ensures that text elements meet minimum luminance contrast thresholds to support readability under reduced visual conditions.

For source-based inputs (ZIP or GitHub), contrast verification relies primarily on deterministic analysis. Foreground and background color tokens are extracted from styling declarations and contrast ratios are computed using the WCAG luminance formula. Confirmed violations are produced only when foreground color, background color, and font context can be resolved and the resulting ratio falls below the WCAG threshold.

When color resolution is uncertain—for example due to theme-dependent styling, opacity-modified elements, or unresolved background inheritance—the rule produces Potential findings rather than Confirmed violations.

In screenshot inputs, structural color information is unavailable. In this modality, contrast assessment relies on perceptual analysis of the rendered interface, and all findings are conservatively classified as Potential.

This rule replaces visual plausibility with measurable contrast compliance while maintaining conservative classification when structural certainty cannot be established.

A2 – Focus Visibility

A2 operationalizes WCAG 2.1 Success Criterion 2.4.7 (Focus Visible). It ensures that keyboard-operable elements provide a visible focus indicator to support navigation without a pointing device.

In source-based inputs, deterministic analysis detects cases where default focus indicators are removed without a sufficiently visible replacement. When focus styling is replaced with subtle visual changes, such as color or background shifts, the rule may produce Potential findings if the perceptual strength of the indicator cannot be reliably verified.

In screenshot inputs, focus visibility can only be assessed perceptually. Because keyboard interaction cannot be reproduced from static images, screenshot-based findings are conservatively classified as Potential.

Confirmed violations occur when focus indicators are structurally suppressed on focusable elements without any detectable replacement. When keyboard operability itself cannot be established, the rule may instead produce Potential findings.

A3 – Keyboard Operability

A3 operationalizes WCAG 2.1 Success Criterion 2.1.1 (Keyboard). It ensures that all interactive functionality can be accessed and operated using keyboard input.

In source-based inputs, deterministic analysis identifies interactive behavior implemented on non-semantic elements without appropriate keyboard support. Examples include pointer-driven handlers attached to non-interactive elements without corresponding keyboard event bindings, missing focusability, or improper tab navigation behavior. Violations are classified as Confirmed when structural evidence indicates that interactive functionality cannot be reached or operated through keyboard interaction.

In cases where partial accessibility signals are present, such as role assignments without corresponding keyboard handlers, the rule may produce Potential findings.

Because keyboard operability depends on DOM structure and event handling, it cannot be reliably inferred from static images. For screenshot-only inputs, the rule is therefore marked as Not Evaluated (Input Limitation).

A3 addresses functional accessibility by ensuring that interface interactions remain operable independently of pointer-based input devices.

A4 – Semantic Structure

A4 operationalizes WCAG 2.1 Success Criterion 1.3.1 (Info and Relationships) and related navigation principles. It ensures that document structure and interface relationships are programmatically determinable through semantic markup.

In source-based inputs, deterministic analysis evaluates page-level structural semantics. The rule detects cases where visual structure is implemented without appropriate semantic elements, such as heading-like text rendered without heading tags, interactive constructs lacking semantic roles, or repeated interface items presented without proper list semantics. Violations are classified as Confirmed when structural deficiencies are directly observable.

Additional signals, such as missing landmark regions, irregular heading hierarchies, or visually prominent page titles without corresponding semantic headings, may produce Potential findings when structural intent can be inferred but not definitively validated.

In screenshot-only contexts, semantic structure cannot be programmatically inspected and is therefore marked Not Evaluated (Input Limitation).

This rule ensures that interface structure is accessible not only visually but also programmatically to assistive technologies.

A5 – Form Control Labeling

A5 operationalizes WCAG requirements for programmatic labeling of form controls (Success Criteria 1.3.1 and 4.1.2). It ensures that input elements expose explicit labels that can be interpreted by assistive technologies.

In source-based inputs, deterministic analysis evaluates whether form controls are associated with a programmatically detectable label. The system verifies the presence of explicit <label> elements, valid label–control associations, or ARIA-based labeling mechanisms. Confirmed violations occur when native form controls lack any detectable labeling mechanism.

For interface components whose internal structure cannot be reliably inferred through static analysis, such as custom wrapper components or role-based input constructs, the rule may produce Potential findings when labeling cannot be confidently verified.

In screenshot-only contexts, label associations cannot be programmatically inspected and are therefore marked Not Evaluated (Input Limitation).

A5 ensures that visually identifiable form fields are also programmatically labeled, enabling assistive technologies to correctly interpret user input controls.

A6 – Accessible Name Exposure

A6 operationalizes WCAG 2.1 Success Criterion 4.1.2 (Name, Role, Value). It ensures that interactive elements expose programmatically determinable names that can be interpreted by assistive technologies.

In source-based inputs, deterministic analysis verifies whether interactive controls provide an accessible name through recognized mechanisms such as visible text content, ARIA labeling attributes, or referenced label elements. Violations are classified as Confirmed when interactive elements, such as buttons, links, or icon-only controls, lack any detectable name source.

For interface components whose internal implementation cannot be reliably inferred through static inspection, such as custom wrapper components, the rule may produce Potential findings when accessible name exposure cannot be confidently verified.

In screenshot-only contexts, accessible name resolution cannot be programmatically inspected and is therefore marked Not Evaluated (Input Limitation).

A6 ensures that interactive elements are not only visually recognizable but also programmatically identifiable to assistive technologies.

3.2.2 Usability Rules (U1–U6)

While accessibility rules target formal compliance with standardized requirements, usability rules address interaction clarity, cognitive efficiency, and task support.

In AI-generated no-code interfaces, usability issues may arise even when visual aesthetics appear coherent, as automated layout decisions can compress workflows, weaken hierarchy, or reduce orientation cues. To capture these quality dimensions, six usability rules (U1–U6) operationalize established HCI heuristics, particularly Nielsen's principles, into evaluable logic within the framework.

Where feasible, deterministic signals are used to detect structural indicators of usability risk (e.g., truncation patterns or missing feedback constructs). However, several usability properties depend on perceptual context (e.g., visual hierarchy or grouping coherence) and are therefore assessed via controlled LLM-assisted reasoning. Consistent with the severity model in Section 3.3, usability findings affect convergence only when supported by deterministic evidence; otherwise, they are reported as Potential to preserve epistemic transparency.

U1 – Primary Action Clarity

U1 evaluates whether the interface communicates a clear primary action within a given task context (e.g., submitting a form or confirming a selection). The rule targets cases where calls-to-action are visually or structurally indistinguishable from secondary controls, increasing cognitive effort and reducing task efficiency.

In source-based inputs, deterministic analysis detects structural signals indicating the absence of a clear primary action. Confirmed violations occur when form interactions lack a valid submission mechanism. Additional patterns, such as multiple visually equivalent calls-to-action or generic action labels lacking contextual cues, may produce Potential findings.

When structural evidence alone cannot determine interaction hierarchy, the framework applies controlled LLM-assisted reasoning to evaluate visual prominence and contextual clarity. These findings are reported as Potential.

In screenshot-only inputs, assessment relies entirely on perceptual analysis and is therefore reported conservatively as Potential.

U2 – Navigation Clarity and Orientation

U2 evaluates whether users can understand their current location within the interface and navigate effectively across sections or views. The rule targets incomplete navigation structures, weak labeling, and missing orientation cues.

In source-based inputs, deterministic signals inspect structural indicators of navigation support, such as the presence of navigation regions, link groupings, or route hierarchies in applications with multiple views. When structural signals suggest incomplete navigation support, the rule produces Potential findings.

When structural evidence alone cannot determine orientation clarity, the framework applies controlled LLM-assisted reasoning to evaluate navigation labeling, hierarchy, and contextual cues.

In screenshot-only inputs, assessment relies entirely on perceptual analysis and is therefore reported conservatively as Potential.

U3 – Content Truncation and Visibility

U3 evaluates whether interface content remains fully accessible to users without requiring hidden interactions or additional discovery effort. The rule targets situations where meaningful information is truncated, visually clipped, or otherwise inaccessible without an explicit recovery mechanism.

In source-based inputs, deterministic analysis detects structural patterns associated with content truncation or hidden information, such as constrained containers that suppress overflow or styling constructs that truncate text without providing scrolling or expansion controls. When meaningful content is structurally inaccessible, the finding is classified as Confirmed.

Additional patterns, such as nested scroll regions or content hidden without clear user controls, may produce Potential findings when the accessibility of the content cannot be conclusively determined.

When structural evidence is insufficient to determine whether content can be recovered, the framework applies controlled LLM-assisted reasoning to evaluate layout behavior and user access to hidden information.

In screenshot-only inputs, evaluation relies entirely on perceptual analysis and findings are therefore reported conservatively as Potential.

U4 – Recognition–Recall Balance

U4 evaluates whether interface interactions require users to recall information from memory instead of recognizing available options within the interface. The rule targets situations where system state, prior selections, or categorical choices are not visibly presented, increasing cognitive load during task completion.

The rule relies primarily on controlled LLM-assisted reasoning grounded in structural evidence extracted from the interface. This analysis evaluates whether relevant contextual information, selection states, or task progression cues remain visible during interaction.

Because recognition–recall balance depends on contextual interpretation rather than purely structural signals, U4 findings are conservatively classified as Potential. The rule does not produce Confirmed violations.

In screenshot-only inputs, evaluation similarly relies on perceptual analysis of the interface, and findings are therefore reported as Potential.

U4 operationalizes Nielsen's heuristic of recognition rather than recall by identifying interaction patterns that unnecessarily increase cognitive effort.

U5 – Interaction Feedback Visibility

U5 evaluates whether user actions receive clear and timely feedback from the interface. The rule targets situations where system responses to interactions, such as form submissions, asynchronous operations, or state changes, are not communicated to the user.

In source-based inputs, deterministic signals identify interaction handlers that may lack visible feedback mechanisms, such as loading indicators, disabled states, or confirmation messages. When interaction outcomes are not clearly communicated, the rule produces Potential findings.

When structural evidence alone cannot determine whether feedback mechanisms are implemented, the framework applies controlled LLM-assisted reasoning to evaluate interaction patterns and user awareness of system state.

Because interaction feedback occurs during runtime behavior, it cannot be reliably inferred from static screenshots. In screenshot-only inputs, the rule is therefore marked Not Evaluated (Input Limitation).

U5 operationalizes the usability principle that systems should keep users informed about the status and outcome of their actions.

U6 – Visual Grouping and Layout Coherence

U6 evaluates whether related interface elements are visually grouped and aligned in a way that supports clear interpretation of content relationships. The rule targets layouts where insufficient grouping, inconsistent alignment, or weak structural hierarchy reduces scannability and comprehension.

The rule relies on LLM-assisted reasoning grounded in structural layout signals extracted from the interface. These signals include the presence of content sections, grouping containers, and hierarchical headings that indicate how interface elements are organized.

Because layout coherence depends primarily on perceptual interpretation rather than purely structural evidence, U6 findings are conservatively classified as Potential.

In screenshot-only inputs, evaluation similarly relies on perceptual analysis and findings are therefore reported as Potential.

U6 operationalizes the usability principle that visually related information should be grouped and structured to support efficient scanning and understanding.

3.2.3 Ethical Design Rules (E1–E3)

Beyond accessibility compliance and usability quality, generative no-code systems introduce ethical considerations related to decision framing, transparency, and user autonomy. While automated interface generation may produce visually coherent

and task-efficient layouts, it can also reproduce persuasive defaults, asymmetrical choice structures, or reduced transparency in high-impact interactions.

To address this dimension, three ethical design rules (E1–E3) are defined. These rules draw on dark-pattern research, persuasive design literature, and responsible AI evaluation principles. Unlike accessibility rules (which enforce formal compliance) and usability rules (which focus on cognitive efficiency), ethical rules examine whether interaction structures preserve transparency, fairness, and user agency.

Because ethical evaluation is inherently contextual, these rules rely primarily on rubric-constrained LLM-assisted reasoning grounded in structural interface signals. Ethical findings are reported conservatively as Potential in order to preserve epistemic transparency.

Together, E1–E3 extend the framework toward responsible interface evaluation, ensuring that AI-generated interfaces are assessed not only for correctness and usability but also for the preservation of user autonomy and informed decision-making.

E1 – Transparency in High-Impact Actions

E1 evaluates whether actions with significant consequences—such as deletion, subscription commitment, financial transactions, or data sharing—are accompanied by sufficient disclosure and confirmation mechanisms.

The rule focuses specifically on high-impact interactions where users must clearly understand the consequences of their actions before committing. Interfaces should communicate the permanence, cost, or scope of such actions and provide an opportunity for users to review or cancel before execution.

Evaluation relies on LLM-assisted reasoning grounded in structural interface signals that indicate the presence of high-impact actions and their surrounding explanatory context. The assessment considers whether consequences are communicated clearly, whether confirmation mechanisms are present, and whether users retain an explicit path to cancel or reconsider the action.

Because the adequacy of disclosure and confirmation often depends on contextual interpretation, findings are reported as Potential.

In screenshot-only inputs, evaluation similarly relies on perceptual analysis and findings are therefore reported as Potential.

E2 – Imbalanced Choice Architecture in High-Impact Decisions

E2 evaluates whether high-impact decisions are presented through balanced and transparent choice structures. The rule targets interaction patterns where one option is visually or structurally emphasized in a way that may steer users toward a particular outcome.

Evaluation is limited to contexts involving consequential decisions, such as consent prompts, subscription commitments, payments, or irreversible operations. Within these contexts, the framework analyzes whether alternative options are presented with comparable visibility and clarity.

In source-based inputs, deterministic signals identify potential indicators of choice imbalance, such as disproportionate visual emphasis, default selections, or ambiguous alternatives. When these signals are present, the framework applies controlled LLM-assisted reasoning to assess whether the overall interaction structure may influence user choice.

Because the ethical interpretation of persuasive design patterns depends on contextual judgment, findings are reported conservatively as Potential.

In screenshot-only inputs, evaluation similarly relies on perceptual analysis and findings are therefore reported as Potential.

E2 operationalizes concerns from dark-pattern research by identifying interaction structures that may undermine balanced user decision-making.

E3 – User Control Restriction

E3 evaluates whether interfaces preserve meaningful user control during high-impact interactions, including the ability to cancel, exit, or reverse decisions. Ethical interaction design requires that users retain agency over navigation and state transitions when performing consequential actions.

Evaluation focuses on contexts involving high-impact operations such as deletion, subscription changes, financial confirmation, or account modification. Within these contexts, the rule assesses whether users are provided with a visible and accessible path to cancel, exit, or reconsider their actions.

In source-based inputs, deterministic signals identify structural indicators suggesting the absence of exit mechanisms, such as destructive actions presented without cancellation controls or dialog interactions lacking clear dismissal paths. When such patterns are detected, the framework applies controlled LLM-assisted reasoning to evaluate whether user control may be restricted.

Because the adequacy of exit mechanisms depends on contextual interpretation, findings are reported conservatively as Potential.

In screenshot-only inputs, evaluation relies on perceptual analysis and findings are therefore reported as Potential.

E3 operationalizes the ethical principle that interfaces should preserve user autonomy by maintaining clear and accessible exit paths during consequential interactions.

3.2.4 Coverage Scope and Design Rationale

The fifteen-rule taxonomy defined in Sections 3.2.1–3.2.3 is intentionally bounded rather than exhaustive. Its design balances three considerations: normative design standards, empirical findings from prior research, and methodological feasibility within an automated evaluation framework. The taxonomy therefore aims to (1) ensure minimum accessibility compliance, (2) capture recurring usability issues observed in AI-generated interfaces, and (3) identify interaction patterns that may pose high-impact risks to users.

Rule selection was guided by four complementary considerations:

- **Normative grounding.** Several rules map directly to established design standards, including WCAG 2.1 accessibility requirements, Nielsen's usability heuristics, and ethical design principles discussed in dark-pattern and responsible AI literature.
- **Empirical relevance.** The systematic literature review identified recurring issues in AI-generated interfaces, including incomplete WCAG compliance, missing semantic labels, weak interaction feedback, and navigation inconsistencies.
- **Exploratory observations.** Additional rules were motivated by issues observed during exploratory inspection of interfaces generated by contemporary no-code AI tools, such as truncated content, recognition-to-recall regressions, and destructive actions executed without confirmation.
- **Automatable detectability.** Rules were included only when their evaluation could be operationalized through deterministic static analysis or structured LLM-assisted reasoning without requiring runtime instrumentation or full user simulation.

Rules that could not be reliably operationalized under these constraints were excluded to maintain methodological coherence and implementation feasibility. Figure 3.3 illustrates how the final taxonomy was derived from three sources: established standards, findings from prior research, and exploratory inspection of AI-generated interfaces.

The resulting taxonomy evaluates three complementary dimensions of interface quality: accessibility, usability, and ethical interaction design.

Accessibility rules (A1–A6) operationalize WCAG 2.1 Level A/AA requirements and establish the accessibility compliance baseline of the framework. Usability rules (U1–U6) address interaction-level issues affecting task clarity, information accessibility, and interface coherence, reflecting recurring design problems reported in studies of AI-generated interfaces. Ethical rules (E1–E3) complement these

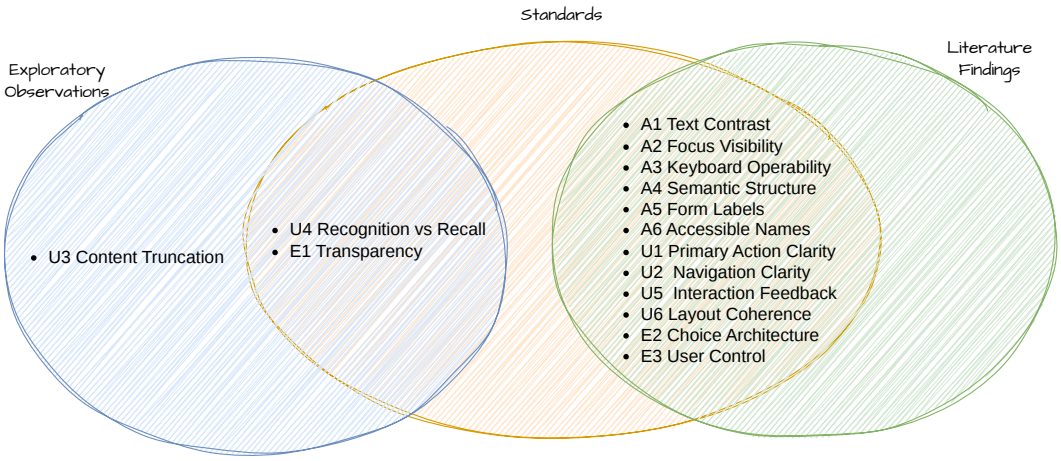

Figure 3.3: Sources of the proposed rule taxonomy.

dimensions by evaluating transparency in high-impact actions, balanced choice architecture, and preservation of user control.

The ethical dimension is intentionally limited to maintain detection feasibility and avoid overlap with usability rules, as many ethical concerns require behavioral or contextual analysis beyond static interface inspection. Future extensions may incorporate additional aspects such as privacy transparency, informed consent mechanisms, and fairness in recommendation systems.

Overall, the framework evaluates structural and interaction-level properties of AI-generated front-end interfaces by combining deterministic static analysis with controlled LLM-assisted reasoning.

To preserve methodological clarity, several aspects are intentionally excluded:

- WCAG Level AAA requirements
- Runtime behavior verification
- Content quality metrics
- Performance optimization metrics
- Native mobile platform conventions
- Backend logic or security evaluation

These exclusions reflect both technical constraints and the thesis focus on structural and interaction design properties of AI-generated web interfaces. The fifteen-rule taxonomy therefore represents a deliberate balance between analytical coverage and implementation feasibility.

3.3 Severity and Convergence Model

This section defines how evaluation outcomes are classified, interpreted, and integrated into iterative compliance assessment. While the rule taxonomy determines what aspects of an interface are evaluated, the severity and convergence model defines how detected findings are interpreted and how they influence longitudinal evaluation across iterations.

Rather than ranking issues according to perceived impact or heuristic weighting, the framework classifies findings based on the certainty and evidentiary basis supporting each detection. This approach establishes a clear boundary between deterministic structural validation, probabilistic interpretation, and modality-driven evaluation constraints. By grounding classification in evidence availability rather than subjective severity scoring, the model preserves methodological transparency and reproducibility while enabling stable comparison across successive evaluation iterations.

3.3.1 Severity Classification

Within the proposed framework, severity is conceptualized as an epistemic classification mechanism rather than a measure of perceived impact. The severity state of a finding indicates how confidently a violation can be asserted given the available evidence and the input modality under which evaluation occurs.

Conventional auditing tools frequently conflate issue seriousness with detection certainty. In contrast, the proposed model explicitly separates these two dimensions. Severity therefore reflects the reliability of the supporting evidence rather than the magnitude of the potential user impact.

Evaluation outcomes are categorized into three mutually exclusive states:

- Confirmed (Blocking) : A structural violation supported by deterministic evidence.
- Potential (Non-blocking) : An interpretive or probabilistic assessment requiring contextual reasoning.
- Not Evaluated (Input Limitation) : A rule that cannot be executed due to limitations of the provided artifact.

Evaluation outcomes are categorized into three mutually exclusive states, summarized in Table 3.2.

Together, these categories encode three core methodological principles: deterministic precedence, interpretive delimitation, and modal boundary recognition.

Table 3.2: Severity states used in the evaluation framework.

Severity State	**Evidence**	**Interpretation**	**Convergence**
Confirmed	Deterministic structural evidence	Verified violation	Affects convergence
Potential	LLM-assisted signals	Probabilistic risk	Advisory only
Not Evaluated	Evidence unavailable (modality)	Evaluation not possible	Excluded

Deterministic Precedence When structural evidence is directly observable, through static parsing, attribute validation, or mathematical computation, a violation is classified as Confirmed. These findings originate from explicit rule-bound logic and are reproducible under identical input conditions.

Confirmed classification is therefore reserved exclusively for violations grounded in verifiable structural conditions. Interpretive reasoning alone cannot produce a Confirmed outcome. This ensures that blocking findings remain strictly tied to deterministic validation.

Interpretive Delimitation Certain design properties, such as perceptual grouping, recognition–recall balance, or manipulative interaction framing, cannot be conclusively validated through structural inspection alone. In such cases, evaluation relies on rubric-guided LLM-assisted reasoning.

Because this reasoning is inherently probabilistic, resulting findings are classified as Potential unless corroborated by deterministic structural signals. Potential findings therefore serve a diagnostic and advisory role, highlighting possible design risks while preserving transparency regarding evidentiary limits.

Modal Boundary Recognition The availability of evidence depends on the input modality used for evaluation. Source-based artifacts provide structural access to DOM elements and programmatic attributes, whereas screenshot-based inputs restrict analysis to visually inferable interface signals.

When a rule requires structural artifacts that are unavailable in the selected modality, the rule is classified as Not Evaluated. This explicit acknowledgment of modality constraints prevents incorrect assumptions of compliance and ensures transparent disclosure of evaluation scope.

Rule Independence and Stability Each rule assigns severity independently within its own evaluation pipeline. The three severity states are mutually exclusive

per finding, and classifications do not escalate across evaluation dimensions such as accessibility, usability, or ethical design.

To preserve longitudinal stability, deterministically Confirmed findings are not downgraded across iterations unless structural remediation occurs. Rule-specific precedence and suppression mechanisms prevent redundant detections and maintain consistent classification behavior across evaluation cycles. The technical implementation of these mechanisms is described in Chapter 4.

Severity Model Characteristics The severity model encodes several methodological properties of the evaluation framework:

- the source of evidence (structural vs. interpretive)
- the evaluation modality (code-based vs. visual)
- the distinction between deterministic and probabilistic reasoning
- the eligibility of findings to influence convergence

By structuring severity around epistemic certainty rather than perceived impact, the framework maintains methodological transparency while enabling structured evaluation across heterogeneous input conditions.

3.3.2 Convergence Model

The Convergence Model defines how severity classification interacts with iterative evaluation and compliance determination. Rather than relying on total issue counts, weighted scoring, or probabilistic confidence measures, convergence is determined exclusively by the number of Confirmed (Blocking) violations detected during evaluation.

In this thesis, an iteration corresponds to a single UI artifact, either an initial generated interface or a regenerated version, evaluated under a fixed rule set and threshold configuration. Each iteration produces a structured evaluation report snapshot, enabling longitudinal comparison across successive refinement cycles.

Deterministic Dependency Only Confirmed findings contribute to convergence status. Potential and Not Evaluated findings are explicitly excluded from convergence calculation. This design ensures that convergence reflects verified structural remediation rather than interpretive variability or modality-driven evaluation limitations.

Formally, convergence is defined as follows.

Let

$$C_i = \text{number of confirmed violations at iteration } i$$

$$T = \text{user-defined blocking threshold}$$

Convergence occurs when:

$$C_i \leq T$$

Equivalently, the convergence status for an iteration can be expressed as

$$\text{Convergence}(i) = \begin{cases} \text{true}, & \text{if } C_i \leq T \\ \text{false}, & \text{otherwise} \end{cases} \tag{3.1}$$

The threshold parameter allows experimental flexibility. For example, strict compliance may be defined by $T = 0$, while tolerance-based evaluation may allow a limited number of remaining violations ($T > 0$) without preventing convergence. Figure 3.4 illustrates the iterative convergence evaluation process defined by this model.

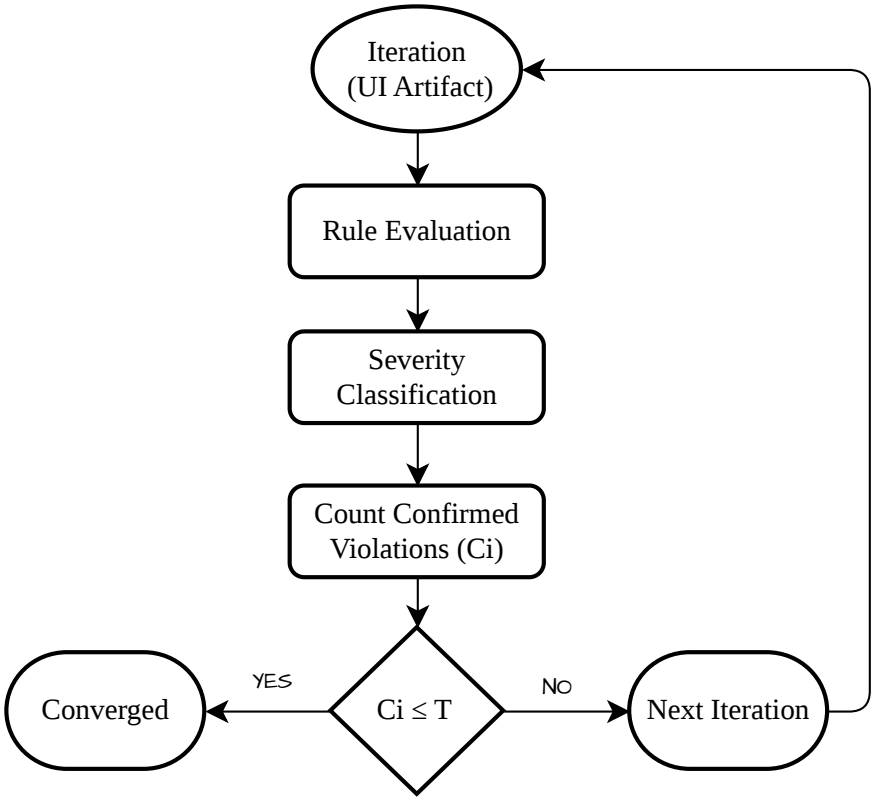

Figure 3.4: Convergence evaluation workflow.

Longitudinal Stability Because Confirmed findings originate from deterministic structural evidence and are governed by rule-level stability policies, changes in convergence status directly reflect structural remediation rather than classification noise. This property enables meaningful comparison across iterative refinements of AI-generated interfaces.

Independence from Interpretive Variability LLM-assisted or probabilistic findings do not influence convergence status. By excluding interpretive findings from convergence calculation, the framework ensures that compliance measurement remains reproducible and independent of probabilistic reasoning variability.

Methodological Significance By anchoring convergence exclusively to deterministic evidence, the model establishes a clear boundary between structural compliance and interpretive design risk. This transforms the evaluation process from a static audit into a structured, threshold-governed improvement mechanism suitable for longitudinal experimentation.

3.4 Summary

This chapter presented the proposed framework for evaluating AI-generated user interfaces. The framework combines deterministic structural analysis with LLM-assisted reasoning to assess interface quality across accessibility, usability, and ethical design dimensions.

A fifteen-rule taxonomy was introduced to operationalize these evaluation criteria, translating established design standards and prior research findings into executable detection logic. The framework also supports multiple input modalities—including ZIP archives, GitHub repositories, and screenshots—allowing interfaces to be evaluated at different stages of development.

Finally, the Severity and Convergence Model was defined to classify findings and determine convergence based on confirmed structural violations. Together, these components establish a structured and reproducible methodology for evaluating AI-generated interfaces. The next chapter describes the technical implementation of the framework.

Chapter 4

Proof of Concept Implementation

This chapter describes the technical implementation of the UI Critic framework as a functional proof-of-concept system. While Chapter 3 introduced the conceptual design and evaluation methodology, the present chapter explains how those principles are realized in a working software architecture.

The system is implemented as a client-side single-page application (SPA) coupled with serverless edge functions responsible for artifact processing and rule evaluation. The architecture follows a stateless request–response model in which each evaluation request is processed independently. Deterministic static analysis and LLM-assisted reasoning are integrated within the evaluation pipeline to enable modality-aware assessment of AI-generated user interfaces while maintaining minimal infrastructure complexity.

4.1 Technology Stack

The UI Critic prototype was implemented using a modern web development stack designed to support rapid prototyping, serverless execution, and integration with large language models. The system was developed using the Lovable development platform, which provides an integrated environment for building AI-assisted web applications and deploying backend logic through serverless edge functions.

The frontend of the system is implemented as a React-based single-page application (SPA) responsible for artifact submission and visualization of evaluation results. Backend processing is executed through serverless edge functions running in a Deno runtime environment, which handle artifact ingestion, rule execution, and response generation. The system also integrates external services, including the GitHub API for repository retrieval and the Gemini large language model for

LLM-assisted rule evaluation.

Table 4.1 summarizes the main technologies used in the implementation of the UI Critic prototype and their respective roles within the system.

Table 4.1: Technology stack used in the implementation of the UI Critic prototype system.

Layer	Technology	Role
Frontend	React 18, Vite, TypeScript	Implementation of the single-page application used for artifact submission and visualization of evaluation results.
UI Components	shadcn/ui, Radix UI, Tailwind CSS	Accessible component library and design system used to construct the user interface.
State Management	Zustand (persist middleware)	Client-side management of projects, iterations, and evaluation results with local storage persistence.
Routing	React Router v6	Client-side navigation within the single-page application.
Backend Runtime	Deno (Supabase Edge Functions)	Serverless execution environment responsible for artifact ingestion and rule evaluation.
LLM Integration	Google Gemini (via Lovable AI Gateway)	LLM-assisted reasoning used for perceptual interface analysis and hybrid rule evaluation.
Image Processing	ImageScript	Screenshot processing used for image-based accessibility analysis.
Hosting Infrastructure	Lovable Cloud (Supabase infrastructure)	Deployment environment for the application and execution of serverless backend functions.

The implementation of the UI Critic framework is publicly available in an open-source repository on GitHub [54].

4.2 System Architecture

This section describes the system architecture of the UI Critic framework and explains how the conceptual evaluation model introduced in Chapter 3 is realized in the implemented system. The architecture is designed to enable reproducible evaluation of heterogeneous UI artifacts while maintaining a clear separation between artifact ingestion, rule execution, and result visualization.

The system follows a distributed design consisting of a client-side application responsible for artifact submission and result visualization, and a serverless backend that performs artifact processing and rule evaluation. The framework supports three input modalities, ZIP archives, GitHub repositories, and screenshots, which are processed through modality-aware ingestion pipelines before entering the evaluation workflow. The overall implementation architecture is illustrated in Figure 4.1

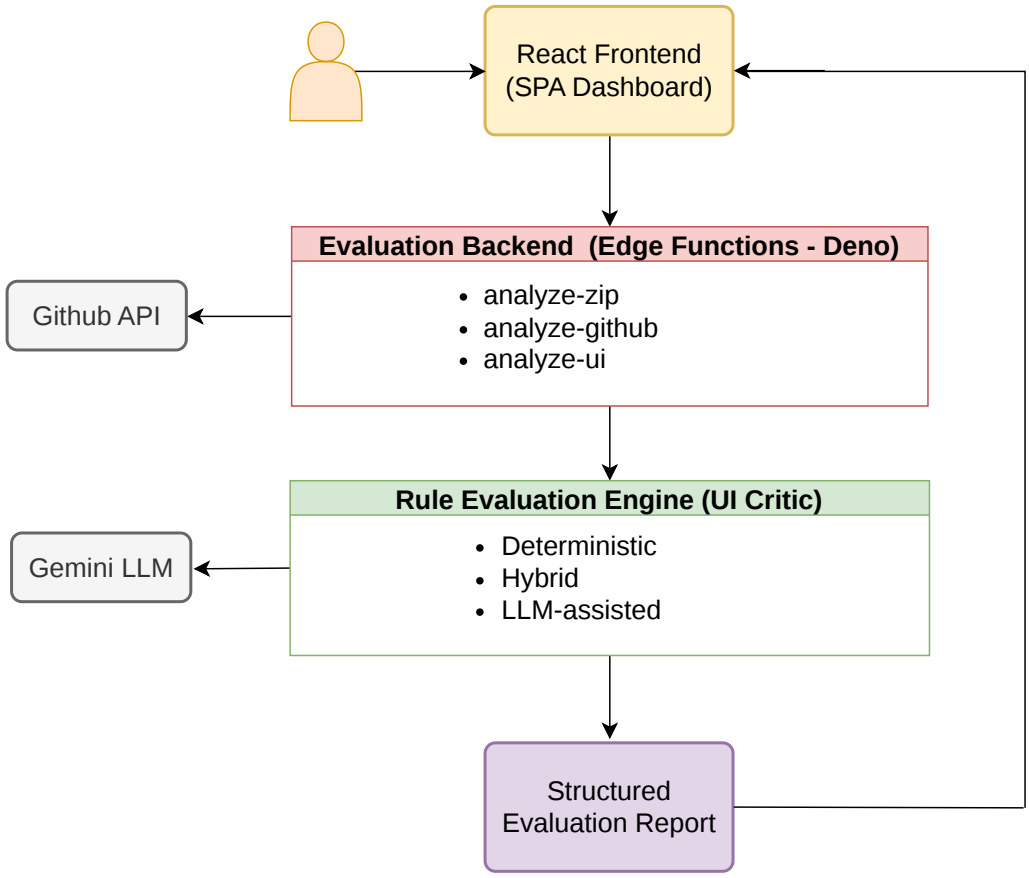

Figure 4.1: Implementation architecture of the UI Critic prototype, showing the React frontend, evaluation backend, rule evaluation engine, and external integrations with GitHub and Gemini.

Rather than relying on a centralized backend service or persistent server-side state, the framework adopts a stateless serverless execution model. Each evaluation request is handled independently by an edge function that processes the submitted artifact, executes the rule evaluation pipeline, and returns structured analysis results to the client.

The following subsections describe the frontend architecture and backend execution model that together implement the evaluation workflow of the framework.

4.2.1 Frontend Architecture

The frontend is implemented as a React-based single-page application designed to support interaction with evaluation results generated by the backend. Its primary role is to allow users to submit evaluation artifacts, visualize detected violations, and manage iteration histories during experimental evaluation.

All rule evaluation logic is executed in backend edge functions. The frontend does not perform source parsing, rule detection, or LLM-assisted reasoning. Instead, it submits evaluation artifacts, receives structured results, and renders violation reports according to the severity model defined in Chapter 3.

The interface organizes evaluation sessions using a hierarchical structure consisting of projects, iterations, and analyses. A project represents a UI artifact under evaluation, while each iteration corresponds to a refinement cycle of that artifact. Every iteration produces an independent evaluation snapshot, enabling longitudinal comparison of violations across successive refinements.

Application state is managed using a lightweight client-side state container with persistence provided through browser local storage. Persistent data includes projects and their associated iterations, while transient interface states remain in memory. When evaluation results are returned from the backend, they are attached to the corresponding iteration and used to recompute convergence status based on Confirmed violations, as defined in Section 3.3.2.

Client-side persistence was intentionally selected to support the research-oriented and proof-of-concept nature of the framework. Storing evaluation artifacts locally simplifies experimental replication, avoids the need for backend databases, and ensures that evaluation sessions remain independent.

By restricting the frontend to artifact submission, result visualization, and iteration management, the architecture maintains a clear separation between analytical computation and user interaction, improving transparency and reproducibility of the evaluation process.

4.2.2 Backend Architecture

The backend component of the framework is implemented as a set of serverless edge functions executed within a Deno runtime environment. These functions encapsulate the rule evaluation engine and are responsible for artifact ingestion, rule execution, and structured response generation. While the frontend manages artifact submission, visualization, and local persistence, all analytical computation is performed within the backend.

The system follows a stateless request–response execution model. Each evaluation request invokes an independent edge function that processes the submitted artifact, executes the rule evaluation pipeline, and returns a structured analysis result. No persistent server-side state or database storage is used; all persistence occurs client-side. This design simplifies infrastructure while ensuring that each evaluation represents an isolated and reproducible analysis event.

Three modality-specific edge functions support the supported input types: `analyze-zip`, `analyze-github`, and `analyze-ui`. The first two process source-based artifacts (ZIP archives and GitHub repositories), while the third handles screenshot inputs. ZIP and GitHub artifacts are normalized through a shared ingestion pipeline that filters unsupported files, standardizes paths and line endings, and constructs a canonical project snapshot used by the rule engine. GitHub repositories are downloaded as ZIP archives before processing to ensure parity with direct ZIP uploads.

After ingestion, artifacts are processed by the rule evaluation engine. The evaluation pipeline begins with a deterministic detection phase that applies static analysis to source files. This stage identifies structural violations using rule-specific logic and produces reproducible findings. Rules requiring contextual interpretation are evaluated through an LLM-assisted phase using a batched inference call to a Gemini model. Hybrid rules rely on deterministic signals to determine whether LLM reasoning should be invoked, ensuring that interpretive analysis is applied only when structural evidence is insufficient.

Screenshot inputs follow a different processing path. Images are decoded and submitted to the LLM for perceptual analysis of interface structure and design patterns. Because screenshots provide no access to source code or runtime behavior, several rules (A3–A6 and U5) cannot be evaluated and are explicitly reported as *Not Evaluated*. Additionally, screenshot findings are restricted to the *Potential* classification, since deterministic structural verification is not available.

For accessibility rule A1 (text contrast), the system uses deterministic contrast computation when analyzing source code artifacts by resolving color tokens to hex values and applying luminance-based contrast calculations. In contrast, screenshot-based A1 evaluation relies entirely on LLM perceptual assessment rather than pixel-level measurement, as pixel sampling was disabled to reduce false positives

caused by image-based estimation uncertainty.

Following rule execution, the backend performs aggregation, deduplication, and cross-rule suppression to consolidate overlapping findings into rule-level violation groups. The resulting analysis is assembled into a structured JSON response containing detected violations, evaluation metadata, and snapshot information. This response is returned to the frontend, where it is associated with the corresponding project iteration and rendered in the reporting interface.

By isolating rule execution within stateless edge functions and separating it from client-side persistence and visualization, the architecture maintains a clear boundary between analytical computation and presentation while enabling consistent evaluation across heterogeneous UI artifacts.

4.3 Evaluation Pipeline

This section describes the evaluation pipeline that operationalizes the framework introduced in Chapter 3. While Section 4.2 presented the architectural components of the system, this section focuses on the execution workflow through which input artifacts are transformed into structured evaluation results (Figure 4.2).

The pipeline begins with the submission of an artifact through the frontend interface, implemented as a React-based dashboard. The interface allows users to upload screenshots, ZIP archives, or GitHub repository links, configure the evaluation rules, and define the convergence threshold used during iterative refinement. Once submitted, the artifact is transmitted to the backend evaluation layer, implemented through Supabase Edge Functions, which initiate the analysis process.

During the input ingestion stage, the system decodes the artifact, filters relevant source files, and constructs a normalized project snapshot suitable for analysis. This representation is then processed by the rule evaluation engine, which translates the rule taxonomy defined in Chapter 3 into executable detection logic. The engine follows a deterministic-first evaluation strategy: structurally verifiable rules are evaluated through static analysis, while rules requiring contextual or perceptual interpretation are assessed through hybrid or LLM-assisted reasoning.

Following rule execution, the detected findings are processed by the result aggregation stage. This stage groups related violations, suppresses cross-rule duplicates, and assigns severity classifications to produce a structured evaluation output. The final evaluation report, containing detected violations, corrective prompts, and convergence status, is returned to the user and may be used to iteratively refine the evaluated interface until the predefined convergence threshold is satisfied.

The following subsections describe each stage of the pipeline in detail, including artifact ingestion, rule evaluation phases, and result aggregation.

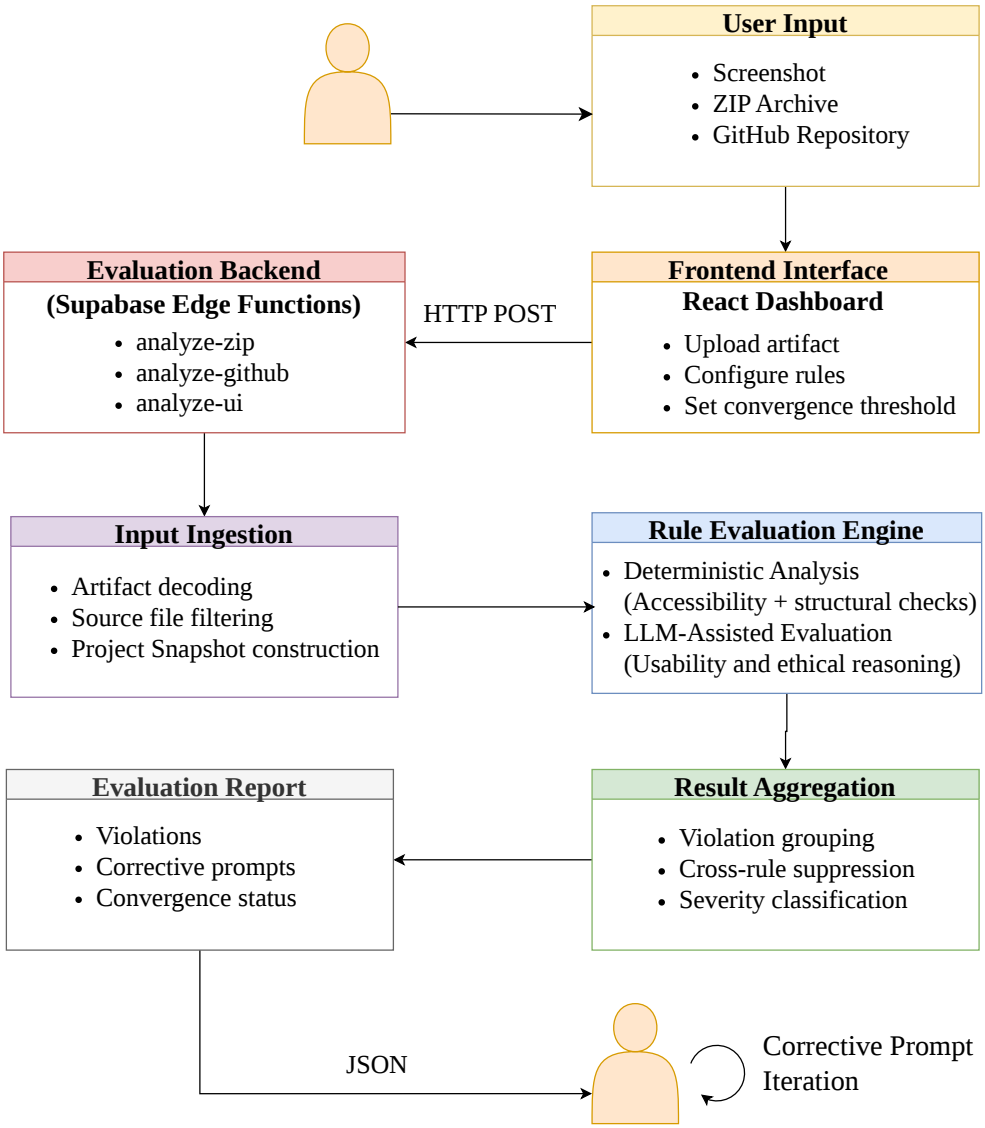

Figure 4.2: Evaluation pipeline of the UI Critic framework showing artifact submission, evaluation stages, result aggregation, and iterative refinement.

4.3.1 Input Ingestion

The ingestion stage validates submitted artifacts and converts them into normalized internal representations suitable for rule evaluation. UI Critic supports three input modalities: screenshot images, ZIP source archives, and public GitHub repository

URLs.

The interface used to submit these artifacts is shown in Figure 4.3. The analysis interface allows users to select the input modality, upload the artifact or provide a repository link, configure the convergence threshold, and choose which rule categories should be evaluated before initiating the analysis.

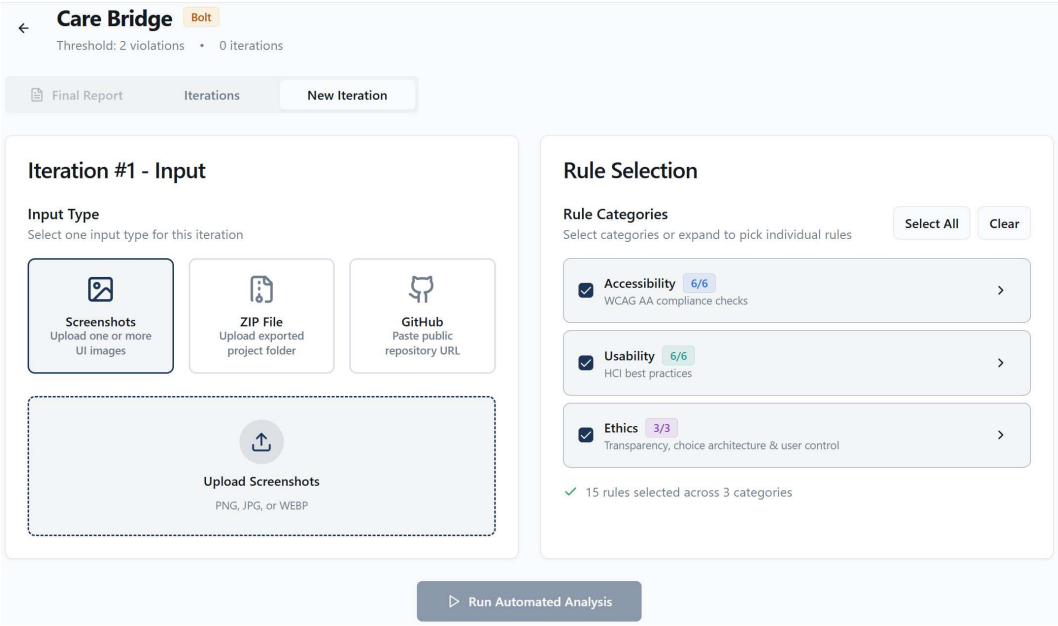

Figure 4.3: Input interface of the UI Critic framework, illustrating the three sup ported evaluation modalities and rule-category selection before analysis execution.

Each modality is processed through a dedicated backend entry point. Source-based artifacts are handled by the `analyze-zip` and `analyze-github` edge functions, while screenshot-based analysis is performed by the `analyze-ui` function.

ZIP Archive Processing

ZIP-based inputs are decoded and extracted in memory. During extraction, the ingestion pipeline filters files according to supported front-end extensions and excludes non-analytical directories such as dependency folders, build outputs, binary assets, and metadata files. Extracted files are normalized with respect to path format and content encoding before being stored in an internal project snapshot used by the rule engine.

To maintain computational stability, the system applies both per-file and total-size limits during extraction. This ensures that analysis remains focused on static interface artifacts rather than full development environments.

GitHub Repository Processing

Public GitHub repositories are ingested by downloading the repository as a ZIP archive and passing it through the same normalization pipeline used for uploaded ZIP files. Both source-based modalities therefore produce the same normalized snapshot representation and are evaluated under identical processing rules. Private repositories are not supported in the current implementation.

Screenshot Processing

Screenshot inputs enable evaluation when structural source artifacts are unavailable. Because screenshots do not expose underlying markup, DOM structure, or runtime behavior, rules requiring programmatic inspection of source artifacts—such as keyboard operability, semantic HTML structure, ARIA attributes, or form-label associations—are classified as Not Evaluated for this modality.

Screenshots are uploaded through the client interface as image files and encoded in the browser as Base64 data URLs. These encoded images are transmitted to the analyze-ui Edge Function as part of the evaluation request. Consistent with the stateless architecture of the framework, images are processed entirely in memory and are not persisted after the request completes.

Within the backend environment, the received images are forwarded to a vision-capable language model for perceptual analysis. The model evaluates visually observable interface properties and returns structured outputs aligned with the rule taxonomy. This enables detection of usability and ethical design concerns such as weak visual grouping, unclear primary actions, or potentially manipulative choice architecture.

Because screenshots represent static interface states and do not allow interaction simulation, dynamic behaviors such as focus transitions, hover states, loading indicators, or keyboard navigation cannot be directly observed. As a result, screenshot-derived findings are always classified as Potential rather than Confirmed violations. A confidence-gating mechanism is applied to filter model outputs, retaining only findings exceeding the minimum confidence threshold.

The screenshot modality therefore provides a perceptual evaluation layer that complements source-based structural analysis, enabling visually observable interface characteristics to be evaluated even when only image artifacts are available.

4.3.2 Deterministic Detection Phase

The deterministic detection phase constitutes the first analytical stage of rule execution for source-based inputs. It operates on statically parsed source artifacts (e.g., HTML, JSX/TSX, CSS, and extracted utility declarations) and applies rule-specific structural pattern matching and threshold-based validation. This stage

forms the reproducible analytical core of the evaluation framework.

Rule Evaluation Model The rule engine evaluates the fifteen heuristics defined in the framework through a three-tier evaluation model that combines deterministic structural analysis with LLM-assisted reasoning. Table 4.2 summarizes the evaluation tiers and their corresponding mechanisms.

Table 4.2: Rule engine evaluation tiers and corresponding detection mechanisms.

Tier	Rules	Mechanism
Deterministic	A1–A6	Static structural analysis based on observable code artifacts such as semantic elements, ARIA attributes, focus indicators, and form-label associations. These checks are fully reproducible and may produce *Confirmed* violations when structural requirements are not satisfied.
Hybrid	U1–U3, U5, E3	Deterministic signal extraction followed by LLM fallback when structural evidence is insufficient to determine a definitive classification.
LLM-assisted	U4, U6, E1, E2	Contextual and perceptual reasoning performed through a language model, applied when rule evaluation depends primarily on interpretive interface understanding.

Within the deterministic phase, all accessibility rules (A1–A6) are evaluated directly through static structural inspection of the parsed source artifacts. These checks rely on explicit signals observable in the code, including semantic HTML structure, keyboard accessibility attributes, focus indicators, label associations, and accessible naming patterns.

In addition, deterministic structural signals are extracted for selected hybrid rules (U1, U2, U3, U5, and E3). These signals represent structural evidence that may either directly produce findings when violations are clearly observable or serve as inputs to subsequent interpretive evaluation. Rules whose detection depends primarily on contextual or perceptual reasoning (U4, U6, E1, and E2) are evaluated in the LLM-assisted phase described in Section 4.3.3.

Execution Model Each rule is implemented as an independent detection unit operating over the parsed artifact set. Structural conditions are evaluated against

relevant elements, and detected violations are recorded in a shared results structure. Execution proceeds sequentially within a single evaluation invocation. All deterministic rules are executed prior to interpretive analysis, ensuring that structural evidence is established before higher-level reasoning is applied.

Output and Convergence Alignment Each deterministic finding produces a structured result containing the rule identifier, classification status (*Confirmed* or *Potential*), and contextual metadata describing the detected condition. When a violation is structurally verified and classified as *Confirmed*, it is marked as convergence-relevant in accordance with the severity model defined in Section 3.3. Only such structurally grounded findings contribute to blocking convergence.

Methodological Characteristics The deterministic phase operates exclusively on statically observable structural evidence, produces all *Confirmed* (blocking) findings, and establishes the evidentiary foundation for subsequent interpretive evaluation. By ensuring that convergence-relevant outcomes originate solely from reproducible structural conditions, the framework preserves analytical stability across evaluation iterations.

4.3.3 LLM-Assisted Detection Phase

Following deterministic evaluation, the framework executes an LLM-assisted detection phase for rules that require contextual or perceptual reasoning and cannot be resolved through static structural analysis alone. This stage extends deterministic enforcement to higher-level usability and ethical design concerns.

Rule Coverage The LLM-assisted phase evaluates rules whose detection depends primarily on interpretive reasoning:

- U4 — Recognition–Recall Balance
- U6 — Visual Grouping and Layout Coherence
- E1 — Insufficient Transparency in High-Impact Actions
- E2 — Manipulative Choice Architecture

The language model may also be invoked for hybrid rules when deterministic signals are insufficient to determine a definitive classification. These hybrid rules include U1, U2, U3, U5, and E3, where deterministic analysis first extracts structural evidence that may guide subsequent interpretive evaluation.

Rule Execution Modes Rules in the framework follow three execution modes depending on the role of deterministic evidence in the evaluation process. These execution modes apply to source-based artifacts such as ZIP archives and GitHub repositories, where structural information is available for deterministic preprocessing. In contrast, screenshot-based evaluation does not expose programmatic interface artifacts such as HTML structure or CSS properties. Consequently, deterministic analysis cannot be applied and all rules are evaluated through vision-based LLM interpretation.

Deterministic rules (A1–A6). Accessibility rules are evaluated entirely through static structural analysis and do not invoke the language model.

Hybrid rules (U1, U2, U3, U5, E3). For hybrid rules, deterministic extractors operate as the primary evaluation mechanism. These extractors analyze parsed source artifacts and attempt to resolve violations directly through structural signals. The language model is invoked only when deterministic signals are ambiguous or insufficient to determine a definitive classification.

LLM-assisted rules (U4, U6, E1, E2). For rules that depend primarily on contextual or perceptual interpretation, deterministic extractors construct structured evidence describing relevant interface properties such as layout structure, container organization, interaction cues, and textual signals. This evidence is incorporated into the LLM prompt, allowing the model to reason over verified interface signals rather than raw source code.

This layered execution strategy ensures that language-model reasoning remains grounded in deterministic preprocessing while enabling the detection of higher-level usability and ethical design patterns that cannot be derived from structural analysis alone.

Consequently, screenshot findings originate entirely from the language model, whereas source-based evaluation allows deterministic preprocessing to guide LLM-assisted reasoning and improve reproducibility.

Batched Interpretive Execution All eligible interpretive rules are evaluated within a single model invocation per evaluation cycle. This batched execution ensures consistent reasoning across related rule assessments while avoiding multiple independent model calls.

Prompt Design and Structured Output The language model is guided by rule-specific evaluation rubrics and instructed to return outputs in a predefined structured schema. Each prompt specifies the rule conditions to be evaluated and requires the model to produce a classification, confidence score, and diagnostic explanation for each rule.

To reduce hallucination risk, prompts incorporate structured evidence extracted during deterministic analysis prior to model invocation. These evidence bundles

describe relevant interface characteristics such as layout structure, element hierarchy, and interaction cues. The model therefore evaluates predefined evidence rather than performing unrestricted analysis of the source artifacts.

In screenshot-based evaluation, where deterministic evidence cannot be extracted, an additional confidence-gating mechanism is applied to reduce hallucinations. Only findings with a confidence score above 0.60 are retained, while lower-confidence outputs are suppressed. This filtering step helps prevent speculative visual interpretations from being included in the final report.

Generation parameters are configured conservatively to reduce stochastic variation in model outputs. In particular, the temperature parameter is set to a low value ($T = 0.3$), encouraging responses that remain closely aligned with the deterministic evidence provided in the prompt.

Epistemic Constraints To preserve methodological rigor, LLM-derived findings are subject to explicit epistemic constraints:

- They cannot produce *Confirmed* (blocking) violations.
- They do not contribute to convergence status.
- They cannot override deterministic findings.

All LLM-derived results are therefore classified as *Potential* findings rather than *Confirmed* violations to account for the probabilistic nature of language-model reasoning. These findings serve as advisory extensions of deterministic analysis.

Methodological Characteristics The LLM-assisted phase integrates probabilistic reasoning within a controlled evaluation pipeline. Deterministic preprocessing, structured prompting, confidence gating, and non-blocking classification ensure that interpretive analysis complements deterministic rule detection while maintaining reproducibility and analytical stability across evaluation runs.

4.3.4 Result Aggregation and Suppression

Following deterministic evaluation and LLM-assisted analysis, all findings are consolidated during the aggregation stage. This stage transforms internal detection outputs into a structured response schema while applying rule-level deduplication, severity assignment, and cross-rule suppression. The objective is to ensure that reported violations correspond to distinct interface issues rather than overlapping or derivative detections.

The aggregated response contains overall execution status, a collection of rule-level violations, auxiliary notes, and contextual metadata (e.g., number of analyzed

files and environment characteristics). Each entry in the `violations` collection corresponds to one rule evaluated during the analysis cycle. Classification and convergence eligibility are assigned at the rule level rather than per individual element, ensuring consistency with the rule taxonomy defined in Chapter 3.

Severity Assignment During aggregation, each rule outcome is assigned a severity classification according to the epistemic model defined in Section 3.3. This classification determines whether a finding contributes to convergence evaluation or serves only as an advisory signal. Table 4.3 summarizes the severity states applied during result aggregation.

Table 4.3: Severity states applied during result aggregation.

Classification	Meaning	Blocking?	Eligible Rules
Confirmed	Structurally verifiable violation derived from deterministic code or DOM evidence.	Yes, blocks convergence	A1–A6 (source-based), U1.1, U3 when structural signals are unambiguous.
Potential	Heuristic or perceptual assessment derived from LLM-assisted analysis or ambiguous structural signals.	No	All remaining usability and ethics rules; all screenshot-based findings.
Not Evaluated	Rule cannot be executed due to input modality limitations.	No	Rules requiring structural artifacts unavailable in the selected modality (e.g., A3–A6 and U5 for screenshot inputs).

Only findings classified as *Confirmed* contribute to convergence evaluation. Because confirmed findings originate exclusively from deterministic structural validation, LLM-assisted rules cannot produce confirmed violations and therefore do not influence convergence status.

Rule-Level Aggregation For rules operating at the element level (e.g., A1–A6 and selected usability rules), the aggregated rule object contains a collection of element findings. Each element entry includes identifying metadata, relevant

computed values where applicable, confidence indicators, and corrective guidance. Grouping element observations under a single rule entry preserves rule identity while enabling detailed reporting.

Deduplication and Cross-Rule Suppression To prevent redundant reporting, the framework performs two forms of consolidation. First, element-level detections referring to the same structural artifact are grouped using contextual identifiers derived from source location and rule-specific metadata. This prevents multiple sub-checks from producing separate violations for the same underlying issue.

Second, predefined cross-rule suppression relationships prevent derivative findings from appearing alongside their root causes. For example, a missing form label (A5) suppresses accessible-name violations (A6) on the same control, while keyboard inoperability (A3) suppresses focus visibility issues (A2). These suppression relationships are implemented through an explicit rule-priority mapping evaluated during aggregation. The principal relationships applied by the system are summarized in Table 4.4.

Table 4.4: Cross-rule suppression relationships applied during result aggregation.

Dominant Rule	Suppressed Rule	Rationale
A5	A6	Missing form labels prevent computation of accessible names.
A3	A2	Keyboard-inoperable elements cannot meaningfully exhibit focus visibility.
A1	U6	Severe contrast failures may produce apparent grouping or layout confusion.
U4	U1	Recall burden may explain ambiguity in primary actions.
E2	U1	Manipulative choice architecture may bias perceived primary actions.
E2	E1	Manipulation may subsume transparency concerns in high-impact actions.
E2 / E3	Conflict resolution	Visual hiding (E2) and functional restriction (E3) are resolved according to the dominant behavioral constraint.

LLM Result Consolidation Rules evaluated through interpretive reasoning (e.g., U4, U6, E1, and E2) produce summary-level outputs rather than element

arrays. Each rule returns a single structured entry containing classification, confidence, and diagnostic explanation. Because these findings operate at the interface level rather than the element level, no additional deduplication is required.

Methodological Implications The aggregation stage provides rule-centric reporting with optional element-level granularity while ensuring stable convergence alignment. By consolidating detections, assigning epistemically grounded severity classifications, and applying predefined suppression rules, the framework maintains interpretable violation counts and consistent evaluation outputs across analysis cycles.

4.3.5 Reporting Interface and Evaluation Dashboard

This section describes the reporting interface of UI Critic, which presents the structured outputs of the evaluation pipeline through an interactive dashboard for inspection and iterative refinement. While the backend performs rule evaluation and convergence analysis, the frontend visualizes results through rule-level reporting, severity differentiation, and iteration-aware comparison.

The interface follows the epistemic evaluation model introduced in Chapter 3. Findings are organized according to the rule taxonomy and classified as confirmed violations, potential risks, or rules not evaluated, preserving the distinction between deterministic and interpretive results.

Results are displayed through aggregated rule cards and compared across evaluation iterations, enabling systematic tracking of detected violations, corrective actions, and convergence status over time.

The following subsections describe the rule-level reporting structure, the corrective prompting and iteration workflow, and the generation of the final evaluation report.

Rule-Level Reporting Structure

Evaluation results are presented through a rule-centric dashboard architecture that mirrors the analytical model introduced in Chapter 3. Each rule in the taxonomy (A1–A6, U1–U6, E1–E3) is represented by a single reporting card summarizing its evaluation outcome, supporting evidence, and diagnostic explanation. This structure preserves rule-level semantics while allowing detailed inspection of individual findings contributing to each result.

Rule-Centric Card Organization The reporting interface adopts a one-rule–one-card structure, where each rule is displayed as an independent card. The collapsed card view presents the rule identifier, rule title, a short diagnostic description, and

the number of detected elements when applicable. Cards are visually grouped according to their evaluation outcome.

The dashboard distinguishes three possible result states:

- Confirmed Violations (Blocking)
- Potential Risks (Non-blocking)
- Rules Not Evaluated

Confirmed violations correspond to structurally verifiable issues detected through deterministic analysis and may affect convergence status. Potential risks represent findings that require contextual verification or arise from interpretive analysis and therefore do not block convergence.

Rules classified as Not Evaluated appear when the available input artifact does not provide sufficient evidence to assess the rule. This typically occurs when the evaluation modality lacks the structural information required for deterministic analysis (e.g., screenshot-only inputs). These cards explicitly state the reason the rule could not be evaluated and may include advisory guidance describing what additional artifacts (such as source code or interaction recordings) would be required for reliable evaluation. An example of a not-evaluated rule card is shown in Figure 4.4.

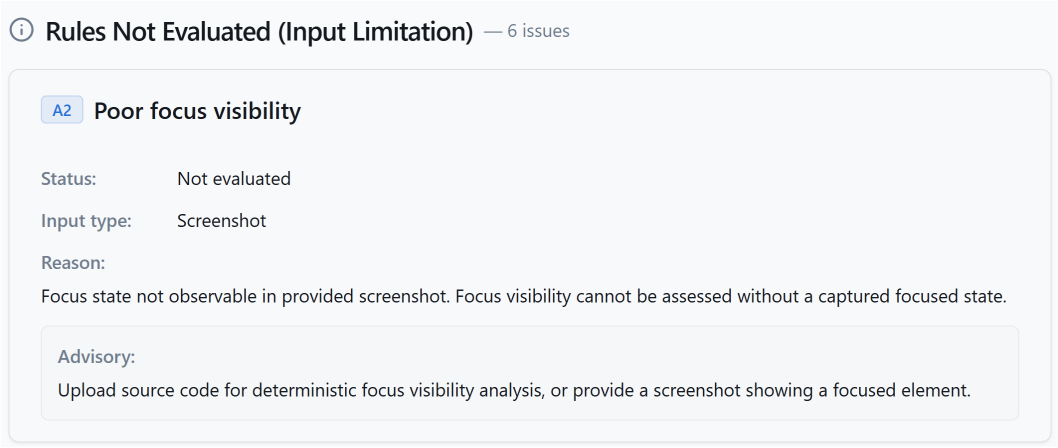

Figure 4.4: Example not-evaluated rule card indicating that the rule cannot be assessed due to input modality limitations.

As illustrated in Figure 4.4, the system marks the A2 (Poor Focus Visibility) rule as Not Evaluated when only screenshot input is available, since focus indicators cannot be assessed without an observable focused state. The card therefore explains the input limitation and suggests providing source code or a screenshot showing the focused element.

Element-Level Findings When a rule is associated with specific interface components, the rule card may contain multiple element-level findings grouped under the same rule entry. These entries can be expanded to reveal detailed inspection information describing the detected condition.

As illustrated in Figure **??**, multiple interface elements violating the same rule are grouped within a single rule card while preserving detailed evidence for each occurrence.

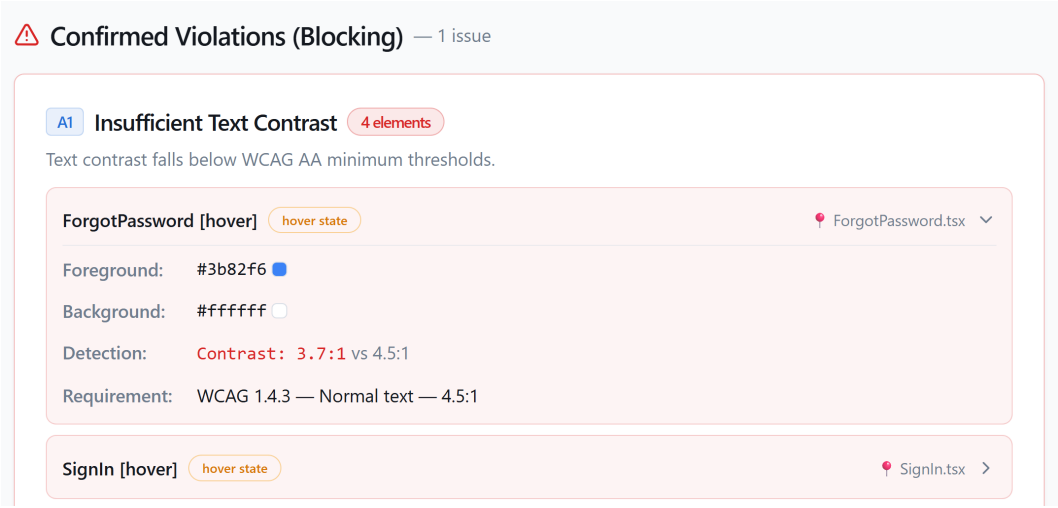

Figure 4.5: Example confirmed violation card showing rule-level aggregation and element-level evidence for A1 (Insufficient Text Contrast).

Each element-level entry may include:

- the detected issue or condition
- supporting diagnostic evidence
- relevant computed values when applicable (e.g., contrast ratios)
- contextual location indicators such as source file paths, line numbers, or screenshot references

Grouping element-level findings under a single rule card prevents fragmentation of rule identity while enabling detailed inspection of individual occurrences contributing to the violation.

Differences Between Confirmed and Potential Results The diagnostic information presented within element entries varies slightly depending on the classification type.

Confirmed violations primarily present structural evidence extracted during deterministic analysis, including element identifiers, detected violations, relevant styling tokens or attributes, and source locations within the analyzed codebase.

Potential findings additionally include a confidence estimate representing the certainty of the assessment and an advisory guidance section describing recommended design improvements or verification steps. These fields communicate that the issue may require contextual review rather than representing a strictly verifiable structural violation.

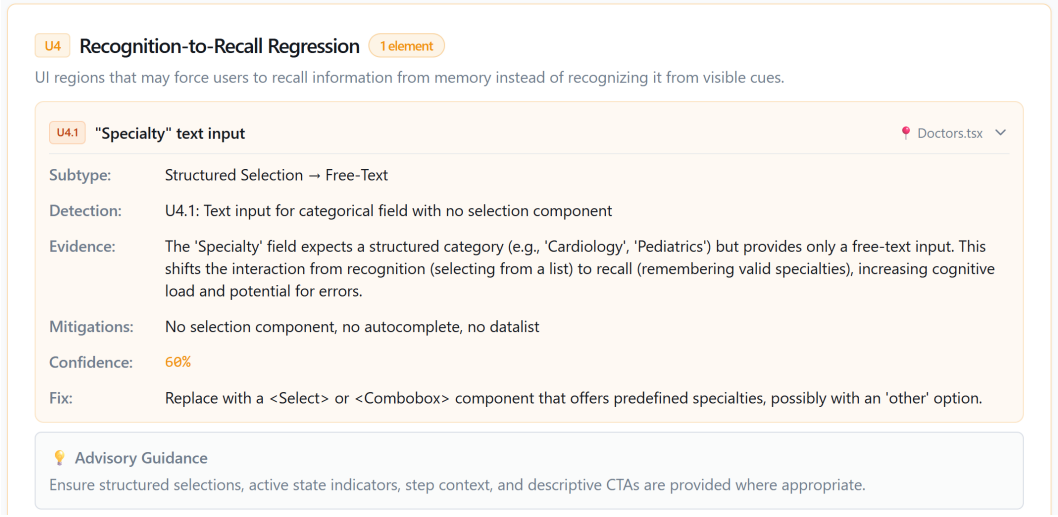

Figure 4.6: Example potential risk card for U4 (Recognition-to-Recall Regression) showing confidence estimation and advisory guidance.

As illustrated in Figure 4.6, the system identifies a potential U4 (Recognition-to-Recall Regression) issue where a free-text input is used for specialty selection, requiring users to recall valid options instead of selecting them from visible alternatives.

Reporting Consistency By structuring results around rule-level cards with nested element observations, the reporting interface maintains clear correspondence with the rule taxonomy while providing granular diagnostic transparency. This structure enables users to quickly understand the overall rule outcome while retaining access to detailed evidence supporting each reported issue.

Corrective Prompting and Iteration Tracking

In addition to reporting detected violations, the framework generates corrective prompts designed to assist iterative refinement of the analyzed interface. These

prompts translate diagnostic findings into actionable instructions that can be provided to generative UI tools or developers to address detected issues.

Corrective prompts are presented in a dedicated section of the reporting interface and are generated only for confirmed violations identified during deterministic evaluation. Since confirmed violations represent structurally verifiable issues, corrective guidance can be produced with a high degree of reliability.

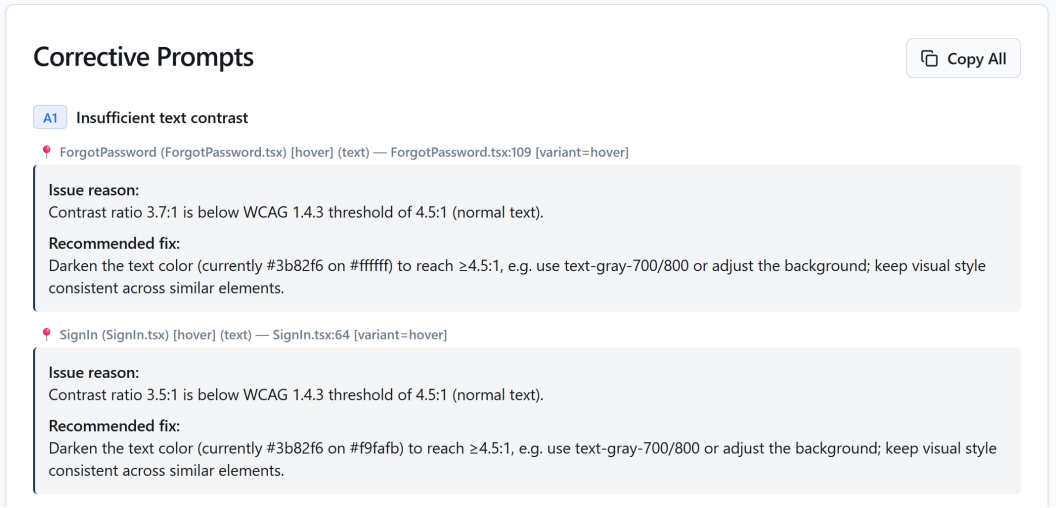

Figure 4.7: Example corrective prompt generated for rule A1 (Insufficient Text Contrast), including the detected issue and recommended fix.

Each corrective prompt corresponds to a specific rule violation and includes two primary components:

Issue reason, describing the detected condition and referencing the relevant guideline or threshold.

Recommended fix, providing concrete instructions for resolving the violation while preserving interface consistency.

When multiple elements violate the same rule, the system generates element-specific prompt entries under the corresponding rule heading. Each entry includes contextual identifiers such as the affected component name, source file location, and relevant styling or attribute information. This structure allows developers or generative systems to locate the affected element and apply targeted modifications.

Figure 4.7 illustrates an example corrective prompt generated for rule A1 (Insufficient Text Contrast), where the system identifies text elements whose contrast ratio falls below the WCAG 1.4.3 threshold and recommends adjusting the foreground color to satisfy accessibility requirements.

The corrective prompt system supports the iterative evaluation workflow of the framework. After applying the suggested modifications, the updated interface can

be re-evaluated through a new analysis cycle. Each cycle produces a new evaluation report, enabling comparison between iterations and tracking whether previously identified violations have been resolved.

Iteration tracking is implemented through the dashboard interface, which records the number of confirmed violations, potential risks, and the configured convergence threshold for each evaluation cycle. Confirmed violations represent structurally verifiable issues and directly affect convergence status, while potential risks correspond to advisory findings that do not block convergence.

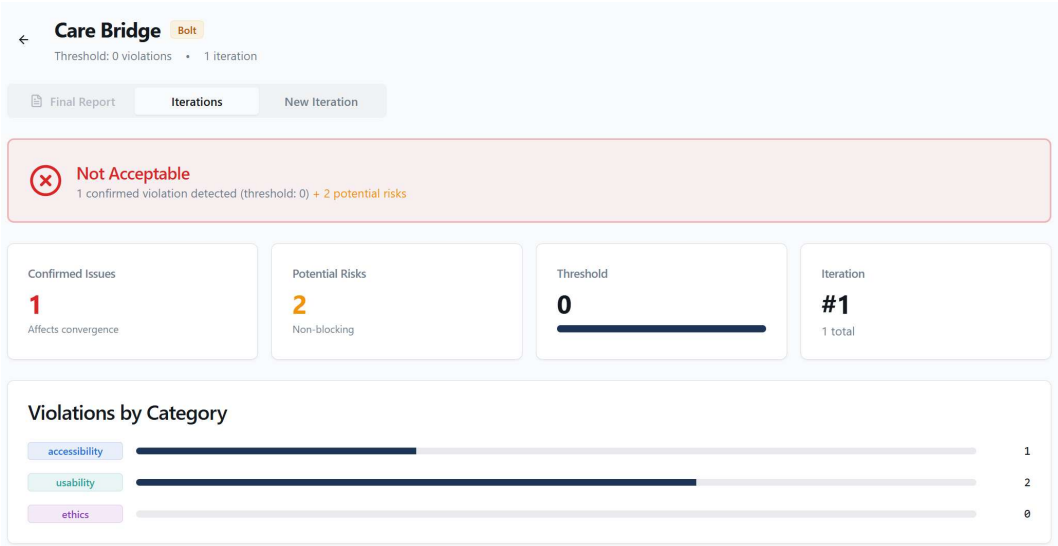

Figure 4.8: First evaluation iteration for the Clinic Bridge interface generated by Bolt. The system reports one confirmed accessibility violation (A1 – Insufficient Text Contrast) and two potential usability risks.

To illustrate this process, the Clinic Bridge appointment management interface generated using the Bolt platform was evaluated across two iterations. In the first iteration, the system analyzes the ZIP source input and reports one confirmed accessibility violation corresponding to A1 (Insufficient Text Contrast) affecting four interactive elements in hover state, including the ForgotPassword, SignIn, and SignUp components.

In addition to this confirmed violation, the evaluation identifies two potential usability risks: U5 (Insufficient Interaction Feedback) related to a possible lack of loading or confirmation states for the handleSignOut action in AdminLayout.tsx, and U4 (Recognition-to-Recall Regression) where the Specialty field in Doctors.tsx is implemented as a free-text input rather than a structured selection component.

Because a confirmed violation is present, the interface is classified as Not Acceptable, and corrective prompts are generated to address the detected contrast

issue. Figure 4.8 presents the dashboard view for this first evaluation iteration.

After applying the recommended corrections generated by the corrective prompt system, the updated interface is re-evaluated in a second iteration. In this subsequent evaluation, the previously detected A1 violation is successfully resolved, reducing the number of confirmed violations to zero. The two potential usability risks remain unchanged, as expected, since they represent advisory findings rather than deterministic violations.

Since the framework threshold is configured as 0 confirmed violations, the system reports that convergence has been achieved at iteration #2. Figure 4.9 illustrates the dashboard view for the second evaluation cycle.

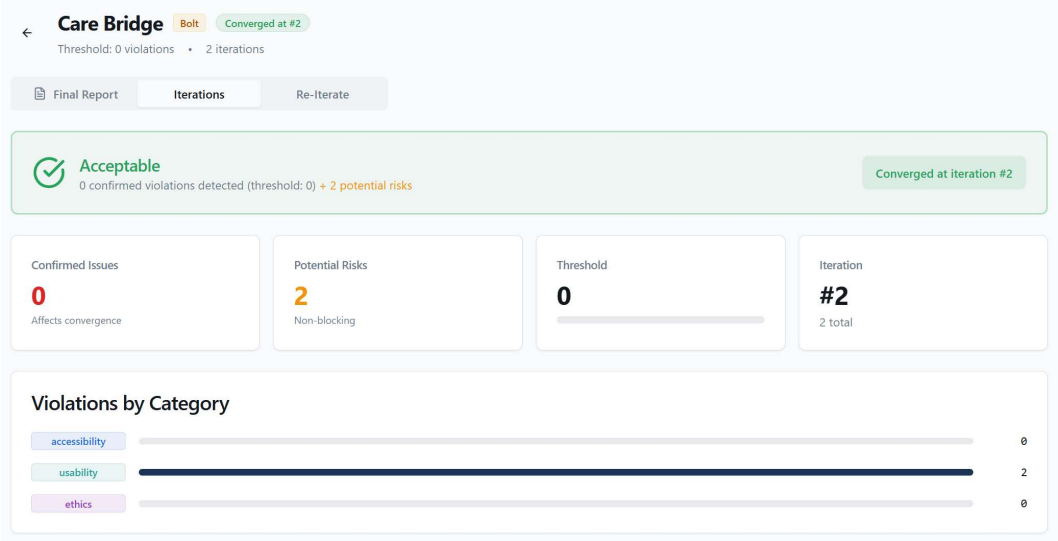

Figure 4.9: Second evaluation iteration after applying corrective prompts. The previously detected A1 violation is resolved and the system reports convergence at iteration #2.

This example demonstrates how the framework supports guided iterative refinement: deterministic violations are detected, corrective prompts are generated, and the interface is re-evaluated until convergence criteria are satisfied.

Final Report Generation

After convergence is achieved, the framework generates a final evaluation report summarizing the complete analysis process and the resulting interface quality status. The report consolidates the diagnostic outcomes from all evaluation iterations and provides a structured overview of rule compliance, remaining advisory findings, and category-level improvements.

The final report is designed to support reproducibility, documentation of evaluation results, and comparison across generated interfaces. It contains a summary of convergence status, improvement metrics across rule categories, and a snapshot of the final rule evaluation state.

Figure 4.10 illustrates the Final Analysis Summary section of the generated report. This section indicates that the Clinic Bridge interface generated using the Bolt platform reached convergence after two evaluation iterations. During this process, the total number of confirmed violations decreased from one to zero, representing a 100% improvement in detected blocking issues.

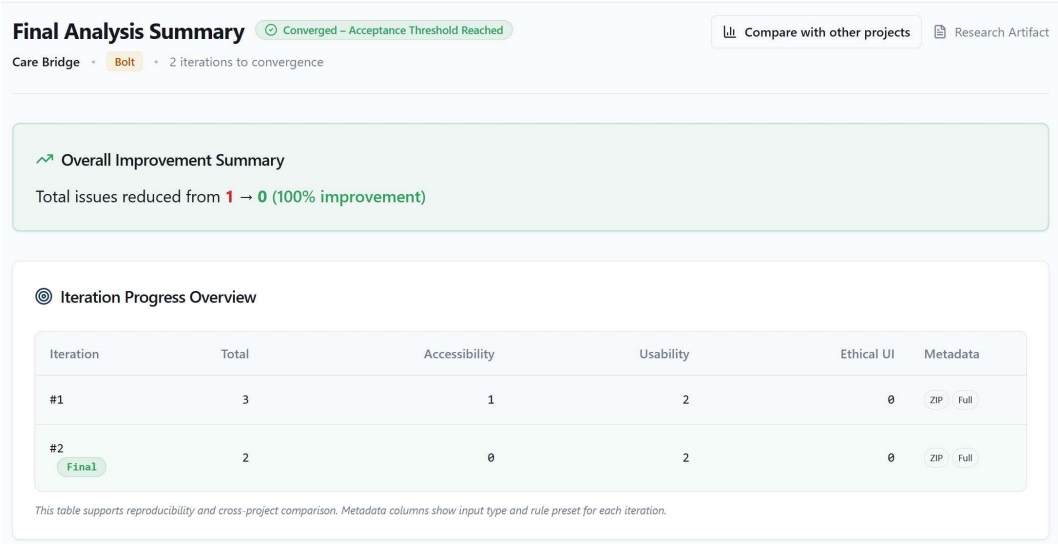

Figure 4.10: Final analysis summary showing convergence status and overall improvement across iterations.

To support transparency of the iterative process, the report provides an Iteration Progress Overview table summarizing the results of each evaluation cycle. This table records the number of detected issues for each rule category, along with metadata describing the evaluation configuration. In the example shown in Figure 4.10, the first iteration detected three issues in total (one accessibility violation and two usability risks), while the second iteration resolved the accessibility issue and retained the two usability advisory findings.

The report also includes a Category-Level Improvement analysis, which presents the change in violation counts across the three rule groups defined in the framework: accessibility, usability, and ethical interface design. As shown in Figure 4.11, the accessibility category improved from one detected violation in the initial iteration to zero violations in the final iteration. In contrast, the usability category retained two advisory findings that remained unchanged across iterations. Ethical interface

rules did not produce any violations during the evaluation.

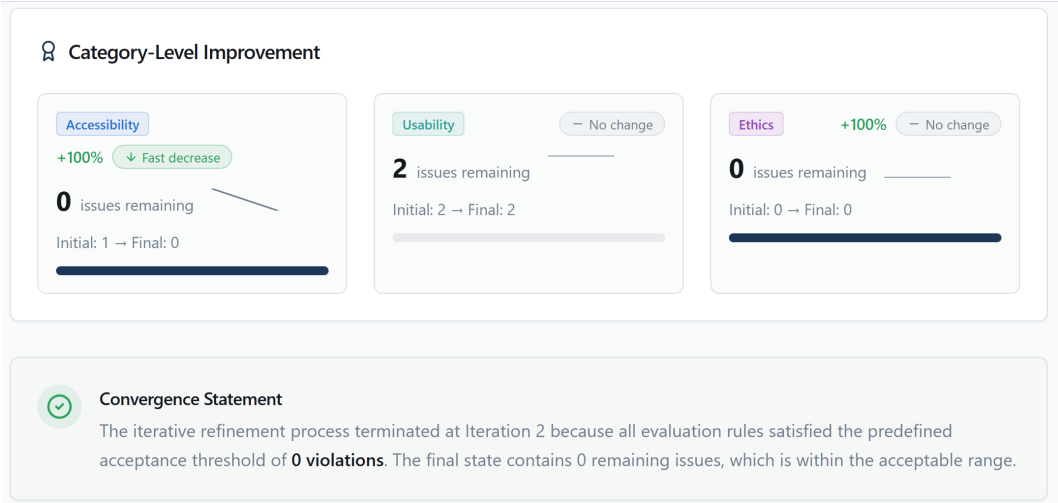

Figure 4.11: Category-level improvement analysis comparing initial and final iteration results.

Once the convergence condition is satisfied, the system produces a Convergence Statement confirming that the final interface satisfies the predefined acceptance threshold. In this case, the framework was configured with a threshold of zero confirmed violations, meaning that all blocking issues must be resolved before convergence is reached. The report explicitly states that the iterative refinement process terminated at iteration two because all evaluation rules satisfied the acceptance threshold.

Finally, the report presents a Final Rule Status Snapshot, which lists all evaluation rules and indicates their final status. As shown in Figure 4.12, the final interface passes all accessibility, usability, and ethical design rules that require deterministic verification. Two usability rules—U4 (Recognition-to-Recall Regression) and U5 (Insufficient Interaction Feedback)—remain as low-severity advisory findings. These findings do not prevent convergence because they require contextual interpretation rather than deterministic structural verification.

The generated report therefore serves as a structured artifact documenting the final evaluation outcome, including rule compliance, iteration history, and convergence justification. This artifact can be exported as a PDF to support reproducibility of the evaluation process and comparison across multiple generated interface projects.

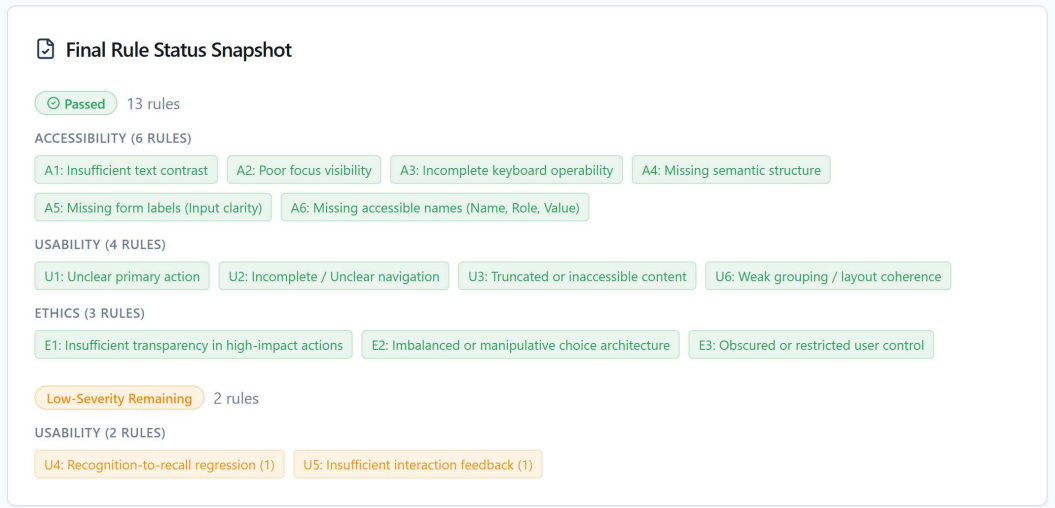

Figure 4.12: Final rule status snapshot showing passed rules and remaining advisory findings.

4.4 Summary

This chapter presented the implementation of the UI Critic proof-of-concept system and described how the proposed evaluation framework is operationalized within a working analysis pipeline. The chapter detailed the ingestion and parsing mechanisms used to process different interface artifacts, the rule-based detection components used to identify accessibility, usability, and ethical design issues, and the reporting interface that communicates diagnostic results through structured rule cards. The corrective prompting mechanism and iteration tracking workflow were also introduced, demonstrating how detected violations can guide iterative refinement of generated interfaces until a predefined acceptance threshold is reached. Finally, the chapter described the structure of the generated final evaluation report, which summarizes rule compliance, improvement across iterations, and convergence outcomes. Together, these components demonstrate how the proposed framework can be implemented as a practical tool for systematic evaluation and iterative improvement of AI-generated user interfaces.

Chapter 5

Experimental Results and Evaluation

This chapter presents the experimental evaluation of the UI Critic framework. The purpose of the evaluation is to assess the effectiveness, robustness, and practical applicability of the proposed approach for analyzing modern web interfaces. To investigate these aspects, a series of experiments were conducted across multiple evaluation scenarios, including AI-generated applications, human-developed projects, different input modalities, and repeated LLM-assisted analyses. These experiments aim to evaluate the framework's ability to detect accessibility, usability, and ethical design issues under varying conditions, while also examining the stability and reliability of its rule-based and LLM-assisted evaluation components.

5.1 Cross-Tool Comparison

This experiment compares the behavior of three generative development platforms, Bolt, Lovable, and Replit, when generating the same web application under identical conditions. The objective is to evaluate differences in the accessibility, usability, and ethical design compliance of the generated interfaces.

Beyond assessing the quality of the initial outputs, the study also examines how each platform responds to iterative corrective prompts derived from detected violations. By applying these prompts until convergence is reached, the analysis evaluates violation reduction and convergence behavior across platforms.

5.1.1 Experimental Setup

The experiment was conducted using three generative development platforms: Bolt, Lovable, and Replit. A single standardized prompt describing the target application

was submitted independently to each platform, producing three interface implementations. The generated code was not manually modified; any improvements during the experiment were performed exclusively through prompt-based interactions.

The generated system, CareBridge, is a full-stack clinic appointment booking and patient portal application with an administrative interface. The specification includes authentication, role-based access (patient and administrator), structured navigation, form-based interactions, and administrative dashboards. These features were selected to ensure the presence of diverse interface components relevant to accessibility, usability, and ethical design evaluation.

To ensure cross-platform comparability, the standardized prompt defined the application architecture, required user roles, core pages, and interaction flows. It also specified accessibility and usability requirements, including proper heading hierarchy, keyboard-accessible controls with visible focus indicators, labeled form inputs, accessible names for icon-only buttons, inline validation feedback, and loading indicators for asynchronous actions.

High-impact interaction flows were also specified, including confirmation dialogs for destructive actions, explicit disclosure for account deletion, and non-preselected marketing consent options. These constraints ensured that the generated interfaces contained interaction patterns suitable for evaluating ethical design considerations.

For evaluation, the ZIP export of each generated project was used as the input artifact. This modality enables static inspection of structural accessibility features such as semantic markup, form labeling, and accessible names while ensuring consistent comparison across platforms.

5.1.2 Evaluation Procedure

The generated applications were evaluated using the UI Critic framework, which analyzes interfaces according to the rule-based evaluation model described in Chapter 3.

The evaluation was conducted in two stages.

Initial Generation. Each platform generated the application using the standardized prompt. The resulting project was exported as a ZIP archive containing the source code, which served as the input artifact for analysis.

Iterative Refinement. After the initial analysis, corrective prompts derived from confirmed violations were applied to guide interface improvements. The updated applications were then re-analyzed using the same procedure.

This process continued until convergence, defined as the point at which no confirmed violations remained.

All fifteen evaluation rules were enabled during analysis, organized into three categories: Accessibility (A1–A6), Usability (U1–U6), and Ethical Design (E1–E3). The violation acceptance threshold was set to zero confirmed violations.

5.1.3 Evaluation Metrics

Several metrics were collected to compare platform behavior.

Violation Counts. The total number of detected violations was recorded for each platform, along with their distribution across accessibility, usability, and ethical design categories.

Severity Distribution. Detected issues were classified according to their evaluation status, distinguishing confirmed violations from potential risks.

Convergence Iterations. The number of refinement iterations required for each platform to reach convergence was recorded.

Stability Across Iterations. Changes in violation counts across iterations were analyzed to examine how interface quality evolved during iterative refinement.

5.1.4 Experimental Results

This section presents the results obtained from evaluating the applications generated by Bolt, Lovable, and Replit using the UI Critic framework. The analysis summarizes violations detected during the initial generation, their evolution during refinement, and the final state reached after convergence.

Confirmed violations were used as the primary quantitative measure for cross-platform comparison. Potential risks were analyzed separately to provide additional qualitative insight into interface behavior.

Initial Confirmed Violations

Table 5.1 summarizes the confirmed violations detected during the first evaluation iteration.

Table 5.1: Distribution of confirmed violations detected during the first evaluation iteration for each platform.

Platform	Accessibility	Usability	Ethics	Total
Bolt	1	0	0	1
Lovable	1	1	0	2
Replit	1	1	0	2

Lovable and Replit produced a higher number of confirmed violations during the initial generation, while Bolt generated the lowest number. Most confirmed violations were related to accessibility rules, although usability issues were also observed.

The specific violations detected were:

- A1 - Insufficient Text Contrast (Bolt)
- A2 - Poor Focus Visibility (Lovable, Replit)
- U3 - Truncated or Inaccessible Content (Lovable, Replit)

Lovable and Replit produced the same pair of violations, indicating similar interface generation behavior in terms of styling and layout decisions. In contrast, Bolt generated a different accessibility issue and did not produce the usability issue observed in the other platforms.

These violations largely originate from default interface design decisions, including insufficient color contrast in hover states, weak focus indicators for interactive elements, and layout constraints that cause content truncation.

Initial Potential Risks

In addition to confirmed violations, the evaluation framework identified several potential risks during the first iteration.

Table 5.2 summarizes the potential risks detected during the first iteration by rule category.

Table 5.2: Distribution of potential risks detected during the first evaluation iteration by rule category.

Platform	Accessibility	Usability	Ethics	Total
Bolt	0	2	0	2
Lovable	2	1	1	4
Replit	3	0	1	4

Most potential risks were related to accessibility conditions that could not be fully verified through static analysis. The most frequent potential issues included:

- A1 - Insufficient Text Contrast
- A2 - Poor Focus Visibility
- A5 - Missing Form Labels
- U5 - Insufficient Interaction Feedback
- E1 - Insufficient Transparency in High-Impact Actions

Lovable and Replit exhibited similar risk distributions dominated by accessibility-related uncertainties, while Bolt produced a different profile consisting primarily of usability-related concerns. In particular, Bolt generated potential risks related to U4 (Recognition-to-Recall Regression) and U5 (Insufficient Interaction Feedback).

Iterative Violation Reduction and Convergence

After the initial evaluation, corrective prompts were applied iteratively until convergence was reached. Table 5.3 presents the reduction of confirmed violations observed during the refinement process.

Table 5.3: Reduction of confirmed violations after iterative refinement.

Platform	Initial	Final	Reduction
Bolt	1	0	1
Lovable	2	0	2
Replit	2	0	2

All platforms successfully resolved their confirmed violations during the iterative refinement process, reaching the predefined acceptance threshold of zero confirmed violations.

Table 5.4 presents the number of iterations required for convergence.

Table 5.4: Number of iterations required for each platform to reach convergence.

Platform	Iterations to Convergence
Bolt	2
Lovable	2
Replit	2

Although the number of initial violations differed across platforms, the refinement process consistently achieved convergence within two iterations. This indicates that the corrective prompts generated by the evaluation framework were effective in guiding the platforms toward acceptable interface states.

5.1.5 Cross-Tool Comparative Analysis

The comparative analysis highlights several differences in the behavior of the evaluated generative development platforms.

First, the platforms produced different numbers of confirmed violations during the initial generation. Bolt generated the fewest confirmed violations, while Lovable and Replit produced more issues.

Second, Lovable and Replit demonstrated similar interface generation patterns, producing the same confirmed violations, A2 (Poor Focus Visibility) and U3 (Truncated or Inaccessible Content), during the first iteration. This suggests comparable styling and layout decisions across the two systems.

Bolt, in contrast, produced a different violation (A1 - Insufficient Text Contrast) and did not exhibit the usability issue observed in the other platforms.

Differences were also observed in the distribution of potential risks. Lovable and Replit produced several accessibility-related risks, particularly related to text contrast (A1) and focus visibility (A2), and both platforms also generated potential risks related to ethical transparency (E1). Replit additionally produced a risk related to A5 (Missing Form Labels).

Bolt's potential risks were instead concentrated in usability concerns, specifically U4 (Recognition-to-Recall Regression) and U5 (Insufficient Interaction Feedback).

Despite these differences, all platforms converged successfully after two refinement iterations. This demonstrates that the corrective prompts generated by the UI Critic framework were effective in guiding each platform toward an acceptable interface state.

Overall, the results indicate that generative development platforms can produce different patterns of accessibility, usability, and ethical design issues even under identical prompt conditions. At the same time, the iterative refinement process supported by the UI Critic framework proved effective in resolving confirmed violations across all evaluated platforms.

5.2 Modality Consistency Test

This experiment evaluates the consistency of the UI Critic framework when analyzing the same interface through different input modalities. While the previous experiment compared the behavior of generative development platforms, this experiment focuses on the reliability of the evaluation pipeline itself. Specifically, it investigates whether similar violations are detected when the same application is analyzed using different artifact representations. By keeping the interface implementation constant and varying only the input modality, the experiment assesses the stability of the framework's evaluation results.

5.2.1 Experimental Setup

The experiment used the CareBridge application generated by the Bolt platform during Experiment 1. The original version produced during the first generation

stage was analyzed, prior to any corrective refinement. Using the initial version preserves the natural violation profile of the generated interface and allows the evaluation to focus exclusively on the effect of the input modality.

The evaluation dataset therefore consisted of a single interface implementation analyzed across three input modalities: ZIP archive analysis, GitHub repository analysis, and screenshot-based visual analysis.

The evaluation configuration remained identical to the previous experiment. All 15 evaluation rules were enabled, covering three rule categories: Accessibility (A1–A6), Usability (U1–U6), and Ethical Design (E1–E3). The violation acceptance threshold was set to zero confirmed violations.

Unlike the iterative evaluation process used in Experiment 1, no corrective prompts or refinement iterations were performed. The objective of this experiment is not to improve the interface but to compare the violations detected when the same application is analyzed using different artifact representations.

5.2.2 Evaluation Procedure

The UI Critic framework supports three input modalities for interface evaluation: ZIP archive analysis, GitHub repository analysis, and screenshot- based visual analysis.

The evaluation procedure consisted of analyzing the same application using each modality independently. First, the project was exported as a ZIP archive and evaluated using the ZIP input modality. Second, the application's public GitHub repository was analyzed using the repository modality. Finally, a set of interface screenshots was uploaded and analyzed using the screenshot modality.

For the screenshot evaluation, 30 screenshots were captured covering the main interface pages and interaction states of the application, including dashboards, forms, tables, navigation menus, and modal dialogs. This ensured comprehensive visual coverage of the interface.

All evaluations were executed using the same rule configuration and threshold settings. Because screenshots do not expose the underlying programmatic structure of the interface, certain rules that rely on source-code inspection may be classified as *Not Evaluated* when using this modality.

To assess modality consistency, the results were compared across modalities by examining rule-level detections and identifying violations that were detected consistently across representations or were specific to particular modalities.

5.2.3 Experimental Results

This section presents the results obtained from evaluating the CareBridge application generated by the Bolt platform using different input modalities of the UI Critic

framework. The purpose of this experiment is to examine whether the evaluation results remain consistent when the same application is analyzed through different artifact representations.

Modality Comparison Between ZIP and GitHub

Table 5.5 summarizes the violations detected when the CareBridge application was analyzed using the ZIP and GitHub input modalities.

Table 5.5: Comparison of violations detected by ZIP and GitHub input modalities.

Modality	Confirmed Violations	Potential Risks
ZIP	A1	U5, U4
GitHub	A4	U4, A1

Ideally, identical results would be expected because both modalities analyze the same underlying source code. However, the evaluation results reveal differences in both the type and classification of detected violations.

The ZIP-based analysis reported a confirmed accessibility violation corresponding to A1 (Insufficient Text Contrast) affecting several interactive elements in hover state. In addition to this confirmed violation, the ZIP analysis detected two potential usability risks: U5 (Insufficient Interaction Feedback) related to the handleSignOut action in AdminLayout.tsx, and U4 (Recognition-to-Recall Regression) associated with the Specialty input field in Doctors.tsx.

In contrast, the GitHub-based evaluation detected a confirmed violation corresponding to A4 (Missing Semantic Structure) involving multiple page headings and dashboard elements lacking semantic landmarks. The GitHub analysis also reported two potential risks: U4 (Recognition-to-Recall Regression) affecting the same specialty input field detected in the ZIP analysis, and A1 (Insufficient Text Contrast). However, the contrast issue could not be fully verified during static analysis and was therefore classified as a potential risk rather than a confirmed violation.

These results indicate that artifact ingestion differences can influence structural detection outcomes. The ZIP modality analyzes a complete exported snapshot of the project, allowing deterministic evaluation of styling properties such as text contrast. Repository-based analysis, however, may reconstruct the project structure differently, which can affect rule verification.

Despite these differences, both modalities consistently identified the U4 (Recognition-to-Recall Regression) issue associated with the Specialty input field, suggesting that certain usability problems remain detectable across different artifact representations.

Modality Comparison Between Screenshot and Code-Based Inputs

The CareBridge application was also evaluated using the screenshot input modality, which analyzes rendered interface images rather than source code.

The screenshot-based evaluation produced no confirmed violations and identified several potential risks, including:

- U3 - Truncated or Inaccessible Content
- A1 - Insufficient Text Contrast
- U4 - Recognition-to-Recall Regression

The U3 detection identified several interface regions where textual content may appear visually truncated, including elements within the Review page, Appointment card, and Admin Appointments table. This issue was not detected in the code-based analyses because truncation often results from rendered layout constraints such as container width or overflow settings, which may not be directly inferable from static source code inspection.

The screenshot analysis also indicated a potential A1 (Insufficient Text Contrast) issue. Because exact contrast ratios cannot be reliably computed from static images without complete styling information, the issue was classified as a potential risk.

In addition, the screenshot modality detected U4 (Recognition-to-Recall Regression) related to the same Specialty input field previously identified in the code-based analyses, indicating that some usability issues remain observable across both structural and visual representations of the interface.

Several rules were classified as Not Evaluated due to the absence of structural or runtime information in screenshots, including:

- A2 - Focus Visibility
- A3 - Keyboard Operability
- A4 - Semantic Structure
- A5 - Missing Form Labels
- A6 - Missing Accessible Names
- U5 - Interaction Feedback

These rules require programmatic inspection or runtime interaction and therefore cannot be reliably assessed using static screenshots.

5.2.4 Modality Consistency Analysis

The modality experiment demonstrates how different artifact representations influence the types of issues that can be detected by the UI Critic framework.

The comparison between the ZIP and GitHub modalities revealed differences in confirmed violations despite analyzing the same source code. The ZIP-based evaluation confirmed A1 (Insufficient Text Contrast), while the GitHub-based evaluation instead detected A4 (Missing Semantic Structure) and classified the contrast issue only as a potential risk. This difference indicates that repository-based ingestion may introduce slight variations in the reconstructed project snapshot used for static analysis.

The screenshot modality produced a different evaluation profile because it analyzes rendered interface images rather than source code. As a result, several structural accessibility rules could not be evaluated. However, screenshot analysis successfully detected U3 (Truncated or Inaccessible Content), a visually observable issue that was not identified in the code-based analyses.

At the same time, the experiment revealed consistent findings across modalities. In particular, U4 (Recognition-to-Recall Regression) related to the Specialty input field was detected in the ZIP, GitHub, and screenshot analyses, indicating that certain usability issues remain detectable across different artifact representations.

Table 5.6 summarizes the main strengths and limitations of the three modalities.

Table 5.6: Strengths and limitations of UI Critic evaluation modalities.

Modality	**Strength**	**Limitation**
ZIP	Most reliable structural analysis	May miss visually rendered layout issues
GitHub	Similar to ZIP but ingestion variability	Some classification differences
Screenshot	Visual usability insights	Cannot evaluate structural accessibility rules

Overall, the results demonstrate that the different input modalities provide complementary perspectives on interface quality. Code-based analysis enables reliable detection of structural accessibility violations and programmatic issues, while screenshot-based analysis can reveal perceptual layout problems that may not be directly observable in source code. Among the evaluated modalities, the ZIP input provides the most stable basis for deterministic rule evaluation, while screenshot analysis serves as a complementary method for identifying visually observable usability concerns.

5.3 Real-World GitHub Evaluation

This experiment evaluates the applicability of the UI Critic framework on real-world web applications and investigates whether different development approaches produce different patterns of accessibility, usability, and ethical design violations. Unlike the previous experiments, which focused on controlled application generation and modality consistency, this experiment analyzes complete software projects obtained from public GitHub repositories.

In particular, the experiment compares AI-generated applications produced using generative development tools with manually developed applications created by human developers. This comparison provides insight into how different development workflows influence interface quality and assesses whether the evaluation framework can effectively identify design issues in realistic development environments.

5.3.1 Project Selection

The evaluation dataset consisted of four web applications obtained from GitHub repositories: two AI-generated projects and two human-developed systems. All selected applications represent typical web interfaces containing multiple pages, forms, navigation components, dashboards, and interactive elements, allowing the framework to evaluate a wide range of accessibility, usability, and ethical design rules.

Two applications were generated using AI-assisted development platforms during earlier experiments in this study. These include the CareBridge application generated using the Bolt platform and the same application generated using Replit. Both versions provide interfaces for managing appointments and patient information through dashboards, tables, and form-based interactions.

To represent human-developed applications, two additional projects were included. The first is the *Thesis Management System*, a web application developed as part of a university software engineering course project at Politecnico di Torino. The system supports the management of thesis proposals and student applications and includes multiple user roles, proposal management interfaces, and administrative views.

The second human-developed project is *Material Kit React*, an open-source admin dashboard template available on GitHub. The project contains multiple interface pages, including dashboards, authentication views, account settings, and management interfaces that incorporate common UI components such as tables, forms, and navigation menus.

The two human-developed systems were intentionally selected to represent different levels of development experience. The Thesis Management System reflects a student-developed application produced within an academic course environment,

whereas Material Kit React represents a professionally maintained open-source interface developed by experienced contributors.

The evaluated projects are summarized in Table 5.7.

Table 5.7: Overview of the web applications evaluated in the real-world GitHub experiment.

Project	Type	Source	Description
CareBridge (Bolt)	AI-generated	Generated with Bolt	Healthcare appointment system
CareBridge (Replit)	AI-generated	Generated with Replit	Healthcare appointment system
Thesis Management	Human-developed	University project	Thesis management platform
Material Kit React	Human-developed	Open-source	Admin dashboard template

5.3.2 Evaluation Procedure and Metrics

Each project was evaluated using the ZIP input modality of the UI Critic framework. The complete source code of each repository was downloaded and analyzed without modifying the original implementation.

The evaluation applied the full rule set defined by the framework, consisting of 15 rules organized into three categories: Accessibility (A1–A6), Usability (U1–U6), and Ethical Design (E1–E3). Only the initial evaluation results were considered in this experiment; no corrective prompts or refinement iterations were applied. This allows the analysis to focus on the interface quality of the original implementations.

During evaluation, the framework analyzed the source code using the deterministic and LLM-assisted detection mechanisms described in Chapter 4. The resulting reports were used to extract quantitative metrics for comparison.

Two primary metrics were collected:

- **Confirmed and potential violations per project.**
- **Distribution of violations by rule category**, examining how issues are distributed across accessibility, usability, and ethical design dimensions.

5.3.3 Experimental Results

Figure 5.1 presents the number of confirmed violations and potential risks detected across the evaluated applications. Among the analyzed systems, the Thesis Management application exhibited the highest number of confirmed violations, with three confirmed issues identified. These violations were primarily associated with accessibility and usability problems, including truncated interface content, incomplete keyboard operability, and missing form labels.

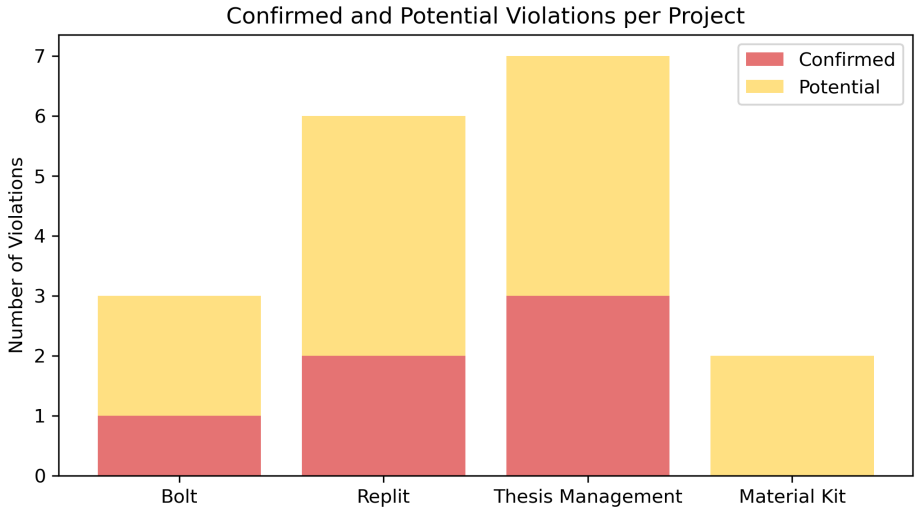

Figure 5.1: Number of confirmed violations and potential risks detected across the evaluated applications.

The Replit-generated CareBridge application produced two confirmed violations, mainly related to focus visibility and truncated interface content. The Bolt-generated version resulted in one confirmed violation, concerning insufficient text contrast. Although both applications were generated from the same specification, the resulting implementations exhibited different accessibility outcomes.

In contrast, the Material Kit React project did not produce any confirmed violations during the evaluation. However, two potential usability risks were identified, mainly associated with insufficient interaction feedback and weak grouping of interface elements.

Figure 5.2 illustrates the distribution of detected violations across rule categories. Accessibility-related issues represent the largest proportion of detected problems across the evaluated systems. These include insufficient text contrast, missing form labels, and incomplete keyboard operability. Usability-related issues were also observed, particularly those related to truncated interface content and insufficient interaction feedback. Ethical design violations were comparatively rare.

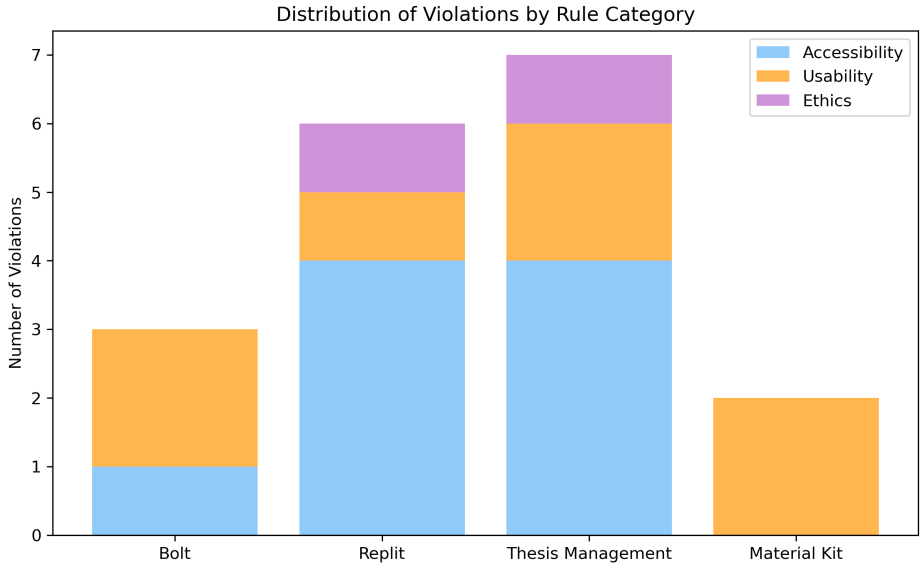

Figure 5.2: Distribution of violations by category across the evaluated applications.

5.3.4 Real-World GitHub Evaluation Analysis

When the results are examined in relation to development context, a clear pattern emerges regarding the influence of developer experience on interface quality. The Thesis Management system, a student-developed application created within a university software engineering course, exhibited the highest number of confirmed violations. This suggests that applications developed by less experienced developers may be more prone to accessibility and usability issues.

The AI-generated applications produced by Bolt and Replit fall between the two human-developed systems. Although both contained fewer violations than the student-developed project, they still exhibited several accessibility and usability issues. The difference between Bolt and Replit suggests that generative platforms may produce varying interface quality even when given the same specification.

In contrast, the Material Kit React project, a professionally maintained open-source interface developed by experienced contributors, produced the lowest number of violations and no confirmed issues. This suggests that interfaces developed by experienced teams tend to demonstrate stronger adherence to accessibility and usability practices.

Overall, the results indicate that the number and type of interface violations are influenced by the level of development experience involved in the implementation. The findings also demonstrate that the UI Critic framework can effectively identify accessibility, usability, and ethical design issues across both AI-generated and human-developed web applications.

5.4 AI-as-a-Judge Sensitivity Analysis

This experiment evaluates the stability of the LLM-assisted components of the UI Critic framework. While several rules rely on deterministic static analysis, others depend on large language model reasoning to assess perceptual and contextual interface properties. Because LLM outputs may exhibit probabilistic behavior, repeated evaluations are required to verify whether consistent results are produced for the same interface.

Two evaluation contexts are considered. In the screenshot modality, all findings originate from vision-based LLM analysis because screenshots do not expose the underlying programmatic structure of the interface. Consequently, deterministic analysis cannot be applied and all detections rely entirely on model interpretation.

In contrast, source-based evaluation combines deterministic preprocessing with LLM-assisted reasoning. Deterministic extractors first analyze the source code and produce structured evidence describing interface characteristics such as layout structure, textual cues, and interaction patterns. This evidence is then used by the LLM for contextual interpretation. Rules primarily relying on this reasoning include U4 (Recognition–Recall Balance), U6 (Visual Grouping and Layout Coherence), E1 (Insufficient Transparency in High-Impact Actions), and E2 (Manipulative Choice Architecture), as described in Chapter 4.

The objective of this experiment is twofold: to evaluate the stability of purely LLM-based screenshot analysis across repeated runs and to examine whether LLM-assisted evaluation grounded in deterministic source-code evidence exhibits similar stability. Comparing these conditions provides insight into how deterministic preprocessing influences the reliability of LLM-assisted evaluation.

5.4.1 Experimental Setup and Procedure

The experiment used the CareBridge application generated with the Bolt platform and considered two complementary evaluation conditions.

The first condition used the screenshot input modality. The evaluation dataset consisted of 30 screenshots capturing the main interface pages and interaction states, including dashboards, forms, modal dialogs, and administrative views. All 15 rules of the UI Critic framework were enabled, covering Accessibility (A1–A6), Usability (U1–U6), and Ethical Design (E1–E3).

To assess stability, the evaluation was executed three times using the same screenshot dataset and identical configuration settings, without corrective iterations. The dataset, rule configuration, evaluation parameters, and screenshot ordering were kept identical across runs to ensure that any differences could be attributed solely to LLM reasoning.

A complementary stability test was conducted using the ZIP source code of the

same CareBridge application. In this condition, only the rules primarily relying on LLM-assisted reasoning were enabled: U4 (Recognition–Recall Balance), U6 (Visual Grouping and Layout Coherence), E1 (Insufficient Transparency in High-Impact Actions), and E2 (Manipulative Choice Architecture). This setup isolates the behavior of LLM-assisted rules when grounded in deterministic structural evidence extracted from the source code.

The ZIP-based evaluation was also executed three times using identical input artifacts and configuration settings. Across both conditions, the confirmed violations and potential risks detected in each run were compared to identify any variability in LLM-assisted rule detection.

5.4.2 Experimental Results and Analysis

Across the repeated runs, the framework produced highly similar results with only minor variations in the number of detected potential risks. As expected, no confirmed violations were reported in any run. This behavior is consistent with the framework's severity model, where LLM-assisted findings are always classified as *Potential* rather than *Confirmed*. This design choice prevents probabilistic model interpretations from producing blocking violations and ensures that convergence decisions remain based exclusively on deterministic structural evidence.

Screenshot-Based Evaluation

Table 5.8 summarizes the rules detected across the three screenshot-based runs. Two rules were identified during the repeated evaluations: A1 (Insufficient Text Contrast) and U6 (Weak Grouping / Layout Coherence). However, their detection frequencies differed across runs.

Table 5.8: Consistency of detected violations across repeated screenshot-based evaluation runs in the AI-as-a-Judge sensitivity analysis.

Rule	Run 1	Run 2	Run 3	Frequency
A1 - Insufficient Text Contrast	✓	–	✓	2 / 3
U6 - Weak Grouping / Layout Coherence	✓	✓	✓	3 / 3

The first run identified two potential risks: A1 (Insufficient Text Contrast) and U6 (Weak Grouping / Layout Coherence). The contrast issue was associated with light gray descriptive text on the landing page and blue text within an informational

box that may not provide sufficient contrast against their backgrounds. Because precise color values cannot be extracted from screenshots, the framework classified these cases as potential risks requiring manual verification. The second issue concerned weak visual grouping within the settings form, where sections such as *Profile Information*, *Emergency Contact*, and *Insurance Information* relied mainly on whitespace and headings rather than stronger visual containers.

In the second run, only the U6 layout grouping issue was detected. The contrast-related warning identified in the first run was not reported in this evaluation. In the third run, both A1 and U6 were again detected.

The slight variation observed for A1 can be explained by the characteristics of screenshot-based evaluation. Since screenshots do not expose underlying style information, contrast assessment depends entirely on the vision LLM's perceptual interpretation. Consequently, each detection carries a degree of uncertainty. To reduce hallucinations in this modality, the framework applies a confidence threshold of 60%. Findings below this threshold are filtered out. It is therefore possible that the A1 detection fell below this threshold during the second run, resulting in its omission.

ZIP-Based LLM-Assisted Evaluation

To complement the screenshot analysis, a second stability test was conducted using the ZIP source code of the same CareBridge application. In this configuration, only the rules that primarily rely on LLM-assisted reasoning (U4, U6, E1, and E2) were enabled in order to isolate the behavior of the LLM-assisted detection layer when grounded in deterministic source-code evidence.

Table 5.9 presents the results of the repeated ZIP-based evaluations.

Table 5.9: Consistency of detected violations across repeated ZIP-based evaluation runs for LLM-assisted rules.

Rule	Run 1	Run 2	Run 3	Frequency
U4 - Recognition–Recall Balance	✓	✓	✓	3 / 3

The repeated ZIP-based evaluations produced identical results across all three runs. A single potential usability issue corresponding to U4 (Recognition–Recall Balance) was consistently detected. No additional LLM-assisted rules were triggered in any run.

Comparative Analysis

Comparing the two evaluation conditions reveals an important difference in stability. The screenshot-based evaluation exhibited minor variability in the A1 contrast detection, while the ZIP-based evaluation produced identical results across all runs.

This difference is explained by the role of deterministic preprocessing in the source-based evaluation pipeline. In the ZIP modality, deterministic extractors first analyze the source code and generate structured evidence describing interface properties such as layout organization and interaction patterns. The LLM then interprets this evidence rather than performing unrestricted analysis. This grounding mechanism reduces uncertainty and improves reproducibility.

In contrast, screenshot-based evaluation relies entirely on vision-based LLM interpretation because no deterministic engine is available for image inputs. As a result, every detection carries a degree of perceptual uncertainty. The confidence-gating mechanism helps mitigate hallucinations by filtering out low-confidence outputs, but small variations across runs remain possible.

Overall, the experiment demonstrates that the framework maintains high stability in LLM-assisted evaluations. While purely perceptual analysis may introduce minor variability, LLM-assisted rules grounded in deterministic source-code evidence exhibit fully reproducible results across repeated runs. These findings support the reliability of the framework's hybrid evaluation design.

5.5 Summary

This chapter presented the experimental evaluation of the UI Critic framework through four experiments examining its behavior, reliability, and applicability. The cross-tool comparison showed that different generative development platforms produce varying interface quality, while corrective iterations enable convergence toward improved designs. The modality consistency test demonstrated that code-based inputs provide more stable structural analysis, whereas screenshot evaluation offers complementary insight into visual usability issues. The real-world GitHub evaluation confirmed the framework's ability to detect accessibility, usability, and ethical design issues in both AI-generated and human-developed applications. Finally, the AI-as-a-Judge sensitivity analysis indicated that LLM-assisted detections remain generally stable, particularly when grounded in deterministic preprocessing.

Chapter 6

Conclusion and Future Work

This chapter concludes the thesis by summarizing the main contributions of the proposed UI Critic framework and discussing their broader implications for the evaluation of AI-generated user interfaces. In addition to reflecting on the results obtained from the experimental evaluation, the chapter examines the limitations of the current approach and outlines several directions for future research. Together, these discussions provide a comprehensive perspective on the practical applicability of the framework and highlight opportunities for further development of automated interface evaluation methods.

6.1 Summary of Contributions

This thesis introduced UI Critic, a modality-aware evaluation framework designed to assess AI-generated user interfaces across accessibility, usability, and ethical design dimensions. The framework combines deterministic static analysis with LLM-assisted perceptual assessment, enabling the analysis of web interfaces through multiple input modalities, including screenshots, ZIP source code archives, and public GitHub repositories.

UI Critic implements a structured rule taxonomy consisting of fifteen rules spanning accessibility (A1–A6), usability (U1–U6), and ethical design (E1–E3). The system produces automated violation reports using an epistemic classification model that distinguishes between Confirmed violations, derived from deterministic analysis of source code, and Potential risks, identified through perceptual reasoning when deterministic verification is not possible. This classification approach supports conservative and interpretable reporting while still capturing interface issues observable through visual inspection.

The framework was evaluated through a series of experiments designed to assess its effectiveness and robustness. First, AI-generated applications produced by

generative development platforms were analyzed to examine baseline interface quality and common violation patterns. In addition, corrective prompts generated from detected violations were used to iteratively refine the generated interfaces. The results showed that confirmed violations were successfully resolved within two iterations, demonstrating that the framework can support automated improvement of generated interfaces. Second, a modality consistency experiment compared evaluation results across different input types, highlighting how outcomes may vary depending on whether the interface is analyzed through source code or screenshots. Third, a real-world GitHub evaluation applied the framework to both AI-generated and human-developed web applications. Finally, an AI-as-a-Judge sensitivity experiment examined the stability of LLM-assisted detections across repeated screenshot-based evaluations.

Overall, the results demonstrate that the proposed framework can effectively identify accessibility, usability, and ethical design issues across different types of web interfaces. The study also shows that generative development tools are capable of producing functional user interfaces but still require systematic accessibility auditing, particularly when interfaces are generated automatically without explicit accessibility constraints. More broadly, the findings suggest that automated evaluation frameworks can support early detection and iterative improvement of design issues in AI-generated interfaces, providing structured insights that complement traditional manual reviews.

6.2 Implications

The results of this study have several implications for both research and practice in the development and evaluation of AI-generated user interfaces.

First, the findings highlight the growing need for automated evaluation tools capable of assessing accessibility and usability in AI-generated interfaces. As generative development platforms increasingly automate UI creation, developers may rely on generated interfaces without fully verifying compliance with accessibility standards. The results of this thesis demonstrate that AI-generated interfaces can still exhibit accessibility and usability shortcomings, indicating that automated auditing tools such as UI Critic can play an important role in supporting developers during early stages of interface development.

Second, the experiments demonstrate the value of combining deterministic analysis with LLM-assisted reasoning. Deterministic rule evaluation provides reliable detection of structural accessibility violations, while LLM-assisted analysis enables the identification of perceptual and contextual interface issues that are difficult to detect through static analysis alone. This hybrid approach illustrates how rule-based and AI-driven evaluation methods can complement each other in

automated interface assessment systems.

Third, the results emphasize the importance of modality-aware evaluation when analyzing modern web interfaces. Different input representations provide different types of information about the interface. Source-code–based analysis enables deterministic inspection of structural properties such as semantic markup and form labels, while screenshot-based evaluation reveals perceptual layout and visual design issues that may not be directly observable in the source code. Supporting multiple evaluation modalities therefore increases the coverage and reliability of automated interface analysis.

Finally, the findings also suggest that automated evaluation frameworks can support iterative improvement of generated interfaces. The corrective prompt experiments demonstrate that detected violations can be translated into actionable guidance, allowing generative development tools to refine their outputs through iterative prompting. This indicates that automated evaluation systems may serve not only as auditing tools but also as feedback mechanisms that help improve the quality of AI-generated interfaces.

Overall, these implications suggest that modality-aware automated evaluation frameworks can contribute to improving the accessibility and usability of modern web interfaces, particularly in development environments where user interfaces are increasingly generated or assisted by AI systems.

6.3 Limitations

Despite the contributions of this study, several limitations should be acknowledged.

First, the experimental evaluation was conducted on a limited number of web applications. Although the selected systems included both AI-generated and human-developed projects, the sample size remains relatively small and may not fully represent the diversity of real-world web interfaces. Evaluating the framework on a broader range of applications and domains would help further validate its generalizability.

Second, part of the evaluation process relies on LLM-assisted reasoning, particularly when analyzing screenshots or other perceptual representations of the interface. While the sensitivity experiment demonstrated generally stable results across repeated runs, perceptual analysis may still introduce minor variability due to the probabilistic nature of large language models.

Third, certain accessibility and usability properties cannot be fully verified through static analysis alone. Some issues depend on runtime behavior, dynamic interface states, or user interaction flows that may not be observable directly from source code or static screenshots. As a result, the framework primarily focuses on structural and visually observable issues rather than fully dynamic interface

behavior.

Another limitation arises from the rule-based nature of the evaluation framework. The UI Critic system relies on heuristics derived from accessibility standards and usability guidelines. While these rules provide a structured and interpretable foundation for automated analysis, they cannot fully capture the complexity of user experience or context-specific design decisions. Consequently, some interface issues may still require human judgment and qualitative evaluation beyond the scope of automated rule-based detection.

Finally, the corrective prompt experiments were conducted on a limited number of generative development tools and iterations. Although the results demonstrate that detected violations can guide iterative interface improvements, further investigation is needed to evaluate how such feedback mechanisms perform across a wider range of AI development systems and more complex application scenarios.

Despite these limitations, the results demonstrate that the proposed framework provides a practical and extensible approach for automated evaluation of accessibility, usability, and ethical design issues in modern web interfaces.

6.4 Future Work

Several directions can be explored to extend the capabilities of the UI Critic framework and further improve automated evaluation of modern web interfaces.

First, future work could expand the evaluation framework to support simultaneous multi-modal analysis. In the current implementation, each evaluation run processes a single input modality, such as screenshots, ZIP source code archives, or GitHub repositories. Combining multiple modalities within a unified evaluation pipeline could enable cross-validation of detected violations and improve the reliability of automated interface analysis by integrating both structural and perceptual evidence.

Second, the framework could be extended to incorporate runtime interaction analysis. While the current approach focuses primarily on static source code inspection and perceptual screenshot analysis, certain accessibility and usability issues emerge only during dynamic user interactions, such as keyboard navigation behavior, animation states, or asynchronous feedback mechanisms. Integrating runtime testing or browser-based interface inspection could improve the detection of interaction-related issues.

Third, the rule taxonomy implemented in UI Critic could be expanded and refined. Although the current system evaluates fifteen rules derived from accessibility standards, usability heuristics, and ethical design principles, additional rules could be introduced to cover a broader range of interface design considerations. Future work may also explore adaptive rule prioritization or context-aware rule evaluation

based on interface characteristics.

Another promising direction involves integration of automated evaluation within development workflows. The framework could be incorporated into continuous integration (CI) pipelines or development environments, enabling automatic UI auditing during application development. Such integration would allow developers to receive early feedback about accessibility and usability issues as interfaces are created or modified.

Finally, future research could investigate the use of evaluation frameworks such as UI Critic as feedback mechanisms for generative UI development systems. The corrective prompt experiments in this study demonstrate that detected violations can guide iterative improvements in AI-generated interfaces. Expanding this approach to support automated refinement loops between evaluation systems and generative design tools could further improve the quality of AI-generated user interfaces.

Overall, these directions suggest that modality-aware automated evaluation frameworks have the potential to play an important role in supporting the design, generation, and continuous improvement of accessible and usable web interfaces.

Bibliography

[1] G. Paliwal, A. Donvir, P. Gujar, and S. Panyam. «Low-code/no-code meets GenAI: A new era in product development». In: *IEEE 8th Ecuador Technical Chapters Meeting (ETCM)*. Cuenca, Ecuador, 2024 (cit. on pp. 1, 10, 20).

[2] Y. Liu, B. Yang, M. Lee, and J.-B. Martens. «Low-code conversation-based hybrid UI design: A case study and reflection». In: *Chinese CHI 2023 (CHCHI 2023)*. Denpasar, Bali, Indonesia, 2023 (cit. on p. 1).

[3] D. Liu, J. He, S. Guo, Y. Chen, and L. Qiao. «What's wrong with low-code development platforms? An empirical study of low-code development platform bugs». In: *IEEE Transactions on Reliability* 73 (2024), pp. 695–708 (cit. on pp. 1, 10, 12, 14).

[4] A. P. Velásquez, C. Pérez-Salazar, L. A. Hernández-González, and Á. J. Sánchez-García. «Systematic literature review of low-code and its future trends». In: *International Conference on Software Engineering Research and Innovation (CONISOFT)*. Puerto Escondido, Mexico, 2024 (cit. on pp. 1, 10, 21, 22).

[5] Y. Gui et al. «UICopilot: Automating UI synthesis via hierarchical code generation from webpage designs». In: *ACM Web Conference (WWW)*. Sydney, Australia, 2025 (cit. on pp. 1, 7, 12, 14).

[6] P. Vaithilingam, E. Glassman, J. Inala, and C. Wang. «DynaVis: Dynamically synthesized UI widgets for visualization editing». In: *CHI Conference on Human Factors in Computing Systems (CHI '24)*. Honolulu, HI, USA, 2024 (cit. on pp. 1, 7).

[7] Y. Zhang, J. Liu, T. Zhang, X. Chen, W. Guo, and X. Chen. «MultiML: A machine learning model construction tool based on multidimensional views». In: *IEEE International Conference on Data Intelligence and Innovative Application (DIIA)*. 2024, pp. 1–7 (cit. on p. 1).

[8] A. Leung, R. Cheng, J. Wu, J. Nichols, and T. Barik. «SQUIRE: Interactive UI authoring via slot query intermediate representations». In: *Proceedings of the ACM on Human-Computer Interaction* 9.CSCW1 (2025), p. 167. DOI: 10.1145/3746059.3747672 (cit. on pp. 1, 7, 13, 23).

[9] Z. Chen, C. Wang, W. Sun, G. Yang, X. Liu, J. Zhang, and Y. Liu. *Promptware Engineering: Software Engineering for LLM Prompt Development*. arXiv:2503.02400 2025 (cit. on pp. 1, 12).

[10] B. Kitchenham and S. Charters. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report. Keele University and Durham University, 2007 (cit. on pp. 3, 4, 6).

[11] D. Moher, A. Liberati, J. Tetzlaff, D. G. Altman, and PRISMA Group. «Preferred Reporting Items for Systematic Reviews and Meta-Analyses: The PRISMA Statement». In: *PLoS Medicine* 6 (2009), e1000097 (cit. on pp. 4, 5).

[12] T. Zhou, Y. Zhao, X. Hou, X. Sun, K. Chen, and H. Wang. «DeclarUI: Bridging Design and Development with Automated Declarative UI Code Generation». In: *Proceedings of the ACM on Software Engineering* 2 (2025), pp. 219–241. DOI: 10.1145/3715726 (cit. on pp. 7, 11–13, 23).

[13] F. Wu, C. Gao, S. Li, X. Wen, and Q. Liao. *MLLM-Based UI2Code Automation Guided by UI Layout Information*. arXiv:2506.10376. 2025 (cit. on pp. 7, 11, 12).

[14] Y. Lu, A. Leung, A. Swearngin, J. Nichols, and T. Barik. *Misty: UI Prototyping through Interactive Conceptual Blending*. arXiv:2409.13900. 2024 (cit. on pp. 7, 12).

[15] C. Kamnerddee, P. Putjorn, and J. Intarasirisawat. «AI-Driven Design Thinking: A Comparative Study of Human-Created and AI-Generated UI Prototypes for Mobile Applications». In: *International Conference on Intelligent Computing and Technologies (InCIT)*. 2024, pp. 237–242. DOI: 10.1109/InCIT63192.2024.10810565 (cit. on pp. 7, 18, 23, 24).

[16] S. Feng, M. Jiang, T. Zhou, Y. Zhen, and C. Chen. *Auto-Icon+: An Automated End-to-End Code Generation Tool for Icon Designs in UI Development*. arXiv:2204.08676. 2022 (cit. on p. 7).

[17] J. Lee, G. Oh, J. Ahn, and X. Qiu. *ReDemon UI: Reactive Synthesis by Demonstration for Web UI*. arXiv:2507.10099. 2025 (cit. on p. 7).

[18] P. Duan, C.-Y. Cheng, G. Li, B. Hartmann, and Y. Li. «UICrit: Enhancing automated design evaluation with a UI critique dataset». In: *CHI Conference on Human Factors in Computing Systems*. 2024 (cit. on pp. 7, 12, 14).

[19] T. Huang, C. Yu, W. Shi, Z. Peng, D. Yang, W. Sun, and Y. Shi. «Prompt2Task¦ Automating UI Tasks on Smartphones from Textual Prompts». In: *ACM Transactions on Computer-Human Interaction* 32 (2025). DOI: 10.1145/3716132 (cit. on pp. 7, 11, 12).

[20] M. Monteiro, B. Branco, S. Silvestre, G. Avelino, and M. Valente. «NoCodeGPT¦ A No-Code Interface for Building Web Apps with Language Models». In: *Software: Practice and Experience* 55 (2025), pp. 1408–1424. DOI: 10.1002/spe.3432 (cit. on pp. 7, 19, 23).

[21] E. Owen and K. Ajeigbe. *Generative AI for Enhanced User Interface (UI) Design*. Unpublished work. 2023 (cit. on p. 7).

[22] A.-E. Guriță and R.-D. Vatavu. «Good Accessibility, Handcuffed Creativity: AI-Generated UIs Between Accessibility Guidelines and Practitioners' Expectations». In: *ACM Designing Interactive Systems Conference (DIS)*. 2025, pp. 1197–1209. DOI: 10.1145/3715336.3735691 (cit. on pp. 7, 17, 23).

[23] M. A. Shahab. «The Impact of Generative AI on UI Design for Mobile Applications in Terms of Efficiency Using the SPACE Framework». Master's Thesis. Tampere, Finland: Tampere University, 2024 (cit. on pp. 7, 18, 23).

[24] I. Abu Doush and R. Kassem. «Evaluating AI-Generated Web Code for Accessibility Compliance: A Metric-Driven Approach». In: *Proceedings of the ACM International Conference on Web Engineering*. 2025, pp. 338–344. DOI: 10.1145/3696593.3696614 (cit. on pp. 7, 19, 23).

[25] M. Fischer and C. Lanquillon. «Evaluation of Generative AI-Assisted Software Design and Engineering: A User-Centered Approach». In: *Lecture Notes in Computer Science*. Vol. 14734. Cham, Switzerland: Springer, 2024, pp. 31–47. DOI: 10.1007/978-3-031-60606-9_3 (cit. on pp. 7, 19).

[26] S. A. Sumit. «The Future of No-Code Web Development: Evaluating the Potential and Limitations of Webflow». Bachelor's Thesis. Helsinki, Finland: Haaga-Helia University of Applied Sciences, 2025 (cit. on pp. 10, 11).

[27] L. Zhang. «Evaluating the Merits of the Low-Code/No-Code Paradigm from the Perspective of ISO 25010 Quality Requirements». Bachelor's Thesis. Lappeenranta, Finland: Lappeenranta–Lahti University of Technology (LUT), 2024 (cit. on pp. 10, 11, 22).

[28] A. Viljoen, B. Stelzl, M. Yang, J. Nguyen, A. Hein, E. Elshan, and H. Krcmar. «Navigating Flexibility and Standardisation in Low-Code/No-Code Development». In: *Information Systems Journal* 36 (2025), pp. 95–109. DOI: 10.1111/isj.70001 (cit. on p. 10).

[29] B. Binzer, E. Elshan, D. Fuerstenau, and T. Winkler. «Establishing a Low-Code/No-Code-Enabled Citizen Development Strategy». In: *MIS Quarterly Executive* 23 (2024), pp. 253–273. DOI: 10.17705/2msqe.00097 (cit. on p. 10).

[30] O. Abahussain and J. Al-Ammary. «Low-Code/No-Code Platforms: Impact and Concerns». In: *Proceedings of the International Conference on IT Innovation and Knowledge Discovery (ITIKD)*. Manama, Bahrain, 2025 (cit. on pp. 10, 11, 22).

[31] D. De Silva, R. Shangavie, and R. Ranathunga. «Role of Quality Assurance in Low-Code/No-Code Projects». In: *Proceedings of the International Conference on Information and Knowledge Integration (ICOIN)*. 2024, pp. 789–794. DOI: 10.1109/ICOIN59985.2024.10572203 (cit. on pp. 10, 11).

[32] Y. Wan, C. Wang, Y. Dong, W. Wang, S. Li, Y. Huo, and M. Lyu. «Divide-and-Conquer: Generating UI Code from Screenshots». In: *Proceedings of the ACM on Software Engineering* 2 (2025), pp. 2099–2122. DOI: 10.1145/3729364 (cit. on pp. 11–13).

[33] Y. Cheng, J. Chen, Q. Huang, Z. Xing, X. Xu, and Q. Lu. «Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains». In: *ACM Transactions on Software Engineering and Methodology* 33 (2023). DOI: 10.1145/3638247 (cit. on p. 12).

[34] B. Naqvi, D. Kedziora, L. Zhang, and S. Oyedeji. «Quality of Low-Code/No-Code Development Platforms through the Lens of ISO 25010:2023». In: *Computer* 58 (2025), pp. 30–40 (cit. on p. 12).

[35] J. Cui. «The Impact of Low-Code and No-Code Programming on Software Product Delivery Quality and Development Efficiency». In: *SSRN* (2024). DOI: 10.2139/ssrn.5010766 (cit. on pp. 12, 22).

[36] H. Khalajzadeh and J. Grundy. «Accessibility of Low-Code Approaches: A Systematic Literature Review». In: *Information and Software Technology* 177 (2024), p. 107570. DOI: 10.1016/j.infsof.2024.107570 (cit. on p. 14).

[37] P. Somer. «Algorithmic Accountability of Low-Code/No-Code Artificial Intelligence: A Literature Review». In: *Proceedings of the Americas Conference on Information Systems (AMCIS)*. Montréal, Canada, 2025 (cit. on p. 15).

[38] H. El Kamouchi, M. Kissi, and O. El Beggar. «Low-Code/No-Code Development: A Systematic Literature Review». In: *International Conference on Innovative Trends in Information Technology (SITA)*. 2023, pp. 1–8. DOI: 10.1109/SITA60746.2023.10373712 (cit. on p. 15).

[39] D. Gao and F. Fagerholm. «Measuring End-User Developers' Episodic Experience of a Low-Code Development Platform». In: *e-Informatica Software Engineering Journal* 18 (2024), p. 240105 (cit. on p. 15).

[40] N. Roy, O. Horielko, and O. Omojokun. «Benchmarking of Generative AI Tools in Software Engineering Education: Formative Insights for Curriculum Integration». In: *Proceedings of the ACM Conference on International Computing Education Research (ICER)*. Charlottesville, VA, USA, 2025. DOI: 10.1145/3702653.3744328 (cit. on p. 15).

[41] J. Lively, J. Hutson, and E. Melick. «Integrating AI-Generative Tools in Web Design Education: Enhancing Student Aesthetic and Creative Copy Capabilities Using Image- and Text-Based AI Generators». In: *DS Journal of Artificial Intelligence and Robotics* 1 (2023), pp. 23–33. DOI: 10.59232/AIR-V1I1P103 (cit. on p. 16).

[42] H. Xu and X.-Y. Yu. *From PowerPoint UI Sketches to Web-Based Applications: Pattern-Driven Code Generation for GIS Dashboard Development Using Knowledge-Augmented LLMs, Context-Aware Visual Prompting, and the React Framework*. arXiv:2502.08756. 2025 (cit. on pp. 16, 23).

[43] V. Dibia, J. Chen, G. Bansal, S. Syed, A. Fourney, E. Zhu, C. Wang, and S. Amershi. *AutoGen Studio: A No-Code Developer Tool for Building and Debugging Multi-Agent Systems*. arXiv:2408.15247. 2024 (cit. on pp. 16, 23).

[44] Y.-F. Liu, M. I. Luthfi, and W.-Y. Hwang. «Enhancing Usability and Learner Engagement: A Heuristic Evaluation of the AI-Enhanced Video Drama Maker App». In: *Joint International Conference on Computer Science, Software Engineering and Applications (JCSSE)*. 2024, pp. 337–342 (cit. on p. 17).

[45] A.-E. Guriță. «Understanding and Improving Accessibility in AI-Generated Interfaces through Interactive Prompt Engineering Methods». In: *ACM Conference on Human Factors in Computing Systems (CHI)*. 2025, pp. 101–104. DOI: 10.1145/3708557.3716347 (cit. on pp. 17, 23).

[46] J. Iman, A. Felisha, M. Kimeison, E. Hermawan, R. Rumagit, and H. Pranoto. «Refining UI/UX with Minimalist Design and AI: Towards Sustainable and Efficient Digital Experiences». In: *Procedia Computer Science* 269 (2025), pp. 669–680. DOI: 10.1016/j.procs.2025.09.010 (cit. on pp. 18, 23).

[47] T. Guthardt, J. Kosiol, and O. Hohlfeld. «Low-Code vs. the Developer: An Empirical Study on the Developer Experience and Efficiency of a No-Code Platform». In: *Proceedings of the ACM International Conference on Software Engineering*. 2024, pp. 856–865. DOI: 10.1145/3652620.3688332 (cit. on p. 20).

[48] S. Liu, H. La, A. Willms, and R. Rhodes. «A "No-Code" App Design Platform for Mobile Health Research: Development and Usability Study». In: *Journal of Medical Internet Research* 24 (2022), e38737. DOI: 10.2196/38737 (cit. on p. 20).

[49] D. Pinho, A. Aguiar, and V. Amaral. «What About the Usability in Low-Code Platforms? A Systematic Literature Review». In: *Journal of Computer Languages* 74 (2023), p. 101185. DOI: 10.1016/j.cola.2022.101185 (cit. on pp. 20, 23).

[50] K. Rokis and M. Kirikova. «Exploring Low-Code Development: A Comprehensive Literature Review». In: *Complex Systems Informatics and Modeling Quarterly* (2023), pp. 68–86. DOI: 10.7250/csimq.2023-36.04 (cit. on pp. 21, 22).

[51] M. Ajimati, N. Carroll, and M. Maher. «Adoption of Low-Code and No-Code Development: A Systematic Literature Review and Future Research Agenda». In: *Journal of Systems and Software* (2024), p. 112300. DOI: 10.1016/j.jss.2024.112300 (cit. on pp. 21, 22).

[52] F. Sufi. «Algorithms in Low-Code/No-Code for Research Applications: A Practical Review». In: *Algorithms* 16 (2023), p. 108. DOI: 10.3390/a16020108 (cit. on pp. 21, 23).

[53] N. Upadhyaya. *Low-Code/No-Code Platforms and Their Impact on Traditional Software Development: A Literature Review*. Unpublished work. 2023. DOI: 10.13140/RG.2.2.31585.72807 (cit. on p. 22).

[54] Gizem Irmak. *UI Critic: A Framework for Evaluating AI-Generated User Interfaces*. https://github.com/gizem-irmak/ui-critic. GitHub repository, accessed 2026. 2026 (cit. on p. 49).